J Vis
DOI 10.1007/s12650-011-0119-5

REGULAR PAPER

Susumu Nakata · Shohei Aoyama · Ryosuke Makino · Kyoko Hasegawa · Satoshi Tanaka

Real-time isosurface rendering of smooth fields

Received: 29 July 2011 / Accepted: 30 October 2011

Abstract This paper presents a new technique for real-time isosurface visualization of three-dimensional smooth fields. This technique enables direct isosurface rendering of smoothly defined fields without generating intermediate polygon models as used in the marching-cube algorithm, a standard technique for isosurface extraction. We developed a parallel algorithm that is suitable for effective computation on graphics hardware by converting any field to a set of polynomials, enabling the detection of intersections between rays and isosurfaces at arbitrary isovalue with small computational cost. In addition, the isovalue to be rendered can be changed in real time, because the algorithm is not restricted to a specific isovalue. The technique can also be applied to the problem of the direct rendering of implicit surfaces that are defined as isosurfaces of smoothly defined fields.

Keywords Volume visualization · Isosurface rendering · Implicit surfaces · GPU

1 Introduction

We present a scheme for isosurface rendering of three-dimensional fields, such as interpolated functions of scattered data or field functions of implicit surfaces. Volume data can be defined on a set of organized or unorganized scattered points depending on the source, for e.g., results of simulations using mesh-free methods such as smoothed particle hydrodynamics (Liu and Liu 2003), the element-free Galerkin method (Belytschko et al. 1994), the meshless local Petrov–Galerkin method (Atluri and Zhu 1998), and the radial point interpolation method (Wang and Liu 2002).

For visualization of such unorganized volume datasets, fields need to be interpolated to determine field values at arbitrary points in a domain. Many methods can be used to perform the interpolation of scattered data. One simple method for interpolation is to generate tetrahedral mesh using, for example, the Delaunay tetrahedrization algorithm (Amidror 2002). In this method, the field is interpolated in each tetrahedron using the values defined at the four vertices, and the reconstructed field is continuous, but not smooth, at the faces between neighboring tetrahedra. Another approach, the scattered-data interpolation technique based on radial basis functions (RBFs) (Jang et al. 2004; Neophytou and Mueller 2005), generates continuous fields by solving a global system of equation. The generated fields are not only continuous but also smooth in the

S. Nakata (✉) · S. Tanaka
College of Information Science and Engineering, Ritsumeikan University, Kyoto, Japan
E-mail: snakata@is.ritsumei.ac.jp

S. Aoyama · R. Makino
Graduate School of Science and Engineering, Ritsumeikan University, Kyoto, Japan

K. Hasegawa
Kinugasa Research Organization, Ritsumeikan University, Kyoto, Japan

Published online: 06 December 2011

sense that they are differentiable at any point in the domain. The moving least square (MLS) approximation technique is also useful for smooth field generation from unorganized point sets (Ledergerber et al. 2008). The field is evaluated by solving the local least square problem, and the cost for the initial process required in RBFs can be reduced. The volumetric version of multi-level partition of unity (Tsukamoto et al. 2011), which we call VMPU, is another approach for field generation from an unorganized volume. The field function is defined as a weighted sum of locally fitted fields, the degree of freedom of which is selected adaptively according to the local complexity of the field. The idea of field reconstruction is also used for representing three-dimensional surface models. The field can be reconstructed as an approximation of signed distance using methods, such as, RBF (Carr et al. 2001), MLS (Shen et al. 2004), or the multi-level partition of unity (MPU) method (Ohtake et al. 2003).

Our aim is to perform isosurface rendering at an arbitrary isovalue in real time, without loss of continuity and the smoothness of a surface. There are several approaches for rendering isosurfaces of scattered volume data or smooth fields, for e.g., polygonal isosurface extraction (Stander and Hart 1997; Navratil et al. 2007; Gelas et al. 2009), point-based surface extraction (Oka et al. 2007; Rosenthal and Linsen 2006; Linsen et al. 2008), or direct isosurface rendering using ray-casting (Levoy 1990; Knoll et al. 2009). The ray-casting-based approach enables accurate rendering of smooth isosurfaces, if intersections between the rays that originate from a camera position and isosurfaces are obtained. A simple method for computing the intersections is the ray-marching method (Perlin and Hoffert 1989), which requires function values only for intersection detection. Sphere tracing (Hart 1996) and LG surfaces (Kalra and Barr 1989) are other methods for ray intersection, which guarantee the detection of the intersections and enables accurate ray-casting rendering in principle.

A standard approach for accelerating the isosurface-rendering process is to make use of graphics hardware (GPUs), and a number of rendering algorithms suitable for GPUs have been developed for volume data (Jang et al. 2004; Engel et al. 2006). A GPU-accelerated rendering method of implicit surfaces was proposed by Hadwiger et al. (2005). In this method, the ray-casting of implicit surfaces is accelerated by restricting the ray-traversal range. Another method proposed by Liu et al. (2009) performs fast isosurface rendering by finding the cells through which the isosurface exists. Reiner et al. (2011) proposed a GPU-based implicit surface rendering technique in which the field is expressed as a signed distance function. Note that the latter three algorithms accelerate the GPU-based rendering using the structure of the isosurface at a specific isovalue.

In this paper, we present a direct isosurface-rendering scheme performed in real time by converting smooth fields into grids of locally defined polynomials and by implementing a parallelized rendering algorithm on a GPU. In our method, fields are converted into grids of locally defined polynomials using B-spline interpolation. We adopt uniform quadratic B-splines for the conversion, which results in piecewise polynomials of degree six. Note that other uniform B-splines such as linear or cubic B-spline can be applied in essentially the same manner depending on the level of smoothness of the field. The degree of the B-spline plays a role in the trade-off between the smoothness of the isosurface and the computational cost in time and memory. The ray/isosurface intersections are computed by solving algebraic equations defined along rays that originate from an eye point and the isosurface and solvers for non-linear or algebraic equations. In this paper, we adopted the traditional ray-marching method combined with the bisection method for the detection of ray/isosurface intersections. Note that the sphere tracing is also appropriate, and in addition, solvers specific to algebraic equations such as Bairstow's method or the Durand–Kerner method (McNamee 2007) can be applied for solving six-degree polynomial equations along the rays. The cost for direct rendering primarily depends only on the number of pixels and the position of the screen. It is independent of the complexity of the original field, because the cost for polynomial evaluation is constant everywhere in the domain, i.e., the computational time does not increase even if we divide the domain into numbers of small cells for accurate approximation. This property contributes to effective parallelization on a device based on single-instruction multiple-data (SIMD) architecture such as GPUs.

2 Field approximation using piecewise polynomials

Consider a field $f(\boldsymbol{x})$ defined in a rectangular domain. Here, we assume without loss of generality that the domain is a unit cube $\boldsymbol{x} = (x, y, z) \in [0, 1[^3$. The function $f(\boldsymbol{x})$ is assumed to be an interpolation of the scattered volume data obtained by methods such as RBF, MPU, or VMPU, as described in Sect. 1, from a set of scalar data $f_1,\ldots, f_n$ given at scattered points $\boldsymbol{x}_1,\ldots, \boldsymbol{x}_n$. Figure 1 (left) shows an example of scattered volume.

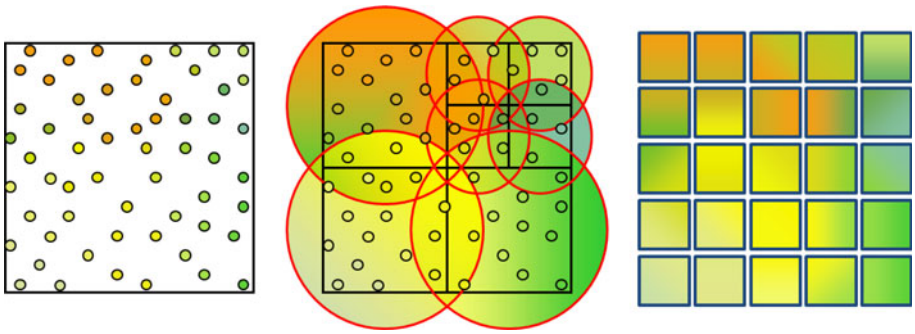

Fig. 1 Field interpolation of scattered volume using VMPU and conversion to grid polynomial. A given scattered volume (*left*) is interpolated using VMPU (*center*) and are approximated as a set of polynomials (*right*)

In the case of simple RBF interpolation, the function $f(\boldsymbol{x})$ is expressed as a linear combination of RBFs that are globally or compactly supported. Note that in the case of globally supported RBFs, the cost for field evaluation is proportional to the number of bases determined, depending on the number of given scattered points or the complexity of the field. If compactly supported RBFs are used, the cost depends on the number of RBFs of which supports contain the evaluation point.

The field reconstructed by MLS is defined using the solution of a least square problem that is determined using the scattered volume in the vicinity of each evaluation point. In standard MLS, the evaluation requires processing neighborhood query and solving a least square problem, the cost for which depends on the number of neighboring points.

In VMPU, the reconstructed field is expressed as a weighted sum of local functions, each of which is defined over a subspace given as a decomposition of the domain, i.e.,

$$f(\boldsymbol{x}) = \sum_{i=1}^{l} \phi_i(\boldsymbol{x}) Q_i(\boldsymbol{x}),$$

where l is the number of subspaces, $\varphi_i(\boldsymbol{x})$ is the locally supported weight functions, and $Q_i(\boldsymbol{x})$ is the local fields approximating the scattered volume in the vicinity of each subspace. Figure 1 (center) shows an example of the structure of VMPU, where each circle is the support of the weight function in which a local field is determined to locally approximate the given scattered volume data. In this case, the cost for evaluation mainly depends on the number of local fields the support of which contains the evaluation point. In addition, the process for searching the local functions is required for each evaluation point. Note that the cost for evaluation varies depending on the location of the evaluation point.

In this manner, the complexity of the reconstructed fields typically depends on the degree of freedom of the reconstruction and, in most cases, the complexity changes place by place depending on the local variation of the fields. To construct fields which satisfy the condition that the cost required for field evaluation must be constant and independent of the position of the evaluation point, we convert the field to a set of polynomials defined in the cells as a uniform grid of the domain, $[0,1[^3$. Let $g_{ijk}(\boldsymbol{x})$ be the polynomial in the (i, j, k)-th cell, where i, j, and k are the indices of the cells in the x-, y- and z-directions, respectively. The functions $g_{ijk}(\boldsymbol{x})$ require the following conditions:

- The polynomial in each cell connects with neighboring polynomials continuously and smoothly.
- The computational complexity of the field evaluation is constant in $[0, 1[^3$.

Such a field can be obtained using a uniform quadratic B-spline interpolating the grid-sampled values of $f(\boldsymbol{x})$. Assume that the field values $f(\boldsymbol{x}_{ijk})$ at N^3 uniform grid points are obtained, where $\boldsymbol{x}_{ijk} = ((i + 0.5)/N, (j + 0.5)/N, (k + 0.5)/N)$. An approximation of the original field $f(\boldsymbol{x})$ can be obtained as a linear combination of B-spline bases as

$$f(\boldsymbol{x}) \approx g(\boldsymbol{x}) = \sum_{i=-1}^{N} \sum_{j=-1}^{N} \sum_{k=-1}^{N} c_{ijk} \varphi_i(x) \varphi_j(y) \varphi_k(z), \quad \boldsymbol{x} = (x, y, z) \in [0, 1[^3, \tag{1}$$

where $\varphi_i(t)$ is the uniform quadratic B-spline function defined as

$$\varphi_i(t) = \phi(t-i),$$
$$\varphi(t) = \begin{cases} (t+1)^2/2 & (-1 \le t < 0) \\ -t^2 + t + 1/2 & (0 \le t < 1) \\ (t-2)^2/2 & (1 \le t < 2) \\ 0 & (\text{otherwise}) \end{cases} \tag{2}$$

and c_{ijk} are the coefficients determined to satisfy the interpolation condition, $f(\boldsymbol{x}_{ijk}) = g(\boldsymbol{x}_{ijk})$. The local polynomial of each cell can be retrieved from (1) and (2) as

$$g_{ijk}(\boldsymbol{x}) = \sum_{r=0}^{2}\sum_{q=0}^{2}\sum_{r=0}^{2} \alpha_{ijk}^{(p,q,r)} x^p y^q z^r, \tag{3}$$

where $\alpha_{ijk}^{(p,q,r)}$ are the coefficients of the monomials.

The parameter N plays the role of the trade-off between the accuracy of the field approximation and the memory cost, where the total memory cost for the coefficients is $27N^3$. Figure 2 shows the relation between the original and the approximated field with different fineness parameters. While a high memory cost is required for accurate approximation, the smooth field in the whole domain is stored and isosurface rendering at an arbitrary isovalue is supported. The evaluation of each local polynomial $g_{ijk}(\boldsymbol{x})$ requires 26-floating-point multiplications. Note that the cost in time is independent of the fineness of the grid in principle.

3 Direct isosurface rendering on GPU

In this section, we present a parallel algorithm for direct isosurface rendering that is suitable for computation on GPU. As described in Sect. 1, many direct rendering algorithms are available. Direct rendering is performed based on the idea of ray-casting, i.e., each pixel value of the screen is determined using the intersection between a ray and an isosurface. In our algorithm, the standard ray-marching algorithm is used to find the initial intervals for the ray/isosurface intersection detection and the intersection is obtained by the bisection method.

The aim is to visualize the isosurface $g(\boldsymbol{x}) = \alpha$ for user-specified positions of a camera and a screen. The process for determining the pixel value of one pixel of the screen consists of five steps: detect the range for ray-marching, detect the initial interval for the bisection method, find the intersection, obtain the normal at the intersection and determine the pixel value. Let t be the distance from the pixel along the ray. For the first step, we use two intersections between the ray and a bounding sphere containing the domain as the range of ray-marching. Let t_S and t_T be the starting and ending points of the range along the ray. Ray-marching is performed at sampling points starting from t_S, with a constant interval determined, so that the number of sampling points on a ray is within a user-specified parameter $M_{\max}$. If an interval such that the solution of $g(\boldsymbol{x}(t)) = \alpha$ is found before the sampling point reaches the termination point t_T the process proceeds to the next step and finds the solution of $g(\boldsymbol{x}(t)) = \alpha$ by the bisection method. We fix the iteration number for effective parallelization, as we describe later. Then, the normal vector of the isosurface at the intersection is computed using the gradient of (3), and the pixel value is determined by the normal.

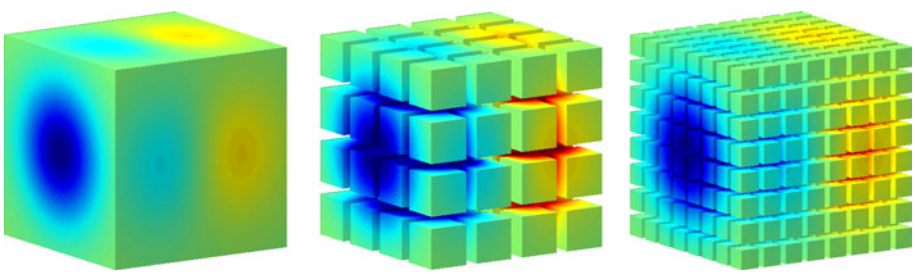

Fig. 2 Example of conversion from a scalar field to a grid of polynomials. The original field (*left*) is converted to a grid of polynomial for user-specific parameters of fineness, $N = 4$ (*center*) and $N = 8$ (*right*)

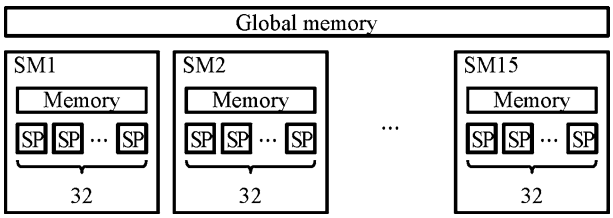

Fig. 3 Architecture of GPU (NVIDIA GeForce GTX 480). This device consists of 480 SPs in total. The rendering process for one pixel of the screen is assigned to one of the SPs

In our algorithm, the rendering process is performed on GPU for fast computation. For the purpose, all the coefficients of the local polynomial need to be transferred to the GPU memory in advance, i.e., $27N^3$ floating points are stored in the GPU memory. In addition, each frame of real-time rendering requires the camera and screen position and the isovalue as user inputs. For GPU computation, the process needs to be divided into small processes hierarchically according to the structure of the processors on the GPU, i.e., the unit consists of plural streaming multiprocessors (SM), each of which has a set of streaming processors (SP). Figure 2 shows a typical structure of processors and memory on GPU. In our method, the screen is divided into pixel blocks, each of which is assigned to one SM, and one pixel of the block is assigned to one SP in the SM. Note that SPs in SM runs in a SIMD manner. This concept should be considered for effective parallel computation. The optimal size of the block depends on the device; we determine the appropriate size empirically. We used a NVIDIA GeForce GTX 480, a GPU possessing a total of 480 SPs that enables the determination of 480 pixel values in parallel in principle (Fig. 3). While the cost for ray-marching varies by the position of the pixel, because of the stopping rule that depends on the position of the ray/isosurface intersection, the difference in cost is expected to be small in many cases. The other steps described earlier require a constant computational cost, which contributes to effective parallel computation on the SIMD architecture.

4 Results

In this section, we present the results of rendering obtained by our parallel algorithm performed on the GPU. We used a NVIDIA GeForce GTX 480 for the rendering process, and we fixed the number of iterations for the bisection method to five. The screen size is fixed to 512 × 512 throughout this experiment. We tested three different examples, "hydrogen atom" (volume data of 512^3 regular grid obtained as a sampling of electron's probability), "head aneurysm" (volume data of 512^3 regular grid, courtesy Michael Meißner, Viatronix Inc.), and "Venus" (point cloud on a surface, 207,697 points, our original data obtained by a 3D range scanner). We constructed the interpolating field $f(\boldsymbol{x})$ for the hydrogen atom using the VMPU method. For the second data, we constructed the field using the standard B-spline interpolation technique with the volume data of 16^3, 32^3, 64^3 and 128^3 regular grids obtained as shrunk volume of the original one. For Venus, we used the MPU method to construct a field defining the implicit surface and converted the field to $g(\boldsymbol{x})$ using the grid sizes $N = 16$, 32, 64, and 128. We set the maximum number of the ray-marching steps along each ray as $M_{\mathrm{MAX}} = 100$ and 200.

The performance of direct rendering on GPU is summarized in Table 1. The relative error caused by the approximation of the polynomial is measured at dense grid points of the domain. The results show that

Table 1 Relative error caused by approximation with grid polynomial and frame rates required for rendering

		$N = 16$	$N = 32$	$N = 64$	$N = 128$
Hydrogen atom	Relative error (Max.)	1.3×10^{-1}	6.4×10^{-2}	4.4×10^{-2}	2.0×10^{-2}
	Relative error (Avg.)	5.4×10^{-4}	1.3×10^{-4}	4.5×10^{-5}	9.3×10^{-6}
	fps ($M_{\mathrm{MAX}} = 100$)	345	248	146	72
	fps ($M_{\mathrm{MAX}} = 200$)	187	135	79	38
Head aneurysm	fps ($M_{\mathrm{MAX}} = 100$)	228	176	117	60
	fps ($M_{\mathrm{MAX}} = 200$)	119	93	62	31
Venus surface	Relative error (Max.)	2.0×10^{-1}	6.7×10^{-2}	5.3×10^{-2}	3.1×10^{-2}
	Relative error (Avg.)	3.3×10^{-5}	7.7×10^{-6}	1.9×10^{-6}	6.2×10^{-7}
	fps ($M_{\mathrm{MAX}} = 100$)	459	369	238	112
	fps ($M_{\mathrm{MAX}} = 200$)	255	202	133	63

accuracy increases with grid size, as expected. The rendering results are shown in Figs. 4, 5, and 6. The rendering quality changes with two parameters: grid size and sampling density for ray-marching. The frame rates (fps: frames per second) shown in Table 1 perform as expected with respect to the number of ray-marching steps, i.e., the parameter M_{MAX} determines the trade-off between quality and precision. In addition, the fps of the Venus model is larger than that of the others, because the number of rays intersecting with the isosurface is larger. Another factor determining fps according to the results is the sampling density N, although the parameter is expected to not affect computational time, because the computational complexity does not change with the change of N, as discussed in the previous section. One possibility of this phenomenon is the cost for memory access on GPU.

Our rendering technique can be applied to other basic visualization techniques such as slices or multiple isosurfaces and the different visualizations can be performed in the same framework of ray-marching

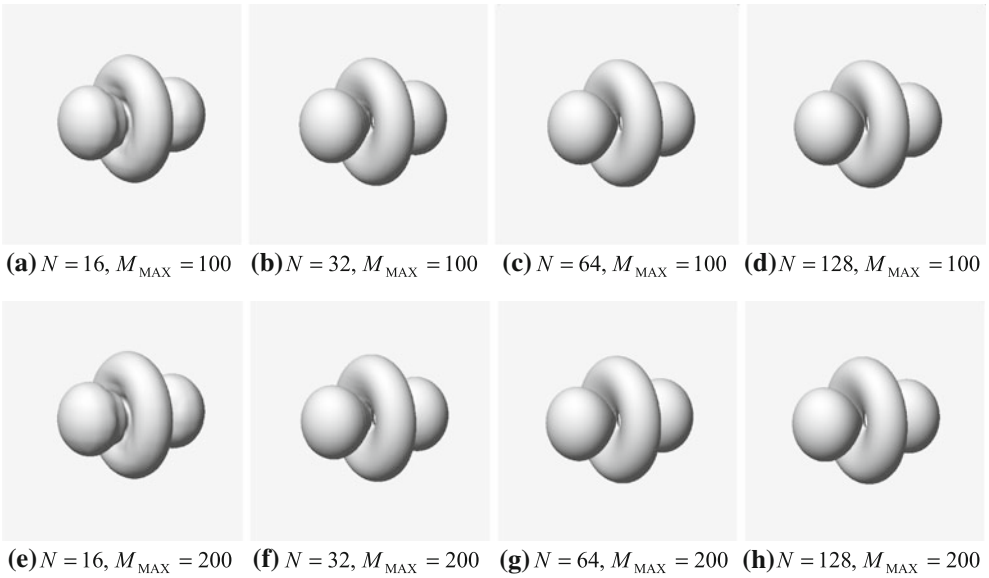

Fig. 4 Isosurface rendering of "hydrogen atom"

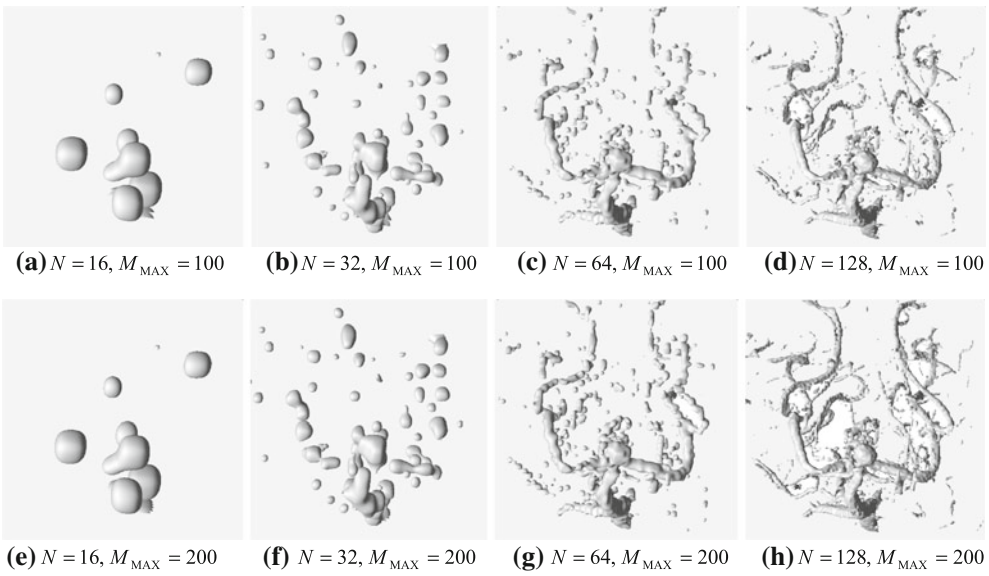

Fig. 5 Isosurface rendering of "head aneurysm"

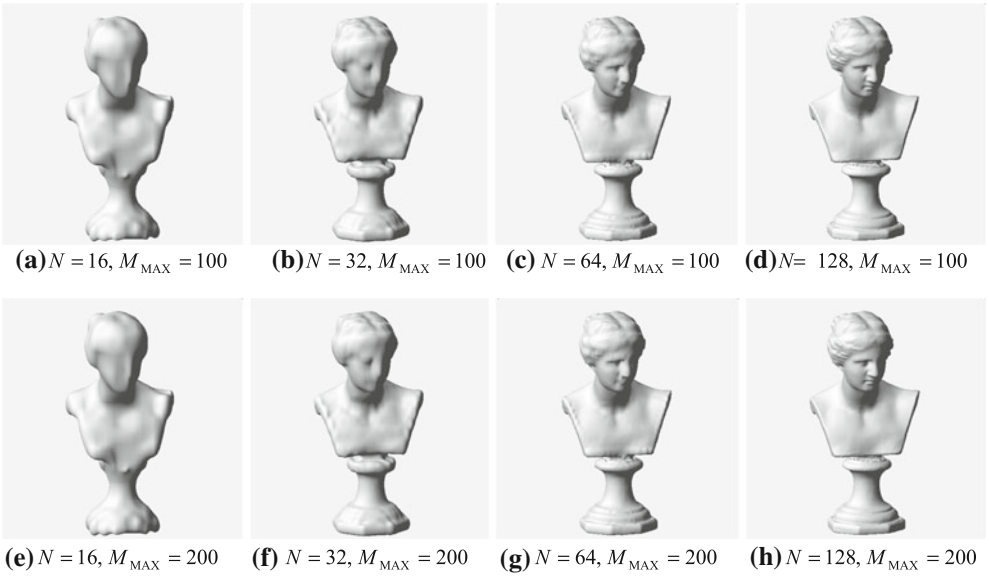

Fig. 6 Isosurface rendering of 'Venus'

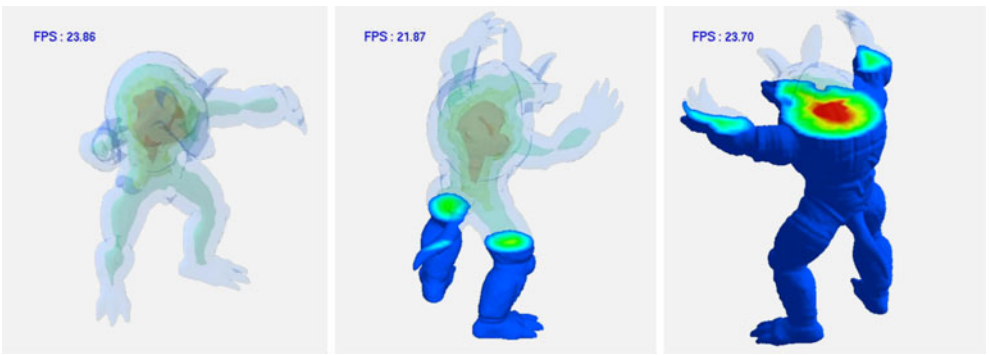

Fig. 7 Combination of opaque surface visualization, transparent multiple isosurfaces visualization and a slice inside the surface

method. Figure 7 shows an example of combination of an opaque isosurface, transparent multiple isosurfaces and a slice of the field obtained using the MPU method for a set of points called 'Armadillo' (courtesy Stanford 3D Scanning Repository). The rendering process for each pixel is performed in one traversal along the corresponding ray. In our rendering test, the frame rate was over 20 in this case, where $N = 128$ and $M_{\mathrm{MAX}} = 100$.

5 Discussion and limitations

The results in the previous section show the efficiency of the isosurface-rendering method based on the combination of two techniques: field expression using piecewise polynomials and parallel computation on the GPU. Given the size of the screen, the speed and the quality of the isosurface rendering are determined by two user-specified parameters: the fineness of the ray-marching steps and the fineness of the grid for piecewise polynomials. The number of the ray-marching steps M_{MAX} affects both the speed and the quality. Thus, the ray-marching step should be selected such that they are as small as possible under an acceptable rendering quality. The grid size N determines the accuracy of the field approximation. Note that in principle, the parameter N does not have any effect on the rendering speed as described in Sect. 2, although the rendering tests show that the rendering time increases with respect to N. A reasonable explanation of this inconsistency can be the memory access overhead caused by high memory consumption.

Although the proposed method works well, as demonstrated by the rendering tests, it has a weakness with regard to memory consumption as described above, e.g., 226-MB memory space is required to store all the polynomial coefficients in case of $N = 128$. In other words, fast rendering is achieved in exchange for an increase in memory consumption. Another limitation is that the approximation only covers continuous and differentiable distribution, although the functions constructed by the VMPU method can be either discontinuous or non-differentiable depending on the input scattered data. Resolving these drawbacks by incorporating some new framework to our method can be a future research of this study.

One application of the proposed method is the rendering of implicit surfaces. The surfaces can be visualized without generating intermediate polygon models and the smoothness of the surface is guaranteed by the property of the quadratic B-spline. Another application is the volume visualization of the functions obtained as a result of scattered-data interpolation. Our method enables not only isosurface rendering but also the visualization of slices, as illustrated in Fig. 7, using the same framework of the proposed method. In addition, users can interactively change the isovalue and the position of the slices in real time because the computational complexity is not influenced by the change in the parameters.

6 Conclusion

In this paper, we have presented an algorithm for the real-time isosurface rendering of fields without loss of smoothness of a surface. In our approach, the field is converted to a grid of locally defined polynomials, and the isosurface-rendering process is performed on GPU in parallel. The computational complexity of the field evaluation is constant, and this property contributes to effective parallelization on the SIMD architecture. For detecting ray/isosurface intersections, not only the traditional isosurface-rendering methods such as ray-marching or sphere-tracing but also any standard solvers for algebraic equation can be applied. In addition, the isovalue can be changed in real time, because the rendering algorithm is not restricted to a specific isovalue. The algorithm is also suitable for real-time rendering of implicit surfaces.

References

Amidror I (2002) Scattered data interpolation methods for electronic imaging systems: a survey. J Electron Imaging 11(2):157–176

Atluri SN, Zhu T (1998) A new meshless local Petrov–Galerkin (MLPG) approach in computational mechanics. Comput Mech 22(2):117–127

Belytschko T, Lu YY, Gu L (1994) Element-free Galerkin methods. Int J Numer Methods in Eng 37(2):229–256

Carr JC, Beatson RK, Cherrie JB, Mitchell TJ, Fright WR, McCallum BC, Evans TR (2001) Reconstruction and representation of 3D objects with radial basis functions. In: Proceedings of SIGGRAPH 2001, pp 67–76

Engel K, Hadwiger M, Kniss J, Rezk-Salama C, Weiskopf D (2006) Real-time volume graphics. A K Peters, Natick

Gelas A, Valette S, Prost R, Nowinski W (2009) Variational implicit surface meshing. Comput Graph 33(3):312–320

Hadwiger M, Sigg C, Scharsach H, Buhler K, Gross M (2005) Real-time ray-casting and advanced shading of discrete isosurfaces. Comp Graph Forum 24(3):303–312

Hart J (1996) Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces. Vis Comp 12(10):527–545

Jang Y, Weiler M, Hopf M, Huang J, Ebert DS, Gaither KP, Ertl T (2004) Interactively visualizing procedurally encoded scalar fields. In: Joint Eurographics—IEEE TCVG Symposium on Visualization, pp 35–44

Kalra D, Barr A (1989) Guaranteed ray intersection with implicit surfaces. In: Proceedings of SIGGRAPH'89, pp 297–306

Knoll A, Wald I, Hansen C (2009) Coherent multiresolution isosurface ray tracing. Vis Comp 25(3):209–225

Ledergerber C, Guennebaud G, Meyer M, Bacher M, Pfister H (2008) Volume MLS ray casting. IEEE Trans Vis Comp Graph 14(6):1539–1546

Levoy M (1990) Efficient ray tracing of volume data. ACM Trans Graph 9(3):99–106

Linsen L, Van Long T, Rosenthal P, Rosswog S (2008) Surface extraction from multi-field particle volume data using multi-dimensional cluster visualization. IEEE Trans Vis Comp Graph 14(6):1483–1490

Liu GR, Liu MB (2003) Smoothed particle hydrodynamics: a meshfree particle method. World Scientific, Singapore

Liu B, Clapworthy GJ, Dong F (2009) Fast isosurface rendering on a GPU by cell rasterization. Comp Graph Forum 28(8):2151–2164

McNamee JM (2007) Numerical methods for roots of polynomials—part I. Elsevier, Amsterdam

Navratil PA, Johnson JL, Bromm V (2007) Visualization of cosmological particle-based datasets. IEEE Trans Vis Comp Graph 13(6):1712–1718

Neophytou N, Mueller K (2005) GPU accelerated image aligned splatting. In: Gröller E (ed) Volume Graphics 2005. Eurographics, New York, pp 197–205

Ohtake Y, Belyaev A, Alexa M, Turk G, Seidel H-P (2003) Multi-level partition of unity implicits. In: Proceedings of SIGGRAPH 2003, pp 463–470

Oka M, Nakata S, Tanaka S (2007) Preprocessing for accelerating convergence of repulsive-particle systems for sampling implicit surfaces. In: Proceedings of IEEE International Conference on Shape Modeling and Applications, pp 232–240
Perlin K, Hoffert EM (1989) Hypertexture. In: Proceedings of SIGGRAPH'89, pp 253–262
Reiner T, Mückl G, Dachsbacher C (2011) Interactive modeling of implicit surfaces using a direct visualization approach with signed distance functions. Comp Graph 35(3):596–603
Rosenthal P, Linsen L (2006) Direct isosurface extraction from scattered volume data. In: Joint Eurographics—IEEE TCVG Symposium on Visualization, pp 99–106
Shen C, O'Brien JF, Shewchuk JR (2004) Interpolating and approximating implicit surfaces from polygon soup. In: Proceedings of SIGGRAPH 2004, pp 896–904
Stander BT, Hart JC (1997) Guaranteeing the topology of an implicit surface polygonization for interactive modeling. In: Proceedings of SIGGRAPH'97, pp 60–69
Tsukamoto Y, Kawashima S, Inoue S, Ito I, Kataoka S, Kojima K, Hasegawa K, Nakata S, Tanaka S (2011) Data fitting independent of grid structure using a volumic version of MPU. J Vis 14(2):161–170
Wang JG, Liu GR (2002) A point interpolation meshless method based on radial basis function. Int J Numer Methods Eng 54(11):1623–1648