

Fast Navigation through an FRep Sculpture Garden

Maxim Kazakov,
Moscow Institute of Physics and Technology,
max_kazakov@hotmail.com,

Alexander Pasko,
Hosei University,
pasko@k.hosei.ac.jp,

Valery Adzhiev
Bournemouth University
vadzhiiev@bournemouth.ac.uk

Abstract

The function representation (FRep) allows for construction of quite complex shapes such as isosurfaces of real-valued functions composed using the functionally defined primitives and operations. Calculating such functions in the complex cases can be very time-consuming. Extraction and visualization of isosurfaces for them can hardly be imagined interactive. In this paper we present a method of an interactive navigation through a "sculpture garden" containing non-intersecting FRep objects defined in the terms of specialized high-level language HyperFun. Before the actual isosurface extraction and visualization occurs, the objects are voxelized on a regular 3D grid with a possibility of a further adaptive voxelization. Polygonization employs a hierarchical representation of the voxelized data and a view-dependent isosurface reconstruction at the different levels of detail. To speedup the extraction process, an isosurface is constructed only in the visible part of the dataset with its updates performed incrementally as observer moves. Due to low preprocessing costs required for isosurface mesh construction, it is possible to visualize time-dependent objects, if hardware is capable to calculate updates in real time.

Keywords: Interactive Visualization, Shape Modeling, Isosurface Polygonization, Implicit Surfaces, FRep.

1. Introduction

Implicit surfaces are usually considered slow to render. The generalized model called the function representation (Frep) [13] makes the problem even more difficult. The real-valued function at the given point is evaluated by a procedure traversing the tree structure with functionally defined primitives as leaves and operations on them as nodes. It allows for representing quite complex shapes by continuous real-valued functions with the time consuming evaluation. In this paper, we deal with the described problem and present a method of the interactive real-time navigation of a moving camera through a scene consisting of several static or time-dependent FRep objects. We use a special type of scene called a "sculpture garden" where the objects do not intersect and are located inside individual bounding boxes. Each FRep object is defined using a high-level modeling language HyperFun [1].

One of the approaches to accelerate visualization of an implicit surface (or more precisely an isosurface of a real-valued function) is to convert it to a polygonal mesh and to use a

standard polygon rendering hardware. Existing polygonal isosurface construction methods are generally based on the Marching Cubes (MC) algorithm [10], which performs triangulation for each cell of a scalar volume dataset thus reconstructing the entire isosurface. We propose to initially voxelize the FRep object at the preprocessing stage and apply an adaptive function sampling during the navigation. Then, a polygonization algorithm can be applied to the generated scalar dataset.

Straightforward polygonization of the entire volume associated with the scene makes an interactive visualization impossible due to an overwhelming number of generated polygons. To handle such a mesh complexity several mesh simplification methods can be employed on the post-processing step [4,5,6,8,9,14,16,18] but it does not help to reduce the mesh construction time generally determined by the dataset dimensions. The latter issue is of a concern for the methods that construct hierarchies of datasets with successively smaller dimensions that are employed for a coarser isosurface representation [15,17]. Some of them are based on the octree approach where each of the eight neighboring cells corresponding to octree nodes can be recursively combined into a larger cell that corresponds to the parent octree node.

One controls a level-of-detail (LOD) for the resulting surface by selecting a hierarchy level for cells to be processed during an isosurface construction. This idea had its further refinement in [17] where the isosurface is constructed with the dataset hierarchy level varying over the dataset parts thus producing varying level-of-detail in the resulting surface. The distance to the observer and local surface curvature determines a hierarchy level selected for a particular dataset part. Other selection criteria can be applied as well.

The construction of isosurface patches at different dataset hierarchy levels imposes problems with an isosurface continuity [6]. This issue was addressed in [6, 17] where a special stitching procedure for the isosurface patches extracted from the different hierarchy levels was proposed.

More simple way to reduce the amount of data for processing is to limit it by the volume visible for the camera. We combine this clipping with an adaptive LOD surface generation to achieve interactivity in visualization of our scenes.

The correspondence between a grid cell and a triangle patch in the isosurface mesh exists due to the Marching Cubes algorithm

organization. This correspondence greatly facilitates visualization of changes in the grid values. This allows for not only direct user-driven modifications of the scalar function values in the grid nodes (resulting in mesh updates) but visualization of time-dependent FRep objects as well. As recalculation of the function defining an FRep object may be very expensive, we propose to employ adaptive function sampling coupled with the current level of detail of the isosurface mesh related to the object position. This can lower the number of recalculations required to render modifications in FRep objects.

The paper layout is as follows: in Section 2 we explain FRep, HyperFun language and the way FRep objects defined in HyperFun become accessible for the navigation procedure. We review the hierarchical construction approaches for faster extraction and visualization of isosurfaces in Section 3. Section 4 addresses stitching isosurface patches generated at different levels of hierarchy. Section 5 describes how we employ view dependency to minimize processing time and an amount of polygons produced during isosurface construction. Section 6 explains visualization of time-dependent FRep objects. The paper is finalized with results described in Section 7. The conclusions come along with the discussion of application areas and plans for the future in Section 8.

2. Function representation and HyperFun

Traditionally, implicit surfaces include algebraic and skeleton-based surfaces. Advantages of using implicit surfaces in computer graphics and solid modeling are well known: let us mention here an ease of point classification and ray-tracing, and natural blending [3]. However, the modeled shapes are largely limited by organically looking blobby objects called also metaballs and soft objects. The function representation (or FRep) is a generalization of traditional implicit surfaces and constructive solid geometry (CSG). In FRep, an object is represented by a tree structure, where leaves are arbitrary "black box" primitives defined by real-valued functions and nodes are arbitrary operations yielding functions as the result. Function evaluation procedures traverse the tree and evaluate the function value at any given point. This model makes it possible to represent by a single continuous function such different objects as traditional skeleton-based implicit surfaces, convolution surfaces, constructive (CSG) solids (using so-called R-functions [13] in the nodes of the tree), swept objects, and volumetric objects. Many operations are closed on this representation, i.e., generate an object defined by a continuous function, which again can be a subject for further transformations. In this sense, in FRep there is no difference between soft objects, CSG solids, and volumetric (voxel) objects that are processed in the same manner.

HyperFun [1] is a high-level specialized language for a parametrized description of functionally-based multidimensional geometric shapes. While being minimalist and suitable for easy mastering, it supports all main notions of FRep. The language has special built-in operators for fundamental set-theoretic operations and exploits functions from an extensible FRep

library that contains the most common primitives and transformations of a quite broad spectrum. Application software deals with HyperFun models through a built-in interpreter (implemented as a small set of functions in ANSI C) or by using HyperFun-to-C compiler and utilities of the HyperFun API. In this work, the latter way has been used because the C code generated ensures more effective function evaluation.

HyperFun can be used to model multidimensional objects that then are mapped to a "multimedia space" for further perception by the user [1]. For example, a 4D object can be mapped to 3D world coordinates x , y , z , and a dynamic variable ("abstract time"). By default, the fourth coordinate in HyperFun models is mapped to the dynamic variable and then with some additional scaling to the physical time of navigation.

To compose a scene for navigation one can use any HyperFun object constructed by available tools. The scene description includes the list of HyperFun objects and corresponding bounding boxes. The bounding box of the scene can be explicitly given or calculated from the given bounding boxes of the included objects. This bounding box is used for further voxelization and motion control during the navigation.

In our work, before actual rendering takes place the objects modeled in HyperFun pass several stages described below:

- Generation of the C code that implements a scalar function for each FRep object described in HyperFun. At this stage the HyperFun-to-C compiler is employed.
- The generated C code is compiled by a plain C compiler and linked with the HyperFun libraries implementing functions for primitive FRep objects and operations. It results in a server application that is accessed by clients to retrieve information on FRep objects in the scene and to query values of the functions defining FRep objects in a set of points.
- Visualization application uses server application to voxelize each FRep object in its own bounding box. Bounding box information is used to re-sort an order of voxelization to minimize the possibility of one object to wipe out another during voxelization. To reduce calculation overhead during voxelization of the time-dependent objects (they are re-voxelized for each rendering frame) an adaptive sampling for distant objects can be employed.
- The visualization application processes dataset produced as a voxelization result to extract and polygonize the isosurface within the visible volume.
- For time-dependent FRep objects, the visualization application may query the server application for the current values of corresponding functions supplying time as one of coordinates to them. Updated results are used to modify the isosurface mesh.

3. Hierarchical isosurface construction

The hierarchical approach to the isosurface extraction implies usage of several datasets originated from the source dataset to construct successively coarser representations. Some methods construct such a dataset hierarchy applying the low-pass filter to the source data several times and then adaptively process smoothed dataset regions extracting the isosurface [6,17]. Although creating appealing visualization results, those methods require complex and lengthy filtering of the dataset.

Another approach used to construct such a dataset hierarchy employs subsampling. In this case the even samples along each direction are employed to create a subsampled dataset representing another level of hierarchy.

FRep objects have their peculiarity in that originally they are mathematically defined so one can prepare a simplified version of the function that defines an object. This simplified version can be employed for the construction of the coarser representation of the object. It can remedy visual artifacts that are inherent in subsampling and eliminate the necessity to support several datasets that cache successively coarser object representations.

If cells in the datasets from neighboring hierarchy levels differ for two times in size in each direction, it is easy to construct an octree over their cells. As one cell contains eight smaller ones from the neighbor hierarchy level, the corresponding octree node refers to eight other nodes that correspond to the smaller cells. Octree nodes may carry some information used during the isosurface extraction. In our case, the main task is checking a presence of an isosurface patch in a cell. When using the octree structure, presence of the node can indicate presence of the surface patch in the corresponding cell. In this way, the isosurface can be constructed by simply traversing the nodes presented in the octree and pruning traversal as the required level-of-detail is reached.

As shown in [17], usage of more than two or three detailing levels and corresponding dataset hierarchy levels results in distracting visual artifacts during the isosurface construction. So, the support for all $\log_2 N$ hierarchy levels (where N is a number of samples along a direction for the original dataset) is excessive. Moreover, the straightforward octree organization where each node requires up to eight (possibly 4-byte) pointers to its children might be memory consuming. We propose another hierarchical data structure that holds information about a surface patch presence in a dataset cell. For each dataset hierarchy level a bitmask is created and supported. Each cell containing an isosurface patch itself or in one of children cells has corresponding “one” in the bitmask while others have “zeroes” in it. Such bitmask adds memory overhead no more than 12,5 % for each dataset hierarchy level, if one byte is used for scalar value and even less if four-byte floats or eight-byte doubles are used. The memory overhead imposed by the bitmask allows for checking an isosurface patch presence in a cell from any hierarchy level of interest using a single bitmask lookup instead

of traversing of several octree levels. Moreover, a number of operations required for the bitmask construction does not exceed the corresponding number for the octree construction.

Summarizing above, we can say that our approach, while being based on hierarchical isosurface extraction ideas as illustrated in [17], uses its own hierarchical data structures that speed up checking an isosurface patch presence in any cell from any dataset hierarchy level.

4. Stitching procedure

Straightforward usage of different dataset hierarchy levels for a LOD isosurface reconstruction leads to certain problems with continuity in the resulting polygonal mesh [6]. As stated in [17], it is due to scalar field discontinuities introduced by subsampling or low-pass filtering used to construct the levels of dataset hierarchy. Cracks in the surface produced at the boundary of dataset regions with different detailing are illustrated at Fig. 1. That is why after applying regular MC-like method for the hierarchical isosurface construction, an additional stitching procedure should be executed for elimination of possible cracks.

This procedure may modify isosurface patch at finer detailing level, coarser detailing level [6] or at both. Our approach employs fine-detailed patch modifications and assumes that neighboring cells are reconstructed using the same or neighbored dataset hierarchy levels (i.e., they differ for no more than two times in their dimensions).

Let us consider the rectangle ABCD that is on the boundary between regions with different detailing (Fig. 1). Linear interpolation of values at corresponding edge centers A' , B' , C' , D' between values at vertices of the hosting edges AB, BC, CD, DA leads to patches coincidence at the mentioned edges causing points P'' and Q'' where the isocurve intersects the edges to become points P' and Q' . This does not eliminate cracks in the internals of ABCD [17] because the isocurve is approximated by the straight line at the coarser detailing level while being a broken line at the finer one. That is where our stitching procedure is applied. It is proposed to straighten (if necessary) the isocurve at the finer detailing level by moving isocurve vertices that lay inside the rectangle ABCD to their neighbors on the rectangle edges (moving P to P' and Q to Q'). After these modifications the finer isocurve completely coincides with the coarse one.

Because the result of the stitching procedure is ruled only by data from the coarser level, the original value at the center of ABCD is of no concern. However, it is required for the initial construction of the finer isocurve. Actual value at the central vertex is of no concern but the choice of its sign should be ruled by a disambiguation method [11,12] employed in polygonization. In our work, the “preferred polarity” [2] approach is used, so the value sign in the face center is chosen to comply with the “polarity” used during the cell polygonization.

Actually, only sample signs at the face corners as well as interpolated samples signs in the centers of the face edges determine all vertex replacements performed during stitching. Thus, all possible (valid) combinations of the signs can be easily summarized in a lookup table to speedup the stitching process.

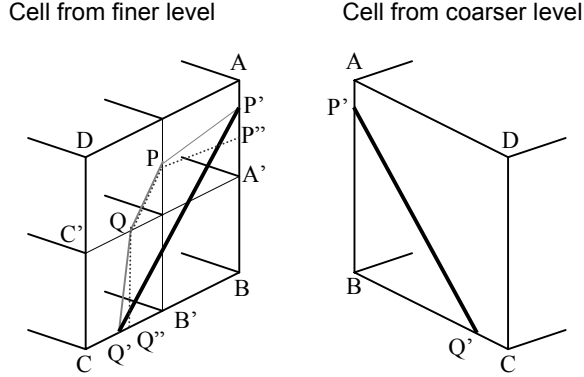


Fig. 1. ——— P'Q' - isocurve at coarser level
 P''P''Q''Q'' - isocurve at finer level
 ——— P'P''Q''Q' - isocurve at finer level with interpolated values at A',B',C' and D'

5. View-dependent isosurface reconstruction

Knowledge about the observer position during visualization helps in performing certain optimizations of the visualization process. As it was already mentioned in Section 2, the isosurface in different volume parts can be refined adaptively according to some criteria. Distance to the current observer position provides one of such criterion that allows us to leave distant surface parts at coarser levels of detailing thus reducing the number of constructed polygons.

Sometimes distant details can be of more interest than closer ones especially when the observer wants to explore features of a particular object part. In this case, the observer position can easily be replaced with the position of some exploring tool that refines the near parts of the isosurface. This criterion is employed in [17] where the following correspondence between distance to the point of interest and the recommended detailing level is proposed:

$$Level = \left\lceil 1 + \log_2 \left(\frac{s+d}{3 \cdot d} \right) \right\rceil (1),$$

where $Level$ is a recommended dataset hierarchy level, $\lceil x \rceil$ is an integer less or equal to x , s is a distance to the cell center and $d = \sqrt{3}$ is a diagonal length of the level 0 cell. This formulation assumes that cells from different hierarchy levels used for the isosurface reconstruction have their edge length depending on the hierarchy level chosen as follows:

$$Length = 2^{level}, \quad level \in [0..N] (2),$$

where $level$ is a hierarchy level, N is a maximal hierarchy level employed, zero hierarchy level corresponds to the original dataset.

Another issue is that the camera frustum formed by clipping planes bounds a visible volume region that is only a fraction of the entire volume. One can restrict the isosurface construction process to be performed only in the visible volume part. This condition should be considered for a substantial decrease of processing time in applications where the either observer does not see the entire dataset at a time or this situation is rather rare [7].

Both issues mentioned above can be considered along with an isosurface mesh reuse and its incremental updates caused by changes in the observer position and the dataset modification to reduce processing times and the number of polygons. Mesh cache or reuse is one of the major differences of our method from [17] where the mesh is reconstructed from the scratch for each frame.

Supporting incremental mesh updates for the moving observer results in several difficulties in its implementation. First, level-of-detail changes in the considerable number of cells after the observer makes a step in the scene or changes a point of his/her interest. This requires finding those cells and updating isosurface patches in them according to a new detailing level. Next, movement of the camera frustum requires processing the certain dataset parts for the isosurface extraction and merging with the existing mesh while eliminating isosurface patches associated with the cells that leave the frustum.

5.1 Initial isosurface construction

Before the construction of the initial visible isosurface part, it is necessary to initialize data structures required for the hierarchical isosurface reconstruction, e.g., the surface containment bitmasks for each dataset hierarchy level of interest. This initialization starts from cells of the original dataset where each cell is examined for presence of an isosurface patch in it. The checking succeeds if dataset samples at the cell vertices have different signs resulting in "one" set in the corresponding place in the bitmask for the original dataset level. Otherwise, "zero" is set.

Next, the bitmasks for the remaining hierarchy levels are initialized. This initialization requires only the bitmask from the previous level to be available as "one" is set when one of the child cells from the previous level has "one" set in the bitmask from their level.

The initial observer position and viewing direction along with depth of a view control the viewable volume part. For simplicity, this volume part is approximated with an axis-aligned box that contains it. The isosurface construction starts with covering this box by cells from the coarsest dataset hierarchy level. Each cell

that has an isosurface patch in it (that is detected by inspecting the corresponding bitmask) is checked if its level is equal to one given by (1). If it is the case, the cell gets triangulated and triangles are added to the resulting isosurface mesh. Otherwise, the cell is processed recursively until the level recommended by (1) is met. During the triangulation, the stitching procedure (see Section 4) is applied, if necessary. Information about non-empty cells that meet the criterion (1) is inserted into the hash table for future usage. This hash table allows for easy lookup for cell information by its position and level. If a surface patch was constructed for a cell, then cell information in the hash table is updated with the reference to the triangles constituting the patch.

This initialization activity results in the isosurface mesh constructed for the visible part of the scene with detailing levels depending on the distance to the observer or a point of observer's interest. Another result is the prepared internal structures (such as the bitmasks and the hash table) that will be used for handling of future mesh updates with the observer movement and dataset modifications.

5.2 Processing incoming volume parts

As the observer moves, both position and size of the closest axis-aligned bounding box containing the camera frustum change. Let us denote this bounding box at the previous camera position as Ω_{prev} and Ω_{curr} at the current position. The volume that comes into the camera frustum box at the current frame is proportional to $\Omega_{curr} \setminus \Omega_{prev}$ and it can be divided up into several axis-aligned boxes $B_1..B_n$. For each such box we apply the procedure similar to one described in the previous subsection to construct the new isosurface part. The only difference comes from the following consideration. When the previous isosurface part was constructed, the cells intersecting its box boundary were possibly added. The isosurface construction for the new volume regions should account for those cells and omit from processing those ones intersecting the common boundary of Ω_{prev} and Ω_{curr} . The result of applying this procedure is illustrated in Fig. 2.

5.3 Updating the isosurface within the camera frustum

Besides adding isosurface parts for volume regions incoming into the camera frustum box, the camera movement forces modifications in the isosurface part that has already been constructed.

First, for the cells leaving the bounding box of the camera frustum corresponding triangular patches should be removed from the isosurface mesh. Second, for the cells that remain in the bounding box the detailing level recommended by (1) may change so they (and isosurface patches within them) should be replaced with ones satisfying (1). Third, the changed detailing

level in the above-mentioned cells requires updates in surface stitching with their neighbors.

The solution for all those tasks is based primarily on a content of the hash table described in Section 4.1. This hash table contains information concerning all non-empty cells that met criterion (1) and were within the bounding box of the camera frustum for the previous observer position. By traversing the list of cells for this hash table, one can perform all update activities mentioned above.

All cells in the list that are not contained by the current bounding box are erased from the hash table and corresponding triangular patches are removed from the isosurface mesh.

The cells that do not satisfy (1) (patches implied) are replaced by their children or parent cells depending on increase or decrease of the recommended detailing level.

Boundary conditions determined by neighborhood with cells from another hierarchy level are cached in cell information in the hash table for easier detection of their changes. If such changes take place then cell's triangles are dropped and new

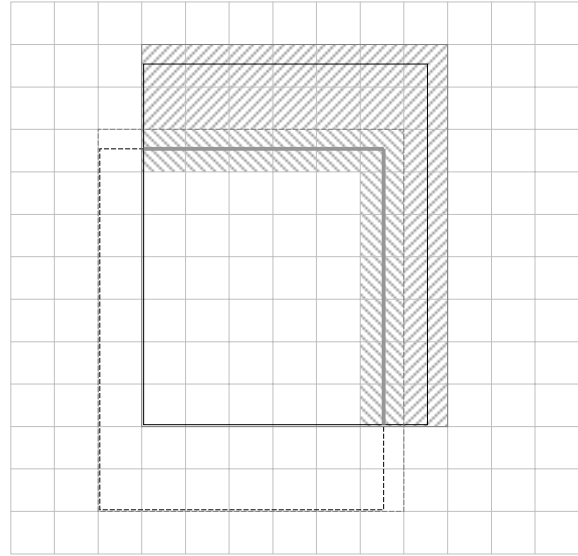


Fig. 2. bounds Ω_{prev} , bounds cells that cover Ω_{prev} , bounds Ω_{curr} , bounds cells that cover Ω_{curr} , is a common part of Ω_{curr} and Ω_{prev} boundary, marks actually processed cells, marks cells omitted from processing.

ones are created and stitched with neighbor cells' patches.

Accomplishing update tasks means that both isosurface mesh and hash table are consistent with the observer position and

direction of sight, the hash table is ready to handle future updates in the observer parameters.

6. Visualization of time-dependent objects

There exist two ways for the time dependence to appear in FRep objects in our work. First one is rather obvious when a function that defines an FRep object has time dependency introduced via one of point coordinates. Another way is when the dataset containing the result of the static object voxelization is directly modified (“carved”) by the user interactively changing shape of an FRep object. In any case, time dependency leads to updates of scalar values in the grid nodes.

Those updates of the dataset have several follow-ups to be considered. If implementation employs support of simplified datasets, it may require updates in them for other levels of the dataset hierarchy. It is not the case if those datasets are just results of subsampling of the source dataset or when a simplified FRep object function is used. When these datasets are obtained from the original dataset by a low-pass filtering (like in [17]), then this filtering procedure must be applied once again. This filtering requires reasonable computational resources and results in noticeable processing time. That is why in our implementation filtering is not supported.

Another issue is related to the necessity for updates in the internal structures employed during the isosurface construction and visualization. In our case those structures are the bitmaps for each dataset hierarchy level and the cells hash table, if changes occur in the visible dataset part. Finally, updates in the dataset possibly cause modifications in the isosurface mesh. Once again it is the case only if the visible portion of the dataset is modified.

In our current implementation subsampling is used for creation of the coarser isosurface construction, and employment of simplified FRep objects is planned. These conditions determine the following update procedure: completed dataset modifications make possible the updating of the bitmaps for each dataset hierarchy level. During bitmask update, examining updates in the masks for each cell that contains updated sample eases checking if a sample update in a dataset requires the corresponding update in the constructed isosurface mesh. If the mesh update is necessary and the node is within visible dataset part then the hash table is used to lookup all non-empty cells that contain this node and related triangular patches. All patches are updated according to changes in the isosurface caused by the sample modification in the dataset. As this update procedure ends in for each update in the dataset, then both internal structures and the isosurface mesh reflect the actual object and may be employed for handling the future updates in the dataset.

Some inefficiency in this implementation comes from the consideration that update of one dataset sample may require as many as eight cell updates. If neighbor samples are modified, then cells that share them will be processed twice during update. To reduce this overhead it is proposed to cache dataset updates

based on the spatial sample proximity and perform cell update for a bunch of dataset sample updates.

7. Experimental results

To demonstrate the results in the interactive visualization of the FRep models that are possible with our approach, we use a scene that contains one FRep object based on the several samples from the HyperFun gallery (see <http://www.hyperfun.org/>) – Pet Bottle, Bunny, Toy and Faucet. Several tests were conducted to reveal the implementation performance in the different usage scenarios. A Pentium III @ 500 MHz machine with a Matrox G400 graphics accelerator was used as a test platform.

7.1 Navigation in a static scene

The static scene was chosen to demonstrate the dependence between overall visualization performance in terms of frames per second (fps) and a number of detailing levels used when reconstructing the isosurface. Three tests were conducted: with LOD disabled (the scene is visualized with one finest detailing level), with two LOD levels used and, finally, with three LOD levels. In all three cases the depth of view was 100 samples determining a visible volume domain to be approximately 100 samples in each direction.

Fig. 3 shows a typical snapshot for the scene rendered with disabled LOD. The amount of frames per second (8-10 fps) is really low due to the substantial number of triangles forming the scene. Navigation in this case can hardly be called “interactive”.

Fig. 4 shows a snapshot when two LOD levels are employed. Adaptive detailing reduces the number of triangles in several times making navigation much more responsive (25-30 fps).

Fig. 5 demonstrates a third case when three LOD levels are enabled to simplify the extracted isosurface mesh. While the performance is rather high (about 40 fps), it leads to the crude appearance of distant objects and noticeable visual artifacts on the boundaries of the isosurface parts with different detailing levels. As was mentioned in Section 3, the latter is due to the isosurface discontinuity caused by subsampling. Visual appearance could be much better, if we use filtering of simplified versions of the functions defining FRep objects. The situation is also remedied a bit with usage of a detail refinement tool shown as a small sphere. The user can control the detailing of a scene portion of interest by approaching this tool to it.

7.2 Carving

In our implementation, the carving tool is coupled with the detail refinement tool thus allowing the user to see finer isosurface visualization results near the point of possible modifications. Spherical shape of the carving tool can be easily replaced with something more complex and is chosen for the simplicity of the demonstration. During carving the user can control position and size of the carver simultaneously thus

making rather complex modifications of original scene possible. We process the entire volume covered by the bounding box of the carving tool. Varying size of the carving tool affects the processing load associated with handling of the dataset updates. In our experiments updates cover subvolumes differing from $14 \times 14 \times 14$ samples to $26 \times 26 \times 26$ samples in size resulting in 2700 – 18000 modified samples per frame.

Fig. 6 illustrates the typical result of carving on the Toy and Pet objects from the scene. Fig. 7 shows dependency of achieved performance on the number of samples modified for each frame. As calculation of the function that defines the spherical carving tool is very cheap compared to the processing of updates, this dependency demonstrates advantages of the visualization algorithm in handling updates of the dataset that holds voxelization result for the visualized scene. As Fig. 6 shows, a satisfactory responsiveness during visualization of the updates in the static scene is limited by a 6000 - 8000 sample updates per frame when we can neglect the function calculation cost for updates.

7.3 Dynamic scene visualization

Visualization of time-dependent FRep objects in our implementation differs from an interactive carving only in that it requires lengthy calculations to be performed for each updated sample. As it is hard to predict which FRep object part change for an arbitrary object, the entire volume associated with object's bounding box is resampled.

Existing C-language API for HyperFun allows the shape server application to reside in a different process and even on a different machine. To effectively parallelize calculation of updates and visualization, the shape server needs to cache calculations somewhere to be quickly returned to the visualization client on its demand. However, in the current implementation the calculation of updates is performed in the same process that visualizes FRep objects.

Fig. 8 shows animation results for a complex object. This example of the time-dependent object is a metamorphosis between two 3D puzzles, one is constructed of three 3D Roman letters «R», «O» and «S», another one is constructed of two 3D Chinese characters «Ni» and «Hon» and represents the word "Japan". This example was constructed to demonstrate an automatic metamorphosis between FRep objects with differing topology. The changes of the function values cannot be localized and require resampling of function values in all grid nodes. Bounding box for metamorphosis object has 30 samples in each dimension leading to 27000 sample updates handled per frame. As recalculation performed during visualization appeared too lengthy and froze the graphic activity at one frame in 6-10 seconds or more we used an array of precalculated object function samples for 100 time steps of morphing. It helped the visualization rates to rise to somewhat more comfortable 3-4 frames per second.

8. Conclusion and future work

In this work, we demonstrate the possibility of interactive navigation in a scene constructed of FRep objects. Existing HyperFun tools allow for conversion of a textual description of FRep objects into the dedicated shape server applications accessible by the visualization client. Shape server information about each FRep object including the defining function allows us to voxelize the scene and update the samples of a time-dependent objects. Voxelization results are used for the isosurface reconstruction and visualization.

To reduce the number of triangles produced during polygonization, this work employs a multiresolution dataset hierarchy that makes interactive navigation possible. Usage of a frame-to-frame coherence along with clipping the processing to the camera viewing frustum constitutes the difference of our approach from analogous isosurface visualization methods. Reuse of the existing isosurface mesh minimizes dataset processing during an interactive navigation. The adaptive level-of-detail isosurface reconstruction based on the multiresolution dataset hierarchy keeps the number of polygons in the isosurface mesh moderate for interactive visualization with sufficient detailing in surface areas of interest for observer. The incremental mesh detailing updates performed as the observer moves allow for keeping mesh up-to-date at a little processing cost.

The algorithm described in this work has built-in support for visualization of modifications in a dataset that keeps scene voxelization results. It allows us to implement an interactive carving over the visualized scene and to work with time-dependent FRep objects that are re-voxelized for each frame.

Besides FRep object visualization our method can be employed in any field where interactive visualization of the isosurfaces in the scalar datasets may be interesting – like visualization of the medical data (MRI scans, CT scans), scientific applications (visualization of the calculation results) and even computer games if FRep will be used for description of scene geometry in them.

Issues that remain to be addressed are implementation of adaptive sampling for time-dependent FRep objects and employment of filtering of the sampled function for coarser levels of details that improves visual appearance of the extracted isosurface. Both are subjects of our future work on the improvement of the presented approach.

9. References

- [1] V. Adzhiev, R. Cartwright, E. Fausett, A. Ossipov, A. Pasko, V. Savchenko " HyperFun project: a framework for collaborative multidimensional FRep modeling", *Implicit Surfaces '99, Eurographics/ACM SIGGRAPH Workshop*, J. Hughes and C. Schlick (Eds.), pp. 59-69. (see <http://www.hyperfun.org>)

- [2] J. Bloomenthal. "Polygonalization of Implicit Surfaces", *Computer Aided Geometric Design*, vol. 5, 1988, pp. 341-355
- [3] *Introduction to Implicit Surfaces*, Ed. J. Bloomenthal, Morgan Kaufmann, 1997.
- [4] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. P. Brooks Jr. and W. V. Wright. "Simplification Envelopes", *Computer Graphics, Proc. Ann. Conf. Series (Proc. Siggraph'96)*, 1996, pp. 119-128
- [5] M. Ech, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsberry, and W. Stuetzle. "Multiresolutional Analysis of Arbitrary Meshes", *Computer Graphics Proc. Ann. Conf. Series (Proc. Siggraph'95)*, 1995, pp. 173-182
- [6] T. He, L. Hong, A. Varshney, S. Wang. "Controlled Topology Simplification", *IEEE Trans. on Visualization & Computer Graphics*, vol. 2, no. 2, 1996, pp. 171-184
- [7] L. Hong, S. Muraki, A. Kaufman, D. Bartz, T. He, "Virtual Voyage: interactive voyage in human colon", *SIGGRAPH'97, Computer Graphics Proceedings*, 1997, pp.27-34
- [8] H. Hoppe. "Progressive Meshes", *Computer Graphics*, Vol. 30.
- [9] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. "Mesh Optimization", *Computer Graphics Proc. Ann. Conf. Series (Proc. Siggraph'93)*, 1993, pp. 19-26
- [10] W. Lorensen, H. Cline. "Marching Cubes: A High Resolution 3D Surface Construction Algorithm", *Computer Graphics*, vol. 21, no. 4, 1987, pp. 163-169
- [11] G. Nielson, B. Hamann. "The Asymptotic Decider: Resolving the Ambiguity in Marching Cubes", *Proc. Visualization '91*, pp. 83-91, Oct. 91
- [12] A. Pasko, V. Pilyugin, V. Pokrovskiy, "Geometric modeling in the analysis of trivariate functions", *Computers and Graphics*, 12 (3/4), 457-465 (1988).
- [13] Pasko A., Adzhiev V., Sourin A., Savchenko V. "Function representation in geometric modeling: concepts, implementation and applications", *The Visual Computer*, vol.11, No.8, 1995, pp.429-446.
- [14] J. Rossignac, P. Borrel. "Multi-Resolution 3D Approximations for Rendering Complex Scenes", *Modeling in Computer Graphics*, Springer-Verlag, June-July 1993, pp. 455-465
- [15] W. J. Schroeder, J. A. Zarge, and W. E. Lorensen. "Decimation of Triangle Meshes", *Computer Graphics*, vol. 26, no. 2, 1992, pp. 65-70 (Proc. Siggraph'92)
- [16] G. Turk. "Re-Tiling Polygonal Surfaces", *Computer Graphics*, vol. 26, no. 2, pp. 55-64, 1992 (Proc. Siggraph'92)
- [17] R. Westermann, L. Kobbelt, T. Ertl. "Real-time Exploration of Regular Volume Data by Adaptive Reconstruction of Iso-Surfaces", *The Visual Computer*, (1999) 15, pp.100-111
- [18] J. C. Xia, J. El-Sana, A. Varshney. "Adaptive Real-Time Level-of-Detail-Based Rendering for Polygonal Models", *IEEE Trans. on Visualization & Computer Graphics*, vol. 3, no. 2, 1997, pp. 171-183

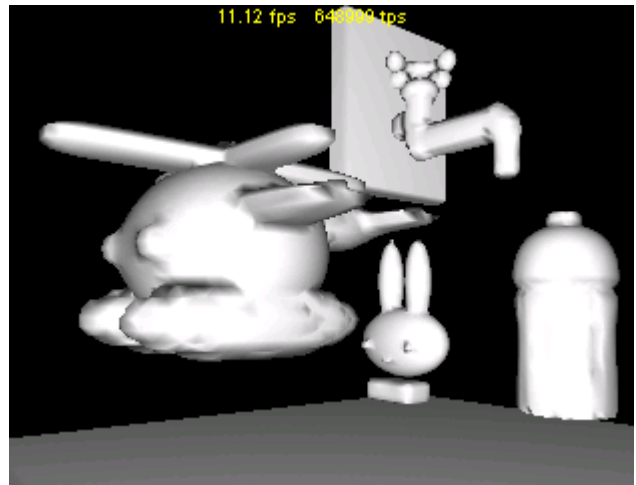


Fig. 3a Results of scene rendering with LOD disabled. Visible isosurface mesh has 58358 triangles from 29199 cells.

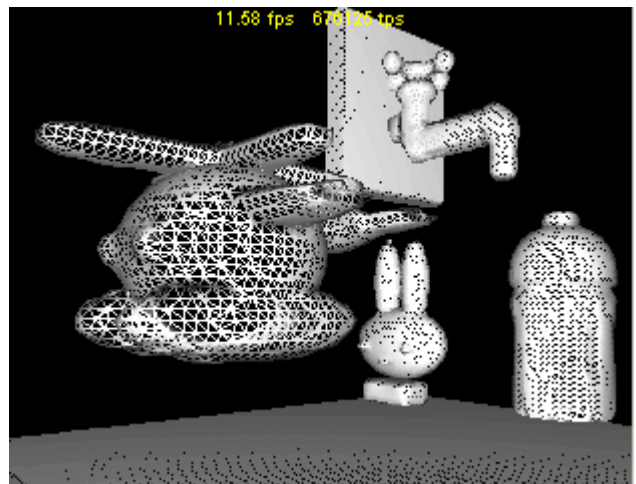


Fig. 3b Wireframe model of the scene shown at Fig. 3a

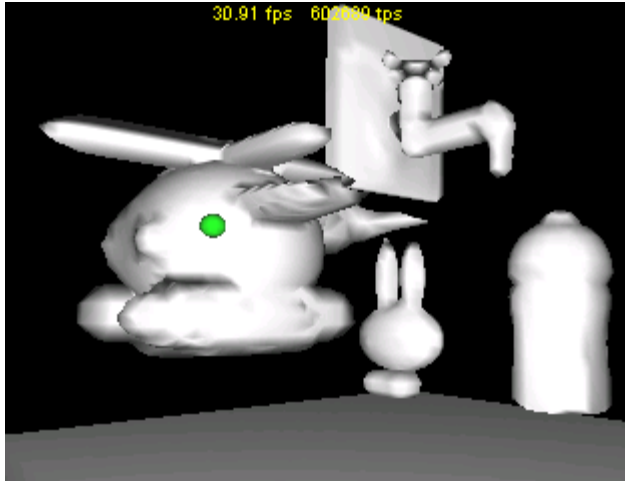


Fig. 4a Results of scene rendering with 2 LOD levels employed. Visible isosurface mesh consists of 16004 triangles extracted from 8036 dataset cells.

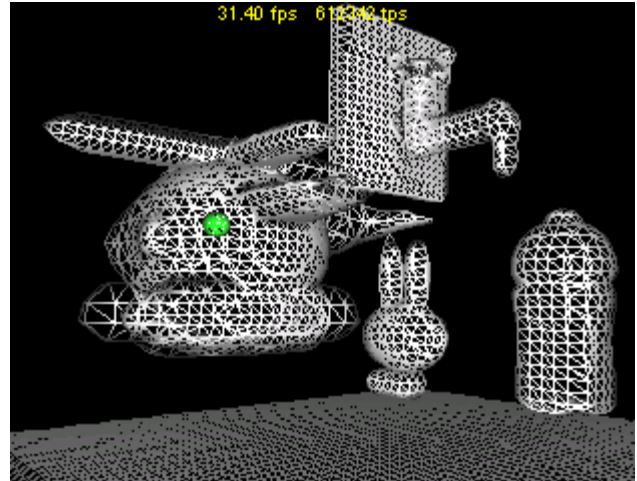


Fig. 4b Wireframe model of the scene shown at Fig. 4a.

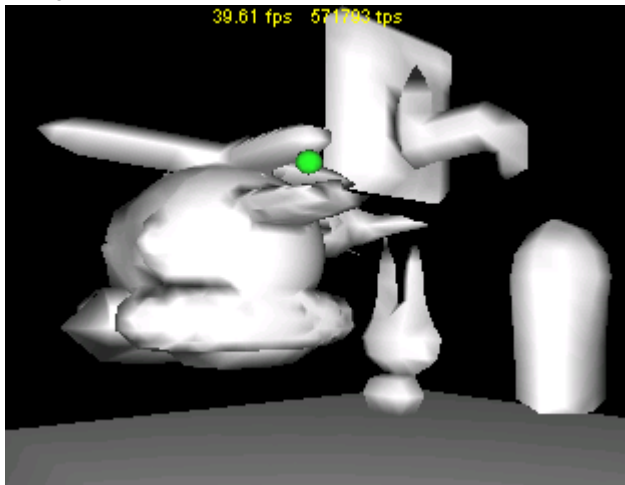


Fig. 5a Results of scene rendering with 3 LOD levels employed. Visible isosurface mesh has 5873 triangles from 2991 cells.

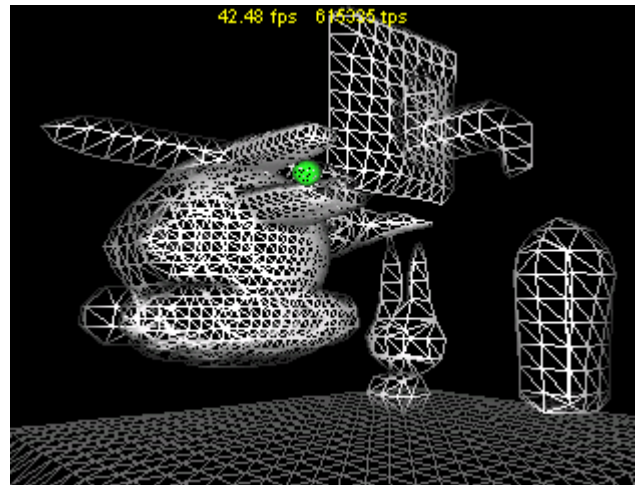


Fig. 5b Wireframe model of the scene shown at Fig. 5a. Sphere depicts the detail refinement tool.

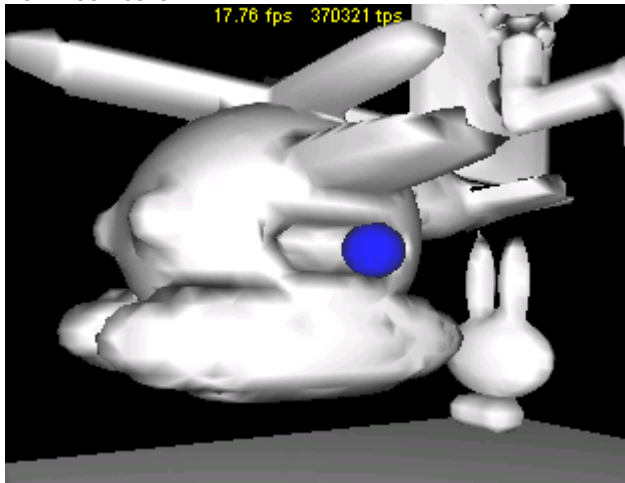


Fig. 6a Carving - sphere depicts the detail refinement tool in a carving mode. Carver radius is 2.

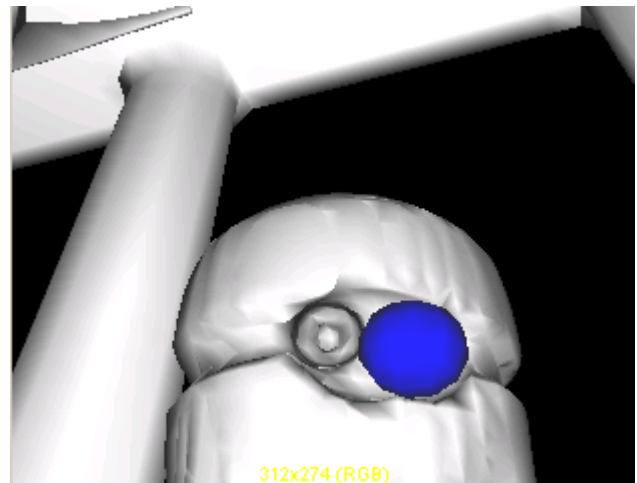


Fig. 6b Carving – carver radius is 10.

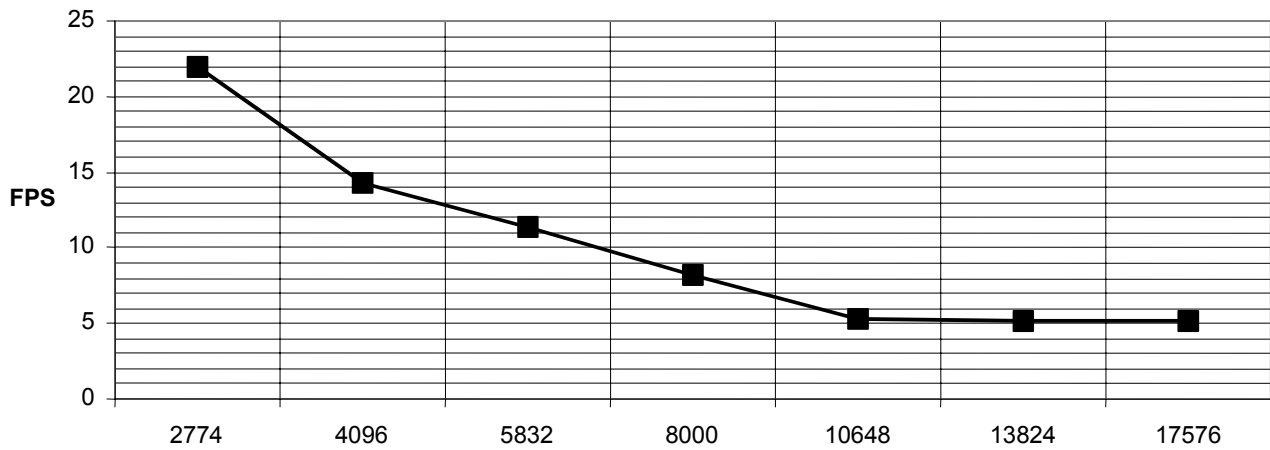


Fig. 7 Dependency between frames per second and a number of sample updates per frame

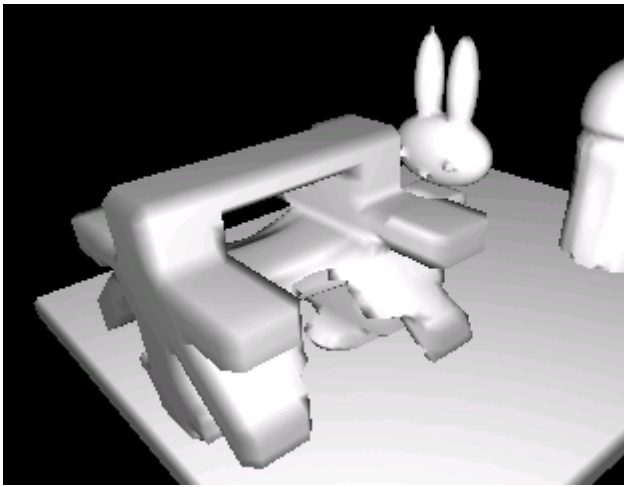


Fig. 8a Visualization of time-dependent metamorphosis.

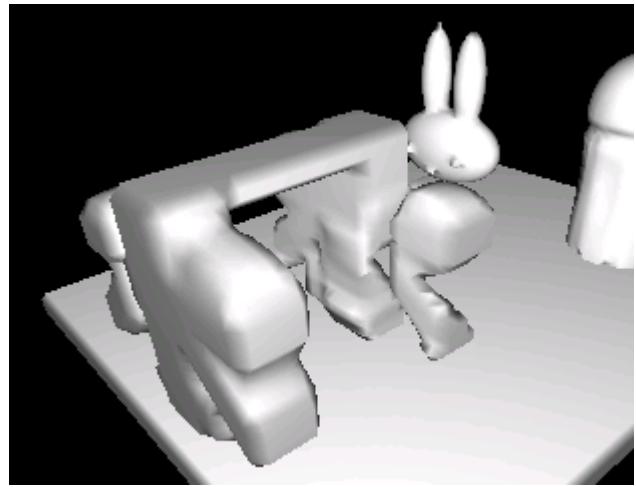


Fig. 8b Time-dependent metamorphosis – interim stage.

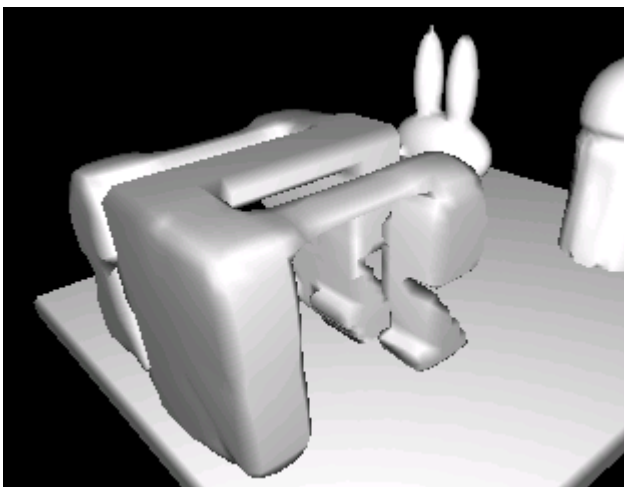


Fig. 8c Time-dependent metamorphosis – interim stage.

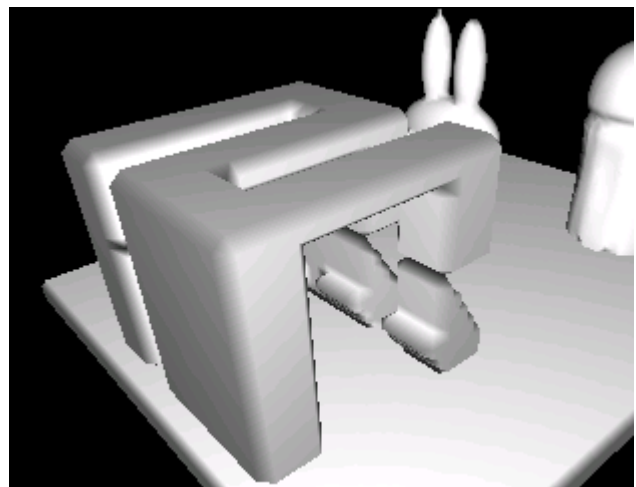


Fig. 8d Time-dependent metamorphosis – final stage.