Efficient Point-Based Isosurface Exploration Using the Span-Triangle

Bartosz von Rymon-Lipinski* Nils Hanssen* Thomas Jansen* Lutz Ritter* Erwin Keeve*

Surgical Systems Laboratory, research center caesar

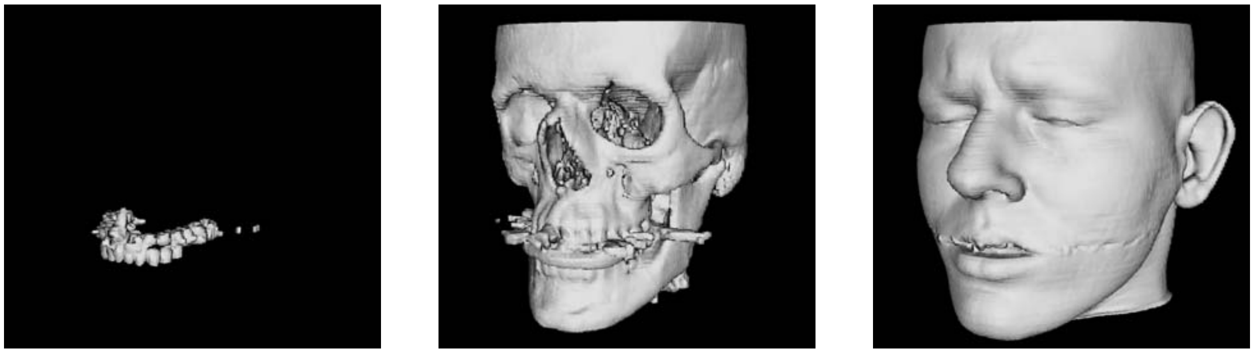

Figure 1: Isosurface exploration of a CT volume model, showing different anatomical structures of a human head

Abstract

We introduce a novel *span-triangle* data structure, based on the span-space representation for isosurfaces. It stores all necessary cell information for dynamic manipulation of the isovalue in an efficient way. We have found that using our data structure in combination with point-based techniques, implemented on graphics hardware, effects in real-time rendering and exploration. Our extraction algorithm utilizes an incremental and progressive update scheme, enabling smooth interaction without significant latency. Moreover, the corresponding visualization pipeline is capable of processing large data sets by utilizing all three levels of memory: disk, system and graphics. We address practical usability in actual medical applications, achieving a new level of interactivity.

CR Categories: I.3.6 [Computer Graphics]: Methodology and Techniques—Graphics data structures and data types; I.3.3 [Computer Graphics]: Picture/Image Generation—Display Algorithms; I.3.5 [Computer Graphics]: Computational Geometry and Object Modelling—Curve, surface, solid, and object representations

Keywords: Point-Based Visualization, Isosurfaces, Hardware Acceleration, Large Data Set Visualization, Visualization in Medicine

1 Introduction

Visualization plays an important role in many scientific and medical applications, facilitating the understanding of three-dimensional volume data. A crucial part is the exploration of the underlying spatial scalar field, targeting at characteristic structures inside solid bodies, like bone or soft tissue in medical data sets. One common approach is direct volume rendering, where the user gets insight vision into a representation of the complete volume. Popular techniques are ray-casting, shear-warp, texture-mapping and splatting [18]. These techniques can be used to display directly surfaces in the volume by utilizing appropriate ray-intersections or by applying transfer functions that correspond to a single spike in value space. On the other hand, many applications require a geometric representation of a surface model. Therefore, an alternative approach is indirect isosurface visualization. An isosurface is a surface-based representation, generated from geometric primitives that are associated with a constant scalar value. Unfortunately, achieving interactive exploration of the underlying volume data with isosurfaces is still a challenge. Many solutions are based on a separation between the generation of surface primitives and the actual rendering. Typically, the generation is performed for each new isovalue and previous geometry is discarded. Such an approach requires a complete recalculation of the model. Additionally, many approaches use triangle models to represent surfaces. In a standard triangle-based method, like Marching Cubes, volume grid cells that are intersected by the isosurface are identified, polygonized and added to the output mesh [17]. The resulting model is built from a set of interconnected triangles. This may lead to high memory consumption, making the isosurface exploration and memory management a hard challenge.

To overcome the aforementioned performance and memory drawbacks we have developed a novel data structure - the span-triangle - derived from the *span-space* representation [16]. It stores relevant cell information for all isosurface models in a user-specified scalar range. Given a specific isovalue, we can extract the corresponding model with optimal time complexity $O(k)$, for output size k. Our incremental and progressive update scheme exploits data coherence between different models and allows smooth manipulation of the isovalue in real-time. We have chosen point primitives to represent and display geometry within the rendering pipeline. Point elements do not contain connectivity information and therefore can be managed efficiently in applications that work with dynamic data. Moreover, representing objects with points enables the possibility to sample the original data only at scalar regions that contain relevant surface information. We have implemented a point-based renderer that is accelerated with recent features of graphics hardware [2]. Therefore, we are able to consider a visualization pipeline, which is distributed among three levels of memory, i.e. disk, system and graphics, supporting data sets with more than 512^3 resolution.

*e-mail:{lipinski,hanssen,jansen,ritter, keeve}@caesar.de

IEEE Visualization 2004
October 10-15, Austin, Texas, USA
0-7803-8788-0/04/$20.00 ©2004 IEEE

In the following we provide the notation and technical terms, used in this paper: We consider a volume as a set of *samples* in three dimensional space. Each sample is associated with a scalar intensity value $x \in \Re$ in the range $[r_{min}, r_{max}]$. Applying a reconstruction function, e.g. trilinear interpolation, results in the corresponding *scalar field* function $V : \Re^3 \rightarrow \Re$. Therefore, having an isovalue $\upsilon \in \Re$ we can define the corresponding isosurface as the set of points $\{\mathbf{x} \in \Re^3 \mid V(\mathbf{x}) = \upsilon\}$. Additionally, neighboring samples of the volume specify a subvolume, called *cell*. For example, a cell in a three-dimensional volume grid is constructed by eight corner samples with scalar values $x_1, ..., x_8 \in \Re$. The *cell range* $[x_{min}, x_{max}]$ is defined, using the minimum and maximum intensity values. A cell is called *active*, if it is intersected by the isosurface, i.e. $\upsilon \in [x_{min}, x_{max}]$. Finally, the *span* is the width of the corresponding interval, i.e. $s = x_{max} - x_{min}$.

2 Related Work

A lot of research is focusing on the improvement of isosurface generation performance. This is achieved mainly by reducing the number of cells visited during the extraction or by providing an efficient access to active cells. There are three different types of techniques: domain search, range search and cell propagation. Domain search techniques work in the geometric space. One example is the hierarchical partitioning of the spatial domain. Wilhelms and van Gelder use an octree with intensity extrema values to avoid the traversal of inactive cells, i.e. that do not intersect with the isosurface [25]. This approach is useful for interactive visualization, but performance is dependent strongly on the data set. Other research has concentrated on processing the scalar space. Typically, data structures are defined that contain cell information, sorted by the cell-minimum and -maximum scalar values. Examples are active lists, sweeping simplices and interval trees [8, 23, 5]. Other range search algorithms are based on the the span-space representation. Using this approach, the isosurface generation can be reduced to a range searching problem. The acceleration is achieved by subdividing the span-space, using data structures such as the kd-tree and lattice decomposition [16, 22]. In general, range search methods can be orders of magnitude faster than Marching Cubes, but they suffer from the common problem of excessive extra memory requirements [24]. The basic idea of cell propagation techniques is to grow the isosurface from an initial set of cells. Known solutions are the extrema graph and seed sets [11, 3]. Finally, recent methods address also the problem of isosurface exploration. The *Isoslider* system is based on the idea that small changes in isovalue require small changes to active cells [4]. Their algorithm uses a triangle-based approach, resulting in high-quality rendering, but long preprocessing times even for moderate data sizes.

Point-based techniques have been proposed by Levoy and Whitted in their technical report from 1985 [15]. New approaches of rendering point primitives have become popular, especially since the work of Grossmann and Dally, who use hierarchical z-buffers to fill holes between point primitives in screen space [9]. Further approaches like QSplat and surfels use splatting techniques to represent and render surfaces by displaying point primitives as small discs or ellipsoids that are scaled as a function of the distance to the eye [20, 19, 26]. Alexa et al. use adaptive resampling of the point set to match the output resolution on the image plane [1]. Recent algorithms incorporate also point-based techniques for isosurface rendering. Bærentzen et al. are not focusing on interactive isosurface exploration, but have shown that speed of hardware-accelerated point-rendering can be superior to triangle rendering [2]. Furthermore, *Iso-Splatting* uses approximate projection of points on the isosurface to improve visual quality of point-based isosurface representations [6].

The development of our data structure was inspired by the work of Gallagher and by the ISSUE algorithm [7, 22]. Our representation is based also on a regular decomposition of the span-space, whereby we use a different subdivision scheme and a different data structure for isosurface cell information. Additionally, once the data is stored in the span-triangle, we do not require any explicit searching for active cells. Our data structure establishes a tight coupling between representation and point-based rendering, leading to both fast updates of the surface model and interactive rendering rates.

3 Methods

The main elements of our visualization pipeline are illustrated in figure 2. In the first processing step we access the input volume in disk memory. We generate the span-triangle data structure in system memory that contains all isosurface cell information for a user-specified scalar range. Then, we extract the point data for one specific isovalue and transfer it to graphics memory for rendering. During exploration we update the surface model incrementally on each change of the isovalue. At the end of the visualization pipeline, the user has the possibility to fix the isovalue, releasing unused memory resources and making them available for further application steps or the generation of a higher-quality model.

In the following subsection 3.1, we describe the span-triangle data structure and the related isosurface extraction algorithm. Subsection 3.2 deals with the generation of the data structure from the input volume. Finally, in section 3.3 we address the rendering technique, based on our data structure, as well as the corresponding update scheme during exploration.

3.1 The Data Structure

The design of our data structure is based on a special scalar range representation that was designed for the acceleration of isosurface extraction [16]. Assuming a cell with a scalar minimum x_{min} and maximum x_{max}, we consider the cell as a point (x_{min}, x_{max}) in $\Re^2$. The corresponding two-dimensional space with cell-minima on the x-axis and cell-maxima on the y-axis is the so-called span-space. Unlike the cell interval representation $[x_{min}, x_{max}]$, it is possible to subdivide the data domain in a simpler way and to develop an efficient searching algorithm for active cells [22]. We focus on medical

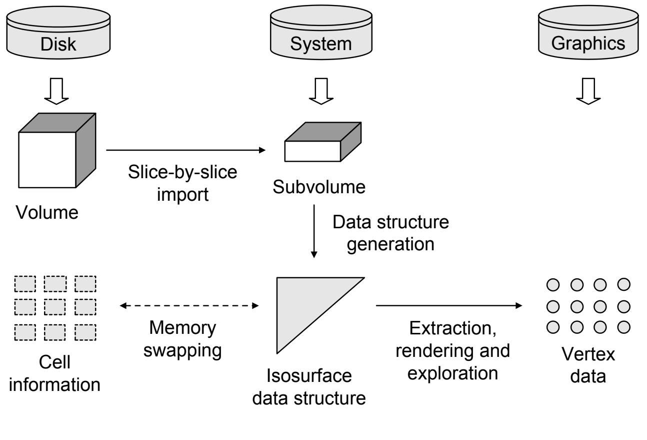

Figure 2: Our visualization pipeline, supporting three levels of memory: disk, system and graphics.

volume data sets (e.g. CT, MRI) that typically contain only discrete scalar values. Consequently, we consider a discretized version of the span-space, illustrated in figure 3.

The span-triangle contains information of potentially active cells for a pre-defined range of isovalues $[\upsilon_{min}, \upsilon_{max}] \subseteq [r_{min}, r_{max}]$, which we call *exploration range*. Our goal was to create a data structure, which allows the efficient extraction of an isosurface model for a specific isovalue as a set of active cells. We have chosen linear arrays as the major building blocks in order to achieve optimal cache utilization. Therefore, there are three types of data containers: *base array*, *span array* and *cell information array*.

- Each element in the base array corresponds to cells with a particular *based* cell minimum value $b = x_{min} - \upsilon_{min}$ and contains a span and cell information array.
- Each span array is implemented as an offset table, containing indices to the appropriate cell information. Furthermore, an item in the offset table corresponds to cells with a certain cell span $s = x_{max} - x_{min}$.
- During generation of the span-triangle, all potentially active cells are collected and stored in the appropriate cell information array, sorted by the cell span s. Therefore, we realize a cell sorting in the span-space by b and s.

The cell information contains two items: The *cell identifier* is a four bytes integer value, equal to the index of the first corner sample of each cell in the input volume. The *cell normal* is represented by a normal vector, belonging to the center of each cell. We store each normal in spherical coordinates quantized to two byte values, i.e. the azimuth and polar angles. This results in a compact representation, requiring only six bytes of memory per cell. Figure 4 illustrates the structure of the span-triangle. The layout of the base and span arrays forms a triangle in the span-space.

The specific structure of the span-triangle enables the possibility to access active cells in a very efficient way. The isosurface extraction reduces to a set of linear traversal operations, working in the cell information arrays. Let *Base*, *Span* and *CellInfo* represent the data arrays of the span-triangle. Furthermore, assume that Base[b].Span[s] returns the offset value to the cell information for a based cell minimum of b and cell span of s. Let Base[b].CellInfo.Max be the maximum index for the CellInfo array, corresponding to the total number of cells for b. The following pseudo-code shows the extraction routine for an isovalue υ:

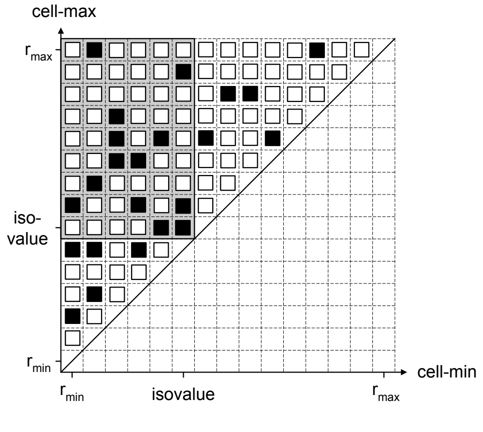

Figure 3: Discretized span-space. The small rectangles represent cell data. The grey region marks the cells that are active for the current isovalue.

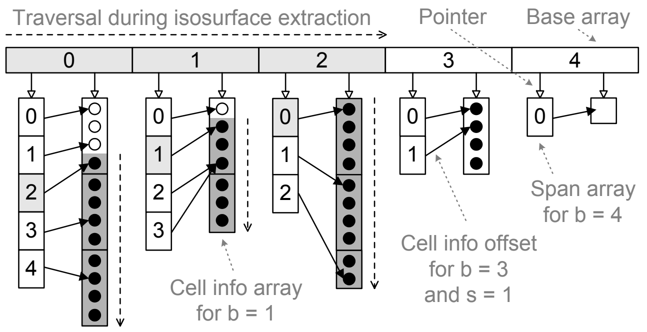

Figure 4: Sample span-triangle data structure for an exploration range of $[\upsilon_{min}, \upsilon_{max}] = [0, 4]$ and isovalue $\upsilon = 2$. Active cells are marked as filled circles, inactive cells as empty circles. Only array elements that are colored grey need to be accessed.

```
FOR $b$ FROM 0 TO $\upsilon - \upsilon_{min}$ DO
  $s := \upsilon - \upsilon_{min} - b$;
  FOR i FROM Base[$b$].Span[$s$] TO Base[$b$].CellInfo.Max DO
    Add cell from Base[$b$].CellInfo[i] to isosurface;
  END
END
```

The extraction of the isosurface is a nonhierarchical procedure, where only those cells that contribute to the current isosurface are accessed. Moreover, it is just a traversal of the base array, combined with look-ups to the cell information via the offsets from the span array, as shown in figure 4. Therefore, the extraction runs in optimal time complexity $O(k)$, for k as the number of active cells. The inner loop of the algorithm can be replaced by a memory block operation in order to read out the cell information with highest efficiency.

3.2 Generation of the Data Structure

Before we describe the span-triangle generation, we list possible strategies for a pre-selection of the exploration range $[\upsilon_{min}, \upsilon_{max}] \subseteq [r_{min}, r_{max}]$. The background for the pre-selection is that only specific scalar subranges contain useful information. Figure 5 illustrates a typical situation, when working with volumetric data sets: Choosing a too low or too high isovalue results in an inadequate visualization. By utilizing point primitives we have the potential to reduce memory consumption of the span-triangle and speed-up its generation. Therefore, it makes sense to choose an exploration range that is as small as possible, discarding regions that are not

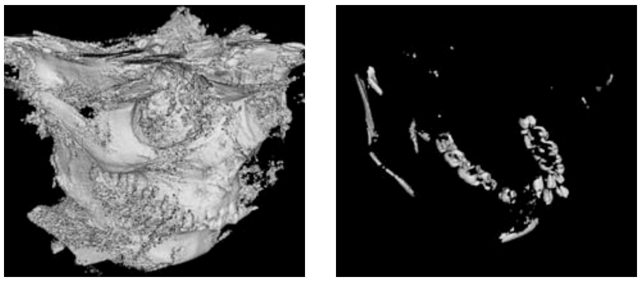

Figure 5: The left image shows a too low, the right image a too high isovalue. Information content of both visualizations is rather low.

useful for the current application. In our implementation we employ manual trial-and-error to find a good interval. This is in fact a simple and robust solution, but not useful for real applications, especially in the medical context. Therefore, we propose three different strategies for the pre-selection:

- When working with medical computer tomography data sets, we can pre-select by using Hounsfield units [10]. Specific scalar ranges can be mapped to anatomical structures of the human body, e.g. skin, soft tissue and bone.
- Another possibility is to exploit data-driven approaches from semi-automatic transfer function design for direct volume rendering. For example, Kindlmann et al. use a method to locate boundaries in the data value space to support the user during selection of the transfer function [13]. The output of such an algorithm could be used for our span-triangle generation.
- Finally, a simple but effective solution is to choose the exploration range, based on a maximally allowed memory consumption. First, we compute a "reduced" span-triangle, storing only the counters of potentially active cells, instead of the cell information itself. Then, we set the largest possible exploration range, shrinking it until memory consumption is low enough.

The span-triangle is created in a two-pass approach. In the first step we generate a *temporary data structure*. Afterwards, we build the final span-triangle. This approach is required to achieve an efficient balance between the speed and memory overhead during the generation: First, we use a fast radix-sort-based approach with $O(n)$ time complexity for sorting the cell information by b and s [21]. This requires the availability of a *cell histogram* in advance. Second, having the number and type of cells, we do not need any extra allocation of the cell information arrays and we have no performance overhead due to additional copy operations during the sorting process.

The temporary data structure is a single linked list of buckets. The bucketing is required to achieve robust allocation even on systems that are prone to memory fragmentation. Moreover, it allows us to minimize the memory overhead in the second pass. Each bucket contains the same information per cell as the final span-triangle and additionally the b and s indices. The temporary data structure is built by a cell-by-cell traversal of the input volume, similar to the Marching Cubes approach:

1. Identification of potential isosurface cells, based on the predefined exploration range: A cell is considered as potentially active, if there is an overlap between the cell interval and exploration interval, i.e. $[x_{min}, x_{max}] \cap [\upsilon_{min}, \upsilon_{max}] \neq \emptyset$. If this condition fails, the cell is skipped.
2. Computation of the cell information, i.e. cell index and cell normal: The normal vector is calculated with the central difference operator [14]. Then, it is normalized and converted into spherical coordinates, as mentioned in subsection 3.1.
3. Computation of offsets b and s for the span arrays: The offsets are computed from the cell histogram. This approach is equivalent to the calculation of the counter and offset tables in the radix-sort algorithm. Note that for cases where $[\upsilon_{min}, \upsilon_{max}] \subset [r_{min}, r_{max}]$, we have to crop b and s to the exploration range, before we can use them for the indexing of the span-triangle.

Remark that each iteration needs to access only two slices of the input volume simultaneously. Therefore, it is not required to load the complete volume into system memory. No further access to the data in disk memory is necessary.

In the second pass of the generation routine, we walk through the temporary data structure and sort the cell information into the span-triangle. Thereby, we follow again the radix-sort approach and use the available offset tables. There is one special case that we have to consider during this procedure: There may be specific cell types that do not occur in the input volume. Then, we need to adapt the corresponding offset value in the span-array to point to the same cell information as the next *valid* cell type element. Such a situation is illustrated in the cell info array for $b = 1$ in figure 4. As soon as we have finished the traversal of a temporary data bucket, we release it to reduce the memory overhead. Following such an strategy we need to keep only one data set at the same time in system memory. Note that the temporary cell information has a memory overhead, compared to the final span-triangle. However, the employed bucketing for the temporary data structure allows us to swap buckets to disk memory, before the maximally allowed memory consumption is reached.

3.3 Isosurface Visualization and Update

To visualize the isosurface model for one specific isovalue we combine the extraction process from the span-triangle data structure with point-based rendering. The main steps of this procedure are:

1. Traversal of the span-triangle and extraction of the relevant cell information.
2. Conversion of cell information to a graphics-hardware compatible format, called *vertex data*.
3. Transfer of vertex data to graphics memory and display.

For isosurface rendering, each vertex consists of a position and normal vector in the floating-point format. When working with high-resolution data sets we have to find a reasonable trade-off between speed and quality. One important focus in our work is interactivity, so we have decided to approximate each cell by one point, located at the cell center. This is a known technique from mesh decimation and level-of-detail rendering and is incorporated also in other surface rendering systems [2]. Points are rendered by projecting them onto the image plane with a surface splatting algorithm, using the normal vector for shading [6]. Remember that cell information is stored in a compact format, packed to one 32-bit cell index and two 8-bit spherical normal angles. We calculate the point positions and normals on-the-fly during isosurface extraction. Having the initial dimensions of the input volume, we obtain the point coordinates (in volume index space) using efficient integer division and modulo operations. Let the variable *index* be the volume array index from a cell information array. Assume constants *dim.x* and *dim.xy* as the initial volume dimensions in x-direction and for a slice, respectively:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \texttt{index DIV dim.xy} \\ \texttt{index MOD dim.xy DIV dim.x} \\ \texttt{index MOD dim.xy MOD dim.x} \end{pmatrix} \tag{1}$$

Moreover, we require sine and cosine functions for the calculation of the normal vector. These are precalculated in small look-up-tables with only 256 entries, because spherical angles are stored in eight bits.

The most important point for achieving interactive exploration is the *incremental update* of the isosurface model. On any change of the isovalue, the currently extracted model is updated without discarding cells that are still active. Due to the coherence of our data structure we transfer only the absolutely necessary information between system and graphics memory: First, we free parts of the vertex data, which correspond to cells that have became inactive after the isovalue change. Afterwards, we upload the data for all activated cells, as shown in figure 6. The exploration is realized using an incremental version of the presented isosurface extraction. The following pseudo-code shows the update algorithm for an increasing isovalue $\upsilon > \upsilon^+$. The update for a decreasing isovalue is analogous:

```
FOR $b$ FROM 0 TO $\upsilon - \upsilon_{min}$ DO
    $s := \upsilon - \upsilon_{min} - b$;
    $s^+ := \upsilon^+ - \upsilon_{min} - b$;
    Free vertex data from Base[$b$].Span[$s$] to Base[$b$].Span[$s^+$-1];
END

FOR $b$ FROM $\upsilon - \upsilon_{min}$ + 1 TO $\upsilon^+ - \upsilon_{min}$ DO
    $s^+ := \upsilon^+ - \upsilon_{min} - b$;
    Upload vertex data, starting from Base[$b$].Span[$s^+$];
END
```

One remaining challenge that has to be considered is that the vertex data upload can be quite slow for large changes of the isovalue. To handle such cases, we have incorporated a *progressive update scheme* to achieve smooth interaction without blocking the application: In advance, we estimate the transfer rate from system to graphics memory. During exploration, we interrupt the upload process whenever necessary to guarantee a user-defined *response time*. We continue with the upload of the remaining vertex data on the next frame. We have observed that choosing a response time of approximately 125 ms is enough to minimize the delay between the extraction of the model and feedback to the user.

4 Implementation

Our implementation is based on the *Julius* software development framework for medical visualization [12]. All steps of the presented visualization pipeline have been realized with reusable and extendible software components.

One challenge that we had to consider during the implementation of our data structure is the problem of allocating large continuous blocks of memory. Usually, the reason is memory fragmentation, occurring on PC machines with 32-bit addressing. The solution to this problem is a bucketing technique as we have utilized for the temporary data structure during the span-triangle generation. The cell information arrays are the major part in the span-triangle regarding memory consumption. Therefore, we divide them into buckets that can be allocated safely.

We exploit the following hardware-accelerated features for point rendering that are available in the OpenGL API: We use OpenGL point primitives `GL_POINTS` for display. This results in low-quality splats, compared to solutions that are based e.g. on Gaussian reconstruction kernel functions [19]. On the other hand, we can exploit full hardware-acceleration for transformation, lighting and rasterization. To solve the hole-filling problem in point rendering we use a similar approach as Bærentzen et al. [2]: The size of each point in screen space is calculated on the graphics processor unit, using the `GL_ARB_point_parameter` extension. Its screen size is attenuated, based on the distance to the viewer. Finally, to allocate vertex information directly in graphics memory we take advantage of vertex buffer objects (VBO), implemented via `GL_ARB_vertex_buffer_object`. Therefore, data does not need to be transferred from system memory on each frame, effecting in rendering speed-up.

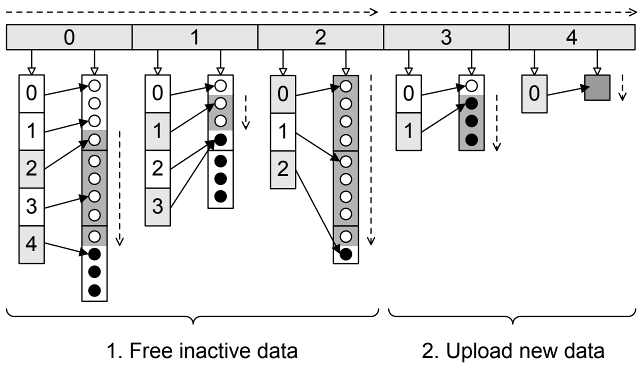

Figure 6: Incremental update of a sample span-triangle for an isovalue, changing from $\upsilon = 2$ to $\upsilon^+ = 4$. Empty circles represent point data that has to be freed from graphics memory or remains inactive. Filled circles indicate cells that have to be uploaded to graphics memory or are still active. Notice the grey background color in this illustration, showing elements of the span-triangle that have to be really accessed.

5 Results

To measure the performance and memory consumption during exploration, we have chosen a current desktop PC, equipped with an Intel Pentium IV 2.8 GHz processor and 2 GB of system memory. Our graphics card has been an ATI Radeon 9800 Pro with 256 MB of memory. Table 1 gives an overview of the data sets, used for the evaluation. We have chosen volumes mainly from medical application areas, covering sizes from 16 MB to 588 MB. The current implementation of our renderer allows only structured grids and isotropic data. Bonsai, Head and Leg have been aquired with a CT-device. The Skull1 and Skull2 data sets are cone-beam scans. Table 2 shows the span-triangle statistics, including pre-processing times for the generation of the data structure and sizes in system memory. Head and Leg reveal an increased memory consumption, showing the significance of image noise and width of the exploration range for the size of the span-triangle. We achieve an average reduction in system memory of 61 %, compared to disk memory in table 1. The reason for this result is a combination of three concepts:

- Adaption of the exploration range to current application needs, instead of using the whole scalar range.
- Utilization of point data, allowing to sample the volume only on the surface of the current model.
- Compact cell information, requiring only six bytes.

Data set	Resolution	Scalar bits	Disk memory
Bonsai	$256 \times 256 \times 256$	8 / 8	16 MB
Head	$320 \times 404 \times 230$	12 / 16	54 MB
Skull1	$512 \times 512 \times 512$	12 / 16	256 MB
Leg	$599 \times 599 \times 500$	12 / 16	342 MB
Skull2	$700 \times 700 \times 600$	12 / 16	588 MB

Table 1: Used volume data sets. The left scalar bit values show *used bits*, the right values are *allocated bits*. Disk memory is based on allocated bits.

The dominating part considering memory allocation are the *CellInfo* arrays. Therefore, we have not included the consumption of the raw span-triangle structure in table 2. The span-triangle contains two pointers in each base array element and one offset value per span array element. Furthermore, there are $\upsilon_{max} - \upsilon_{min} + 1$ base array elements. This effects in the following raw size with $\Delta e = \upsilon_{max} - \upsilon_{min}$:

$$size_{raw} = (\Delta e + 1)(\frac{\Delta e}{2} + 2) \qquad (2)$$

Each pointer and offset value requires four bytes in our implementation, which results in additional 0.1 MB to 28.1 MB of memory for our test data sets. Equation 2 shows that the memory consumption of the raw data structure (i.e. without cell information) is dependent quadratically on the width of the exploration range. The resulting restriction is tolerable for medical data sets, where scalar ranges are limited typically to 12 bits. Data sets with wide scalar intervals need to be reduced during pre-processing. The generation times from table 2 include both steps: creation of the temporary data structure and the span-triangle. The resulting pre-processing times are low enough for medical use of our visualization pipeline.

Table 3 shows the number of active cells, the time for the initial extraction of the point data from the span-triangle and the rendering speed for selected isosurface models. We have asked a medical expert to select specific isovalues by using our technique in order to reveal characteristic anatomical structures. It is hard to compare directly our extraction times to previous results, because they use other grid types, perform additional cell triangulations or do not focus on iterative updates [24, 2, 6, 4]. Nevertheless, our results show that extracting the point model from the span-triangle instead of a volume grid results in a significant increase in performance, even though the reported measurements include the transfer to graphics memory. Rendering times have been measured for random rotations and 300 frames. The average rendering rate is 35.8 mio. of points per second for Bonsai and Head, and 60.1 mio. points per second for the large data sets. These values are depending on the size of the input volume and consequently on the splat sizes in screen space. The rendering performance shows great potential even for larger data sets than presented in this paper. Note that the range of variation for the measured values in table 3 is quite large. One reason is the varying number of buckets due to our bucketing techniques that we have implemented for the cell information arrays, as mentioned in section 4. Another possible reason is the internal resource management on the GPU. In the current implementation, each vertex element consumes 24 bytes in graphics memory (due to six floating point values). There is no additional overhead per cell for the span-triangle in system memory. Note that as soon as we fix the isovalue and release non-used information we gain additional reductions of memory from 1 % to 26 %, as compared to the values in table 1.

Finally, we provide measurements concerning the exploration speed with our progressive update scheme. Table 4 shows the *exploration times* for each data set. We define it as the time, required for the update of the isosurface model when slicing from one extrema to the other within the exploration range in one single step. All measurements include also visual feedback. The values are dependent on the user-specified response time, here 125 ms. Typically, users explore in small isovalue step-sizes. Therefore, the exploration time gives a worst-case view for the delay concerning interactivity of an exploration system. Additionally, we have measured exploration times that are two orders of magnitude slower, when using an isovalue step-size of one. Combining this result with the full-extraction times in table 3 shows clearly the advantage of the progressive system and utilization of arbitrary large step-sizes. Finally, the results show that the performance is highly dependant on the direction of the exploration. The reason is that the vertex data upload to graphics memory requires more time than freeing of the data. The balance between both operations may vary, depending on the isovalue direction during update of the isosurface model.

Figures 7 to 10 on the last page of this paper show screenshots of selected data sets for characteristic isovalues. The Head data set is shown on the first page. We have observed that for large volumes the rendering of one point per cell results in adequate quality. In practice the projected cell in screen space is typically not much larger than the size of a pixel. Therefore, we have decided not to implement the support for multi-component isosurfaces or point-on-surface projection, as described in [6]. In other cases, e.g. where the quality of this approach is not sufficient or the input volume covers a too large scalar value range, our exploration system can be used for an accelerated *pre-selection* of the isovalue: Our fast pre-processing and extraction times make possible to create a lower-quality point model in order to determine the desired isovalue. Afterwards, we can use the corresponding cell information from the span-triangle for accelerated generation of a higher-quality model, using other point- or even triangle-based techniques.

Data set	Isovalue	Active cells	First extraction	FPS
Bonsai	40	563273	41 ms	91.6
	78	215523	17 ms	163.7
	193	122052	15 ms	247.1
Head	1077	541303	64 ms	87.5
	1730	452676	104 ms	85.8
	2814	77091	120 ms	148.6
Skull1	1698	2912153	220 ms	23.0
	2048	1470773	131 ms	42.0
Leg	2714	2703212	201 ms	24.5
	3053	4844748	828 ms	7.1
	3165	2851516	238 ms	22.9
Skull2	1900	3829972	286 ms	18.0
	2230	612395	66 ms	93.6

Table 3: Number of active cells, time for the initial extraction of point data including transfer to graphics memory, and rendering speed in frames-per-second (FPS). The number of active cells corresponds to the number of points during rendering.

Data set	$\upsilon_{min}, \upsilon_{max}$	Generation	System memory
Bonsai	30, 255	1.3 s	12.5 MB
Head	260, 4095	6.7 s	69.8 MB
Skull1	1450, 4095	13.9 s	82.4 MB
Leg	2500, 4095	19.6 s	173.2 MB
Skull2	1750, 4095	21.2 s	102.1 MB

Table 2: Generation times and memory sizes of the span-triangle for manually pre-selected exploration ranges. All timings include construction of the temporary and final data structure without volume import. The memory values are based only on pure cell information.

Data set	$\upsilon_{min} \rightarrow \upsilon_{max}$	$\upsilon_{max} \rightarrow \upsilon_{min}$
Bonsai	< 0.1 s	0.1 s
Head	0.2 s	0.2 s
Skull1	0.3 s	0.9 s
Leg	1.3 s	0.6 s
Skull2	0.2 s	2.6 s

Table 4: Exploration times, using progressive update. The left column contains timings for increasing isovalues, the right column for decreasing isovalues.

6 Conclusion and Future Work

We have presented a novel span-triangle data structure, based on the span-space representation. We have shown that it can be used for interactive isosurface exploration of high-resolution volumes. Isosurface models can be extracted without significant delay during interaction by utilizing an incremental and progressive update scheme. We assure real-time point-based rendering with typically more than 20 frames-per-second by using a tight coupling between data representation and display, as well as by taking advantage of hardware acceleration. We can process volumes with approximately 600 MB of memory with a visualization pipeline that works on all levels of memory. Pre-processing times for high-resolution volumes are about 20 seconds and are therefore suitable for actual medical applications. Furthermore, our approach has the following benefits:

- Each cell information is stored only once in the isosurfacing data structure.
- We achieve high cache efficiency due to continuous low-level data structures.
- The utilization of data coherence makes possible to minimize the transfer between system and graphics memory.
- Isovalues can be changed with arbitrary large step-sizes within the exploration range.
- Isosurface models are represented compactly, because geometric primitives are placed only within active cells.
- The isosurface extraction algorithm has linear time complexity with respect to the output size and is therefore optimal.

Currently, we are extending our point-based renderer to support anisotropic input volumes via non-circular splats. Additionally, we are implementing out-of-core processing for the span-triangle data structure. The idea is to swap parts of the cell data to disk memory, when system memory consumption gets too high. Moreover, we plan to utilize more advanced features of graphics hardware and shader programming to improve the rendering quality. Examples are *point sprites* and *vertex programs*. Another interesting topic is the utilization of our data structure in other applications than isosurface exploration, e.g. animation and surface growth techniques.

Acknowledgements

The Head and Leg data sets are provided by H.-F. Zeilhofer from the Department of Maxillofacial and Reconstructive Surgery, Kantonspital Basel, Switzerland. The Bonsai tree is courtesy of S. Roettger, VIS, University of Stuttgart, Germany, available from http://www.volvis.org.

References

[1] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva. Point set surfaces. *IEEE Visualization 2001*, pages 21–28, October 2001.

[2] J. A. Bærentzen and N. J. Christensen. Hardware accelerated point rendering of isosurfaces. *Journal of WSCG*, 11(1):41–48, February 2003.

[3] Chandrajit L. Bajaj, Valerio Pascucci, and Daniel Schikore. Fast isocontouring for improved interactivity. In *VVS*, page 39 pp., 1996.

[4] Jatin Chhugani, Sudhir Vishwanath, Jonathan Cohen, and Subodh Kuma. ISOSLIDER: A system for interactive exploration of isosurfaces. In *VisSym 2003 Joint Eurographics-IEEE TCVG Symposium on Visualization*, pages 259–266, May 2003.

[5] P. Cignoni, P. Marino, C. Montani, E. Puppo, and R. Scopigno. Speeding up isosurface extraction using interval trees. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):158–170, April 1997.

[6] Christopher S. Co, Bernd Hamann, and Kenneth I. Joy. Iso-splatting: A point-based alternative to isosurface visualization. In *11th Pacific Conference on Computer Graphics and Applications (PG'03)*, page 325 pp., October 2003.

[7] R. S. Gallagher. Span filtering: An efficient scheme for volume visualization of large finite element models. In G. M. Nielson and L. Rosenblum, editors, *Proceedings of IEEE Visualization '91*, pages 68–75, October 1991.

[8] M. Giles and R. Haimes. Advanced interactive visualization for CFD. *Computing Systems in Education*, 1(1):51–62, 1990.

[9] J. P. Grossman and William J. Dally. Point sample rendering. In *Eurographics Rendering Workshop'98*, pages 181–192, June 1998.

[10] G. N. Hounsfield. Computerized transverse axial scanning (tomography). 1. description of system. *The British Journal of Radiology BJR*, 46(552):1016–1022, December 1973.

[11] Takayuki Itoh and Koji Koyamada. Automatic isosurface propagation using an extrema graph and sorted boundary cell lists. *IEEE Transactions on Visualization and Computer Graphics*, 1(4):319–327, December 1995.

[12] Erwin Keeve, Thomas Jansen, Bartosz von Rymon-Lipinski, Zbigniew Burgielski, Nils Hanssen, Lutz Ritter, and Marc Lievin. An open software framework for medical applications. In *International Symposium on Surgery Simulation and Soft Tissue Modeling IS4TM*, pages 302–310, June 2003.

[13] Gordon Kindlmann and James W. Durkin. Semi-automatic generation of transfer functions for direct volume rendering. In *IEEE Symposium on Volume Visualization*, pages 79–86, 1998.

[14] Marc Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3):29–37, May 1988.

[15] Marc Levoy and Turner Whitted. The use of points as a display primitive. Technical Report TR 85-022, University of North Carolina at Chapel Hill, 1985.

[16] Y. Livnat, H.-W. Shen, and C. R. Johnson. A near optimal isosurface extraction algorithm using the span space. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):73–84, 1996.

[17] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proc. SIGGRAPH'87*, pages 163–169, 1987.

[18] Michael Meißner, Jian Huang, Dirk Bartz, Klaus Mueller, and Roger Crawfis. A practical evaluation of popular volume rendering algorithms. In *Proceedings of the 2000 IEEE symposium on Volume visualization*, pages 81–90. ACM Press, 2000.

[19] Hanspeter Pfister, Matthias Zwicker, Jeroen van Baar, and Markus Gross. Surfels: Surface elements as rendering primitives. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 335–342. ACM Press, 2000.

[20] Szymon Rusinkiewicz and Marc Levoy. QSplat: A multiresolution point rendering system for large meshes. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 343–352. ACM Press, 2000.

[21] Robert Sedgewick. *Algorithms in C++*. Series in Computer Science. Addison-Wesley, 3rd edition, 1998.

[22] Han-Wei Shen, Charles D. Hansen, Yarden Livnat, and Christopher R. Johnson. Isosurfacing in span space with utmost efficiency (ISSUE). In Roni Yagel and Gregory M. Nielson, editors, *IEEE Visualization '96*, pages 287–294, 1996.

[23] Han-Wei Shen and Christopher R. Johnson. Sweeping simplices: A fast iso-surface extraction algorithm for unstructured grids. In *IEEE Visualization*, pages 143–150, 1995.

[24] Phil Sutton, Chuck Hansen, H.-W. Shen, and Dan Schikore. A case study of isosurface extraction algorithm performance. In *2nd Joint Eurographics-IEEE TCCG Symposium on Visualization*, May 2000.

[25] Jane Wilhelms and Allen van Gelder. Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201–227, 1992.

[26] Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, and Markus Gross. Surface splatting. In Eugene Fiume, editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 371–378. ACM Press, 2001.

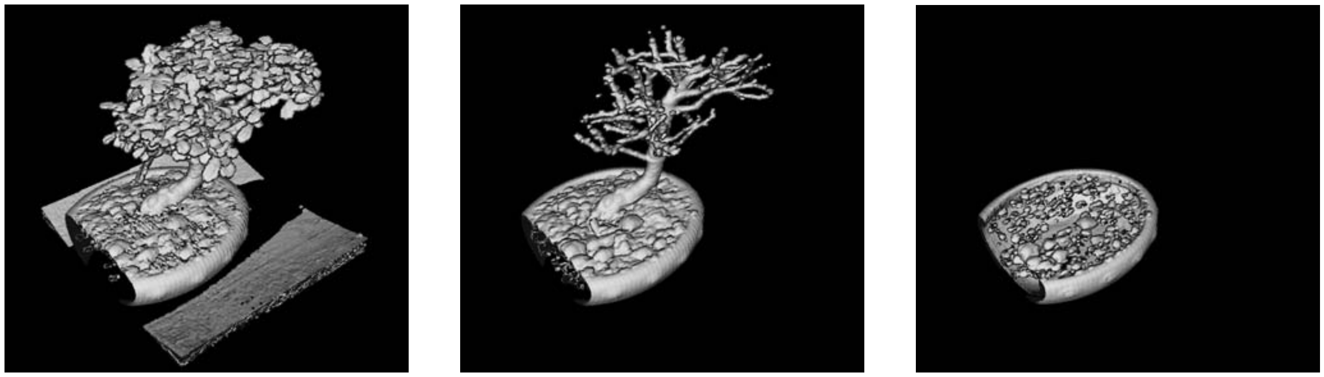

Figure 7: Isosurface exploration of the bonsai tree data set for isovalues $\upsilon = 40$, 78 and 193.

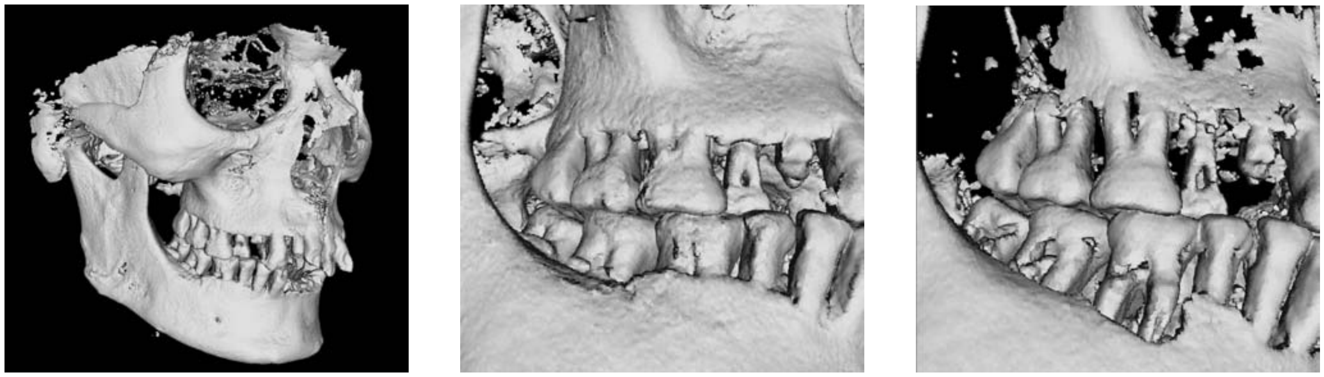

Figure 8: Human dry skull volume for isovalue $\upsilon = 1698$ (left and center image) and $\upsilon = 2048$ (right image).

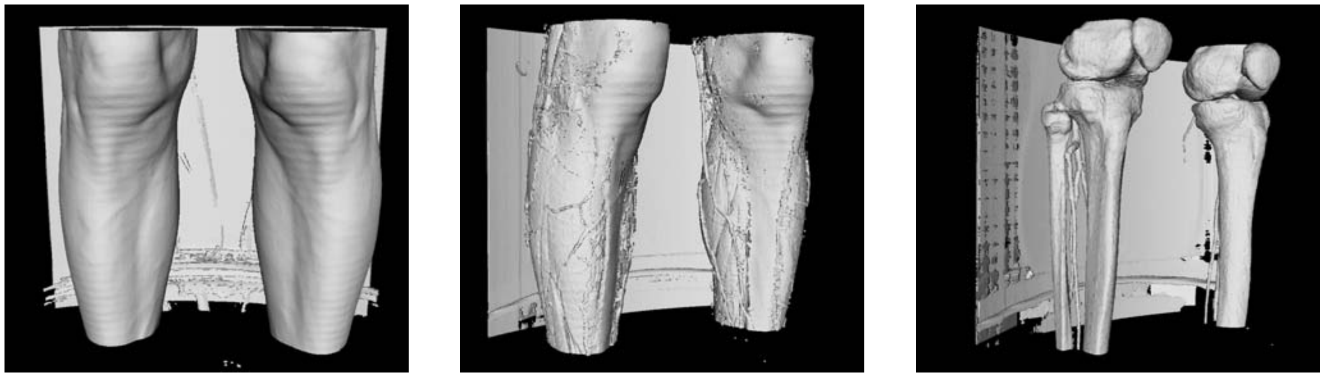

Figure 9: Leg data set, showing the skin ($\upsilon = 2714$), subcutaneous vessels ($\upsilon = 3053$) and bone ($\upsilon = 3165$).

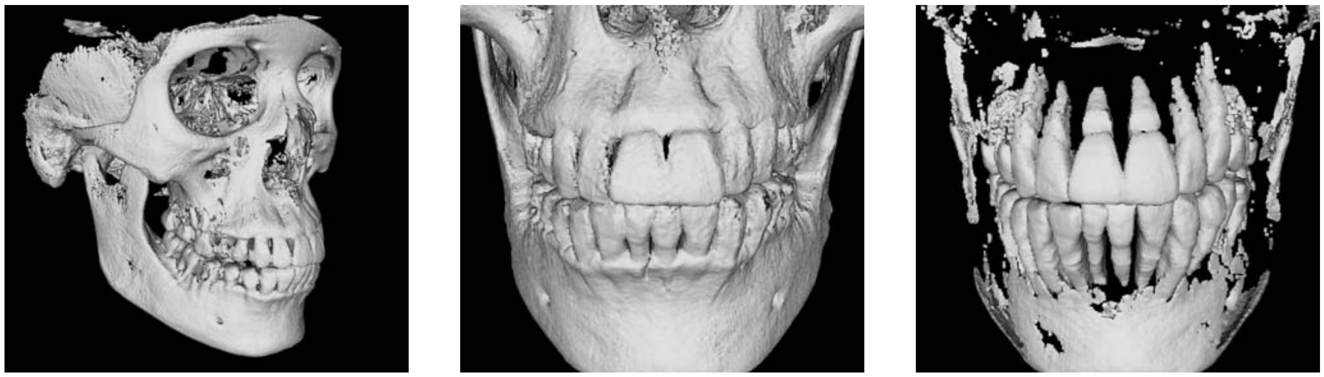

Figure 10: Large model for isovalues $\upsilon = 1900$ (left, center) and $\upsilon = 2230$ (right). The images show the skull bone, teeth and dental roots.