Volume 57 Number 1 January 2014

ISSN 1674-733X CN 11-5847/TP

中国科学： 信息科学

SCIENCE CHINA Information Sciences

Sponsored by

CHINESE ACADEMY OF SCIENCES

NATIONAL NATURAL SCIENCE FOUNDATION OF CHINA

info.scichina.com
www.springer.com/scp
link.springer.com

SCIENCE CHINA PRESS

Springer

Go To Website

SCIENCE CHINA
Information Sciences

Contents

Vol. 57 No. 1 January 2014

SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

January 2014, Vol. 57 012106:1–012106:15
doi: 10.1007/s11432-012-4722-4

Real-time rendering of algebraic B-spline surfaces via Bézier point insertion

WEI FeiFei & FENG JieQing*

State Key Laboratory of CAD & CG, Zhejiang University, Hangzhou 310058*, China*

Received July 14, 2012; accepted October 17, 2012; published online December 28, 2012

Abstract This paper presents a GPU-based real-time raycasting algorithm for piecewise algebraic surfaces in terms of tensor product B-splines. 3DDDA and depth peeling algorithms are employed to traverse the piecewise surface patches along each ray. The intersection between the ray and the patch is reduced to the root-finding problem of the univariate Bernstein polynomial. The polynomial is obtained via Chebyshev sampling points interpolation. An iterative and unconditionally convergent algorithm called Bézier point insertion is proposed to find the roots of the univariate polynomials. The Bézier point insertion is robust and suitable for the SIMD architecture of GPU. Experimental results show that the proposed root-finding algorithm performs better than other root-finding algorithms, such as Bézier clipping and B-spline knot insertion. Our rendering algorithm can display thousands of piecewise algebraic patches of degrees 6–9 in real time and can achieve the semi-transparent rendering interactively.

Keywords real-time rendering, algebraic B-spline surface, root finding, depth peeling

Citation Wei F F, Feng J Q. Real-time rendering of algebraic B-spline surfaces via Bézier point insertion. Sci China Inf Sci, 2014, 57: 012106(15), doi: 10.1007/s11432-012-4722-4

1 Introduction

Because of its advantages on intersection, blending, offset, arbitrary topology, etc., the algebraic surface is an important alternative to the parametric surface. The algebraic surfaces in terms of tensor product B-spline, abbreviated as ABS in this paper, introduced by Patrikalakis and Kriezis [1] can model globally smooth shapes of complex topologies. The coefficients of the ABS surface have intuitive geometry means and can be used to adjust the surface shape, while those of the general algebraic surfaces have no such property. Compared to the NURBS surfaces [2] and the piecewise algebraic surfaces defined in the Bézier tetrahedrons, i.e. A-patches [3], the ABS surface is capable of modeling the globally smooth shapes with arbitrary topologies without explicitly specifying smoothness constraints along the patch boundaries. Compared to the subdivision surfaces, the ABS surface has analytical expression. Thus the related geometric computations and shape interrogations can be performed straightforwardly. Patrikalakis and Kriezis [1] also proposed two modeling methods for the ABS surfaces. The first one is to fit the lower dimensional geometries by using least-squares approximation, and the second one is to manipulate the existing ABS surfaces. Jüttler et al. [4], and Tong et al. [5] proposed different least-squares methods

*Corresponding author (email: jqfeng@cad.zju.edu.cn) to fit given scattered data. However, the topics of modeling free-form shapes with the ABS surfaces go beyond the scope of this paper.

Although the ABS surfaces have many potential advantages on shape modeling and shape interrogation, their real-time and high quality display is still an open problem. This paper will address the problem of real-time ray casting and semi-transparent rendering of the ABS surfaces. To exploit the SIMD architecture of GPU, 3DDDA and depth peeling algorithms are adopted to traverse the piecewise patches of ABS surface efficiently. An iterative and efficient root finding algorithm is proposed, namely Bézier point insertion, which performs better than the prevalent GPU-based root finding algorithms, such as Bézier clipping [6], B-spline knot insertion [7], etc. Our approach fully leverages the tremendous computing powers of contemporary GPU. Experimental results shows that the proposed method can high-quality render the ABS surfaces of degrees 6–9 containing thousands of patches in real time.

2 Related work

Besides contouring method proposed by Patrikalakis et al. [1], there are also other methods to display the ABS surfaces. Witkin et al. [8] adopted particles to sample and control the shape of implicit surfaces, which include the ABS surface. The polygonization is the most prevalent visualization method for the algebraic surfaces [9,10]. It fits the polygon rendering pipeline very well. However, these methods suffer from sampling artifacts.

Ray casting or ray tracing can reveal the topological and geometric details of the algebraic surface in pixel accuracy. Thus it is free of sampling artifacts under the condition of raster display resolution. However, it contains huge amount of computational costs in general, especially in the computations of the rays and surface intersections. Analytical solutions only exist for algebraic surfaces of degree 4 or lower. Based on this observation, Loop et al. [11] proposed a real-time rendering algorithm for A-patch surfaces up to degree 4 defined on the tetrahedron domains. However, it cannot guarantee correct α-blending results for complex shape defined by a large number of piecewise A-patches. Furthermore, it is prohibitively difficult to be extended to other domain cases, e.g. the rectangular domain of the ABS surface.

Many numerical root finding methods are adopted to compute the ray and algebraic surface intersections. Hanrahan [12] proposed a ray tracing algebraic surface method by using the Descartes, rules of signs for root isolation and the Newton's method for root refinement. Kalra et al. [13] adopted Lipschitz constant of the function and gradient to guarantee correct ray and implicit surface intersection. Interval arithmetic can estimate a conservative convex hull of function over a given domain. By using this property, Mitchell [14] isolated the roots by using repeated bisections till the interval contains a single root.

All the methods above focused on the root-finding problem of univariate polynomials in terms of power basis. Farouki et al. [15] proved that the polynomials in Bernstein form offer better numerical condition and accuracy than their power basis counterparts. The subdivision method proposed by Lane et al. [16] is the pioneering work that can solve the univariate Bernstein polynomials robustly. Bézier clipping proposed by Nishita et al. [17] is based on the convex hull property of Bézier curve and surface, and it is applied to ray tracing trimmed parametric surfaces. Spencer further explored the intuitive geometric properties of the Bernstein polynomial and proposed more efficient root finding algorithms [18]. Other variants and extensions of Bézier clipping algorithm [19–21] exploited different levels of convex hulls of the Bézier curves.

Recently, with the rapid development of performance and emergence of flexible programming languages, GPU becomes an attractive platform for scientific computing. Many GPU-based approaches are proposed to address the interactive ray tracing of algebraic surfaces. Pabst et al. [22] and Kanamori et al. [6] proposed iterative versions of Bézier clipping with fixed size stacks on GPU. Knoll et al. [23] proposed an iterative interval bisection root finding algorithm by using the SSE instruction-level optimization and coherently traversal methods. They extended their work to modern graphics hardware GPU by simulating the stack traversals [24]. Both the Bézier clipping and the interval arithmetic algorithms suffer from their

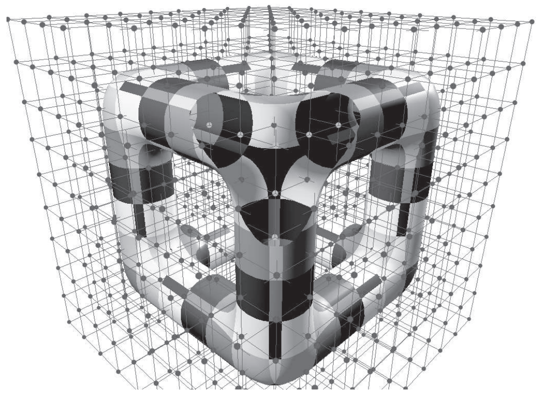


Figure 1 An ABS surface and its "control coefficients" on their abscissa positions. Light grey points inside the shape and dark grey points outside the shape indicate negative and positive coefficients respectively.

branching and irregular memory access and lead to poor efficiency on GPU. Reimers and Seland [7] transformed an implicit surface to its frustum form via the polynomial interpolation, and adopted the iterative B-spline knot insertion algorithm [25] for root finding. The algorithm is tailored for the SIMD architecture of GPU. However, they only process single algebraic surface, rather than piecewise ones. Recently, Wei and Feng propose a fast raycasting algorithm for the piecewise ABS surfaces [26], which combines the marching cube and the Newton-Raphson method. However its robustness greatly suffers from the ill-conditioned initial values.

3 Preliminary and algorithm overview

3.1 Algebraic B-spline surface

Before describing the proposed algorithm in detail, we first introduce the definition of algebraic B-spline surface. Let $\boldsymbol{X} = [x_0, x_1, \ldots, x_{M+m+1}]$, $\boldsymbol{Y} = [y_0, y_1, \ldots, y_{N+n+1}]$ and $\boldsymbol{Z} = [z_0, z_1, \ldots, z_{Q+q+1}]$ be three knot vectors in x, y and z directions respectively. An algebraic tensor product B-spline surface is defined as

$$F_{\mathrm{ABS}}(x,y,z) = \sum_{i=0}^{M}\sum_{j=0}^{N}\sum_{k=0}^{Q} w_{ijk} N_i^m(x) N_j^n(y) N_k^q(z) = 0, \tag{1}$$

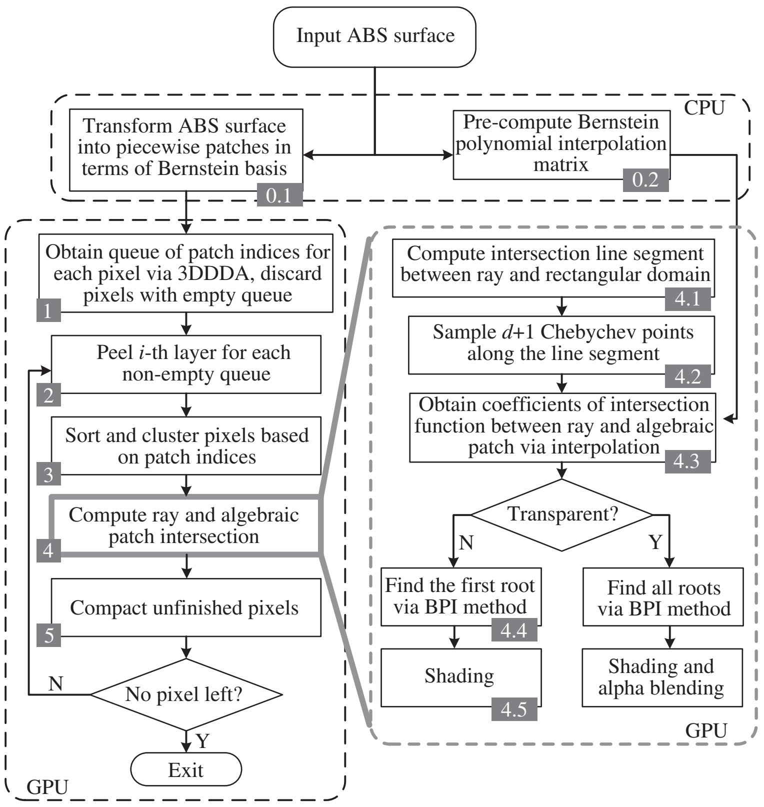
where $N_i^m(x)$, $N_j^n(y)$ and $N_k^q(z)$ are the B-spline basis functions of degrees m, n and q, determined by the knot vectors $\boldsymbol{X}$, $\boldsymbol{Y}$ and $\boldsymbol{Z}$ respectively. The degree of the ABS surface is $d = m + n + q$. The scalars $\{w_{ijk}\}$ in Eq. (1), namely control coefficients, have the similar function to the control points of parametric B-spline surface. An ABS surface example is illustrated in Figure 1.

In this paper, the degrees of ABS surfaces considered are no greater than $3 \times 3 \times 3$, which is adequate to model complex shapes of arbitrary topologies. The proposed rendering algorithm is implemented on GPU with SIMD architecture. In general, a GPU has much more processor cores (ALUs) than a CPU. However, each GPU core runs significantly slower than a CPU core and is not designed for solving complex problems. It is suitable for simple, data or computation intensive operations. By using the knot insertion algorithm, the ABS surface defined on $[x_i, x_{i+1}] \times [y_j, y_{j+1}] \times [z_k, z_{k+1}]$ $(m \leqslant i \leqslant M, n \leqslant j \leqslant N, q \leqslant k \leqslant Q,)$ can be converted into an algebraic patch in terms of tensor product Bernstein polynomials as follows:

$$F(x,y,z) = \sum_{i=0}^{m}\sum_{j=0}^{n}\sum_{k=0}^{q} F_{ijk} B_i^m(u(x)) B_j^n(v(y)) B_k^q(w(z)) = 0, \tag{2}$$

where

$$u(x) = \frac{x - x_i}{x_{i+1} - x_i},\ v(y) = \frac{y - y_j}{y_{j+1} - y_j},\ w(z) = \frac{z - z_k}{z_{k+1} - z_k},$$



Input ABS surface
CPU
Transform ABS surface into piecewise patches in terms of Bernstein basis
0.1
Pre-compute Bernstein polynomial interpolation matrix
0.2
Obtain queue of patch indices for each pixel via 3DDDA, discard pixels with empty queue
1
Peel i-th layer for each non-empty queue
2
Sort and cluster pixels based on patch indices
3
Compute ray and algebraic patch intersection
4
Compact unfinished pixels
5
No pixel left?
N
Y
Exit
GPU
Compute intersection line segment between ray and rectangular domain
4.1
Sample d+1 Chebychev points along the line segment
4.2
Obtain coefficients of intersection function between ray and algebraic patch via interpolation
4.3
Transparent?
N
Y
Find the first root via BPI method
4.4
Find all roots via BPI method
Shading
4.5
Shading and alpha blending
GPU

Figure 2 Flowchart of the proposed algorithm.

and $B_i^m(u)$, $B_j^n(v)$ and $B_k^q(w)$ are the Bernstein basis functions of degree m, n and q respectively. Thus, we can compute the intersection between the ray and the ABS surface patch by patch. And the coefficients of $F(x, y, z)$ can be load into on-die memory and shared by neighbor threads, which is very efficient on GPU. When the control coefficients of $F(x, y, z)$ have the same sign, i.e., all positive or negative, the algebraic patch defined by Eq. (2) will be null according to the convex hull property of Bernstein polynomials. This observation will help us to reject null algebraic patches quickly.

3.2 Algorithm overview

The flowchart of the proposed algorithm is shown in Figure 2. In the algorithm, the input ABS surface is first converted into piecewise algebraic patches in terms of Bernstein polynomials, whose domains form a 3D rectangular grid. For each ray, the 3DDDA algorithm is employed to traverse the rectangular domains and obtain a queue of patch indices. Then the depth peeling algorithm is adopted to process the queues in parallel. By using the functional composition via polynomial interpolation in Subsection 4.2, the ray and the patch intersection is reduced to a root finding problem of an univariate Bernstein polynomial. An iterative and unconditionally convergent root finding algorithm, namely Bézier point insertion (abbreviated as BPI in this paper), is proposed in Subsection 4.3. Finally the shading computation is performed after the smallest root is found. To render the semi-transparent effect, which is helpful to reveal the complex topology of the ABS surface, the queue of patch indices should be traversed in a back-to-front order so that all the zeros of the univariate polynomial are evaluated.

4 Real-time raycasting ABS surfaces via Bézier point insertion

4.1 Ray traversal

Unlike its parametric counterpart, the domains of the piecewise algebraic patches in Eq. (2) have the

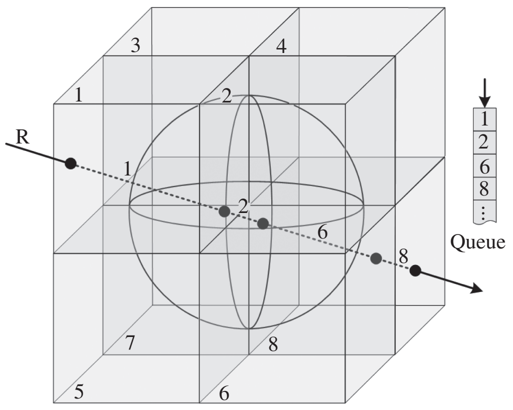


Figure 3 A ray traverses the rectangular domains of the algebraic patches. A queue of patch indices is obtained for the ray.

regular spatial structure, i.e., a 3D rectangular grids. As a preprocessing step of root finding, it is naturally to check whether the ray intersects the rectangular domains or not. The task can be accomplished efficiently via the 3DDDA algorithm [27], which is illustrated in Figure 3. By clipping the domains against the near and far clipping planes and skipping the domains that contain no surface via sign check of the control coefficients, a queue of patch indices is obtained, with which the ray may intersect potentially. In general, the queue will be traversed in the front-to-back order to compute the first hit point. To render the semi-transparent surface, the queue should be traversed in the back-to-front order to compute all of the intersections.

Loop and Blinn computed the intersections analytically between a ray and an A-patch of degree [11]. They projected the tetrahedral domain onto the screen space to determine the extent of the A-patch, then scan convert the projection of the domain, finally compute the first hit point for each pixel. For overlapped A-patches, they employ z-buffer approach to find the first hit point. Their algorithm is not applicable to the algebraic patch in terms of tensor product Bernstein polynomials since the projection of the rectangular domain on the screen space is much more complex. Therefore we adopt the depth peeling algorithm to traverse the queues in parallel [28,29]. In general, the depth peeling is terminated if the first hit point is found or there are no patch index left in the queue. In the case of rendering semi-transparent surface, the whole queue should be traversed in the back-to-front order.

4.2 Function composition

There are several methods to compute ray and surface intersections. They are two planar curves intersection method [30], Bernstein pyramidal polynomials method [31], ray marching and sampling method [23,24,32], functional composition method [7,11,33], etc. The functional composition method converts the ray and the algebraic surface intersection into a root finding problem of univariate polynomial. It is universal and the resulting univariate polynomial can be solved by many numerical algorithms. The functional composition between the ray and the metaball has analytical expression due to its simplicity and low degree [33]. In the methods [7,11], the algebraic patch defined in the world coordinate system is first transformed into the view coordinate system. Thus the functional composition between the ray and the algebraic patch can be evaluated directly by substituting x and y coordinate values into the transformed algebraic function. However it is not applicable to the ABS surface since the ABS surface may contain thousands of piecewise algebraic patches in terms of Bernstein polynomials.

In our setting, the functional composition is performed in the world coordinate system directly. According to the theory of Bernstein polynomials [34], the functional composition between the ray and the ABS surface function $F_{\text{ABS}}(x, y, z)$ in Eq. (1) is an univariate piecewise polynomial in terms of B-splines, which contains large numbers of knots and control coefficients. It is impractical to solve the B-spline polynomial in GPU since GPU has limited registers and local memory resources in general. Alternatively, we perform the functional composition between the ray and the algebraic patch in terms of Bernstein polynomials $F(x, y, z)$ in Eq. (2) as follows:

$$\begin{cases} F(x,y,z)=0, \\ R(t)=(x(t),y(t),z(t))=(1-t)R_{\text{in}}+tR_{\text{out}}, \end{cases} \tag{3}$$

where $R(t)$ is the parametric equation of the view ray, R_{in} and R_{out} are two intersection points between the ray $R(t)$ and the rectangular domain of the algebraic patch, and $t \in [0, 1]$. By substituting each component of $R(t)$ into $F(x, y, z)$, an univariate Bernstein polynomial of degree $d = m + n + q$ can be obtained as follows:

$$f(t)=F(x(t),y(t),z(t))=\sum_{i=0}^{d} f_i B_i^d(t)=0. \tag{4}$$

There are mainly three methods to compute the coefficients $\{f_i\}_{i=0}^{d}$: generalized de Casteljau method, optimal method and polynomial interpolation method. Feng et al. [35] compared three methods in detail and concluded that the polynomial interpolation method has advantages on computational cost, coding efficiency, and storage cost. Furthermore its numerical stability is good enough for rendering the algebraic patches of degrees no more than $3 \times 3 \times 3$. Rather than sampling $d + 1$ points of $f(t)$ in $[0, 1]$ uniformly, $d + 1$ Chebyshev points $\{c_i\}_{i=0}^{d}$ are adopted to reconstruct the polynomial $f(t)$ as follows:

$$c_i=\frac{1}{2}\Big[1-\cos\Big(\frac{i+1/2}{d+1}\pi\Big)\Big].$$

Then the coefficients $\{f_i\}_{i=0}^{d}$ can be obtained via a multiplication between an $(d+1)\times(d+1)$ interpolation matrix $\boldsymbol{I}$ and a $d + 1$ vector $\boldsymbol{f}$ composed of function value of the Chebyshev sampling points, where the $\boldsymbol{I}$ and $\boldsymbol{f}$ are defined as

$$\begin{cases} \boldsymbol{I}=\{I_{ij}\}_{i,j=0}^{d}=\{B_j^d(c_i)\}^{-1}, \\ \boldsymbol{f}=\{f(c_i)\}_{i=0}^{d}. \end{cases}$$

Since the Chebyshev points are fixed for a polynomial of given degree, the interpolation matrix $\boldsymbol{I}$ can be pre-computed and saved. Compared with the uniformly sampling approach, the Chebyshev sampling approach offers better numerical condition. According to our experiments, the conditional number of Chebyshev interpolation matrix $\boldsymbol{I}$ grows as an exponential function of degree whose base is 2, while the base of uniformly sampling approach is about 2.3.

4.3 Root finding algorithm

After the functional composition in Subsection 4.2, computing the ray and the algebraic patch intersection is reduced to a root finding problem of the univariate Bernstein polynomial of degree $d = m + n + q$, which equals to the degree of the ABS surface. If $d > 4$, there is no analytic solution. In our setting, the univariate Bernstein polynomial is solved numerically on GPU in parallel.

A detailed survey on root finding methods of univariate polynomials in geometric computing and computer graphics was given by authors [36]. Inspired by the iterative knot insertion approach for B-spline functions [25] and the recursive subdivision-based approach for Bernstein polynomials [37], a novel BPI root finding method is proposed. The BPI method is iterative rather than recursive. Thus it is suitable to be implemented in parallel and it can fully exploit the parallelism of contemporary GPU.

For the univariate Bernstein polynomial $f(t)$ as in Eq. (4), its image $(t, f(t))$ is a planar Bézier curve of degree d, whose control points are $\{(i/d, f_i)\}_{i=0}^{d}$. For simplicity, $f(t)$ also indicates its corresponding Bézier curve in our paper. The coefficients are normalized with a scale factor that is merely the reciprocal of the largest modulus of the coefficients in order to avoid floating-point overflow and underflow. The BPI method first subdivides $f(t)$ into a left sub-curve and a right sub-curve at the first zero t_0 of its control polygon via de Casteljau algorithm. Then the two sub-curves are checked sequentially from left to right whether their control coefficients change signs. If there is no sign change for the left sub-curve, we can purge it away safely. And if this happens to the right sub-curve also, we can determine that the polynomial has no root. Otherwise, the sub-curve is subdivided at t_1, i.e., the first zero of the control

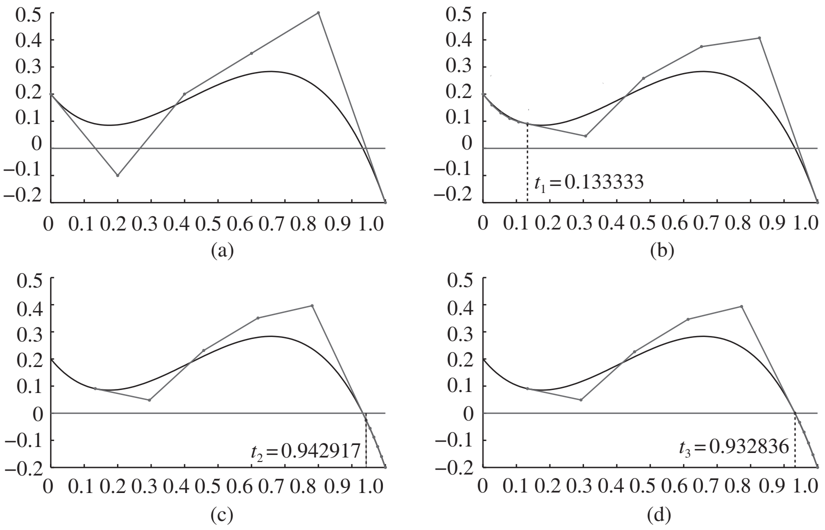


Figure 4 An example of implementing Bézier point insertion root finding algorithm. (a) The original polynomial; (b)–(d) the 1st, 2nd and 3rd Bézier point insertion results and the errors of the root. After 6 subdivision, the error of the root is less than 1.0×10^{-6}, which meets the prescribed accuracy requirement. t_i = 0.133333, 0.942917, 0.932836, 0.932724, 0.932513, 0.932518, error $|t_i - t|$ = 7.99e−1, 1.04e−2, 3.19e−4, 2.07e−4, 0.40e−5, 1.00e−6. The $\{t_2\}_{i=1}^{6}$ and $\{||t - t_i||\}_{i=1}^{6}$ give six iteration results and their numerical errors.

polygon of the sub-curve, and obtain two new sub-curves. Repeatedly performing above procedures, the sequence of inserted points $\{t_k\}$ will converge to the first zero of $f(t)$. The detailed BPI algorithm for the smallest root is described in Algorithm 1. An implementation example of the BPI algorithm is shown in Figure 4.

The BPI method can be regarded as an extension of knot insertion method [25] for the B-spline polynomial, where a d multiple knot is inserted one time. As stated in [25], the knot insertion method unconditionally and quadratically converges to a single root, while it does not require any initial guesses. Thus the proposed BPI method also quadratically and unconditionally converges to a single root. The BPI method only requires memory allocations for the coefficients of two sub-curves, which can also be reused in each insertion step. However, the knot insertion method requires keeping both the knot vector and the control coefficients of the B-Spline polynomial. Thus it consumes more memory than the BPI method [25]. Since a GPU contains very limited on-die caches and registers, the BPI method is suit for the GPU implementation. The results of the Section 6 validate that the BPI method is more efficient than the knot insertion method.

There are three ways to compute all the roots of $f(t)$. The first one is deflation approach. After the first root is found via the BPI method, the deflation is performed to reduce the degree of $f(t)$. The deflation method is perfect in theory. However it is very sensitive to the numerical error of the root. Repeatedly performing the deflations will accumulate the numerical errors rapidly [18]. Finally, the evaluated root will lose its numerical accuracy. The second one is to compute all roots in parallel by inserting Bézier points at all the zeros of the control polygon in each step. Due to the complexity of parallel programming and much more storage costs, designing an efficient parallel algorithm for all roots is difficult. We adopt the third way, i.e., computing all roots sequentially in the increasing order via the BPI method. When the smallest zero is computed, $f(t)$ is subdivided at this point. The the right sub-curve is adopted to compute the next smallest zero, and so on. The algorithm of computing all roots via the BPI method is described in Algorithm 2.

4.4 Shading and anti-aliasing

After the first hit point is obtained, the normal at the hit point should be evaluated for the shading computation. There are three methods to compute the normal. They are analytical method, difference method and fitting method.

Algorithm 1 Bézier point insertion (BPI).

```
1: Input: A Bernstein polynomial $f(t) = \sum_{i=0}^{d} f_i B_i^d(t)$ on $[0,1]$; $\tau$ is tolerance of zero function value; $\epsilon$ is tolerance
   of converged interval size;
2: if $\|f_0\| < \tau$, Output: 0; {If left end of $f(t)$ passes zero, then $t = 0$ is the first root.}
3: Initialize: $f_i^c \leftarrow f_i^l \leftarrow f_i, f_i^r \leftarrow 1$, for $i = 0, 1, \ldots, d$; $f^* \leftarrow f^c$; $t_{\text{cur}} \leftarrow t_{\text{old}} \leftarrow 1$; $t_{\text{act}} \leftarrow 0$, $\mathrm{d}t \leftarrow 1$; $\{f^c(t) = \sum_{i=0}^{d}$
   $f_i^c B_i^d(t)$ is the currently processed sub-curve, $f^l(t)$, $f^r(t)$ are the left and right sub-curve respectively. $f^*$ is a
   pointer to one of them. $t_{\text{cur}}$, $t_{\text{old}}$ are current and previous zeros of control polygons of sub-curves respectively; $t_{\text{act}}$
   is the zero with regards to $[0,1]$; $\mathrm{d}t$ is the interval size of currently processed sub-curve.$\}$
4: while $\|f_0^r\| > \tau$ do
5:   Find the smallest $k$ such that $f_k^l f_{k+1}^l < 0$;
6:   if all the coefficients of $f^l$ have the same sign then
7:     Find the smallest $k$ such that $f_k^r f_{k+1}^r < 0$;
8:     if all the coefficients of $f^r$ have the same sign. Output: $-1$;//no root.
       else
9:     $f^* \leftarrow f^r$; flag $\leftarrow$ right; $f_i^c \leftarrow f_i^r$;//purge away left sub-curve.
       end if
10:  else
11:    $f^* \leftarrow f^c$; flag $\leftarrow$ left;
12:  end if
13:  if $\frac{f_k^*}{d(f_k^* - f_{k+1}^*)} < \epsilon$, then $t_{\text{cur}} \leftarrow \frac{k}{d} + \epsilon$;
14:  if flag == left then
15:    $t_{\text{cur}} \leftarrow t_{\text{cur}} * t_{\text{old}}$;
16:  else
17:    $t_{\text{act}} \leftarrow t_{\text{act}} + \mathrm{d}t * t_{\text{old}}$; $\mathrm{d}t \leftarrow \mathrm{d}t * (1 - t_{\text{old}})$;
18:  end if
19:  Subdivide $f^*$ at $t_{\text{cur}}$ via de Casteljua algorithm, generate new sub-curves $f^l(t)$ and $f^r(t)$;
20:  $t_{\text{old}} \leftarrow t_{\text{cur}}$;
21: end while
22: Output: $t_{\text{act}} + \mathrm{d}t * t_{\text{old}}$.
```

The normal of the algebraic patch in Eq. (2) is the gradient function $\nabla F(P) = (F_x(P), F_y(P), F_z(P))$. All three derivatives are also in terms of Bernstein polynomials and their degrees are $d-1$. To compute the $\nabla F(P)$ in GPU efficiently, the coefficients of $F_x(P)$, $F_y(P)$ and $F_z(P)$ should be loaded into shared memories and additional evaluation routines should be written for three components. The difference method is to approximate the normal by the difference formulae $(F(x+\epsilon, y, z)/\epsilon, F(x, y+\epsilon, z)/\epsilon, F(x, y, z+\epsilon)/\epsilon)$ with a proper scalar ϵ. The computational cost of the difference approach is almost same with that of the analytical approach, which requires 3 function evaluations of $F(P)$. However the difference approach is more efficient for reusing GPU codes. In the fitting approach, a plane is computed to fit the hit point and its neighbor pixels. The normal of the fitting plane is adopted as the approximate normal. The three normal computation approaches can produce the similar shading effects. The fitting approach is the most efficient one since there is no additional function evaluations of $F(P)$. However it is not worked in the semi-transparent rendering case since there is no neighboring pixel information available in each step of depth peeling. Thus the fitting approach is adopted in our setting. For the semi-transparent effect rendering, the difference solution approach is adopted alternatively. An example of comparison among shading effects by using three normal computation methods are shown in Figure5.

The shaded pixels on the silhouette and boundary of the algebraic patch tend to suffer from the sampling artifact in general. To alleviate it, anti-aliasing computations should be performed for these pixels. In our implementation, a 4×4 rotated grid super-sampling (RGSS) approach is adopted [38] since it can avoid samples aligning on the horizontal or vertical axis and greatly improve anti-aliasing quality for the most commonly encountered cases. An illustration of RGSS is shown in the bottom right corner in Figure 6. To facilitate the RGSS, the pixels on the silhouette and boundary should be detected and gathered first.

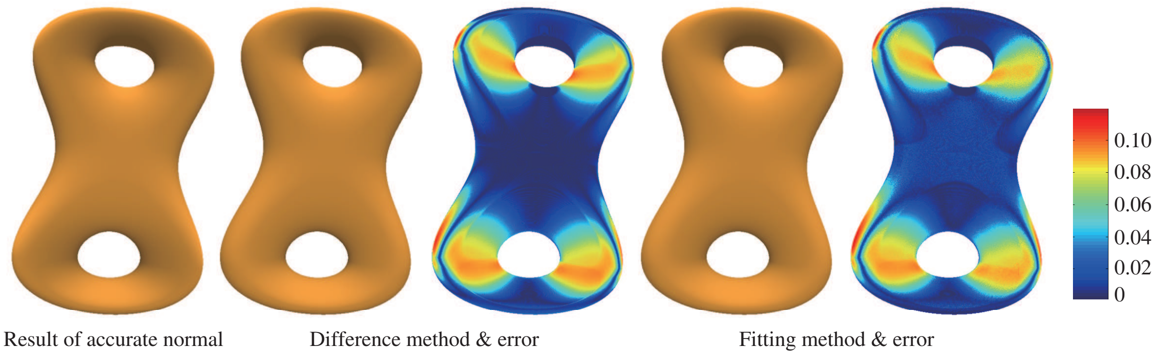


Figure 5 (Color online) Comparison among shading effects by using three different normal computation methods. The left figure is the shading result by using accurate normal. The middle and right figures show the shading results and brightness errors of in the HSV color space by using the difference method and the fitting method respectively. The errors of hue and saturation are less than 1.0e–3. They are omitted.

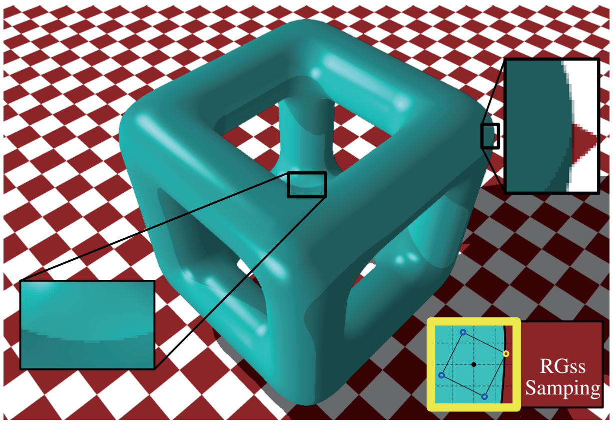


Figure 6 (Color online) Anti-aliasing rendering of silhouette. The small figure at bottom right corner shows 4 × 4 RGSS kernel.

Algorithm 2 BPI method to find all roots.

```
1: Input: $f(t)$, $\tau$, and $\epsilon$ as in Algorithm 1.
2: Output: $N_r$ the root number of $f(t)$, $[R_i]_{i=0}^{d}$ the root array.
3: if $\|f_d\| < \tau$, then rEndInter=true;
4: $f_i^c \leftarrow f_i$, for $i = 0, 1, \ldots, d$;//reuse the $f^c(t)$ in Algorithm 1;
5: repeat
6:   $t = BPI(f^c(t), \tau, \epsilon)$;
     if $t \geqslant 0 \&\& \ t \leqslant 1 - \epsilon$ then $R_{N_r++} = t$; $t+ = \epsilon$;
     else break;
     endif
7:   Subdivide $f(t)$ at $t$ via de Casteljua algorithm, choose the right sub-curves as new $f^c(t)$;
8: until $t > 1$
9: if rEndInter==true;, then $R_{N_r++} = 1$;
```

In theory, the silhouette is defined as the intersection between a surface and its polar surface with respect to the viewpoint [31]. For a given horizontally or vertically scan plane, the silhouette points are defined as the roots of the following non-linear system:

$$\begin{cases} F(x, y, z) = 0, \\ P(x, y, z, w) = x_e F_x + y_e F_y + z_e F_z + w_e F_w = 0, \\ a_k x + b_k y + c_k z + d_k = 0, \ \ k = h \text{ or } v, \end{cases} \tag{5}$$

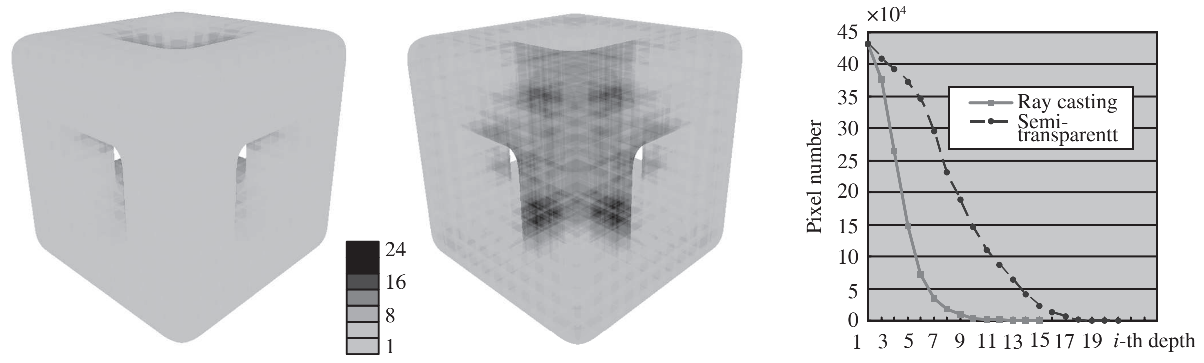


Figure 7 The left figure shows the depth color-map of the opaque algebraic patch. The middle one shows the semi-transparent case, where each ray traverses in a back-to-front order. The right one shows the pixel number distribution in each depth peeling step for the resolution 1024 × 768 pixel.

where $P(x, y, z, w)$ is the polar surface, $E(x_e, y_e, z_e, w_e)$ is the homogeneous coordinates of the viewpoint. The third linear equation is either horizontally or vertically scan plane. The system (5) can be solved via quadratically convergent Newton-Raphson method if the well-defined initial guesses are given. Unfortunately, it is non-trivial to provide a good initial guess due to the complex configurations of the surface. Furthermore, the system is ill-conditioned near the singular or self-intersection points. Thus it is not a feasible solution to detect the silhouette points via numerically solving the system (5).

Here an alternative silhouette point detection method is proposed. In the depth channel of primary rendering image, the depth gradient norm at the silhouette, singular or self-intersection point is much greater than those of the regular points on the surface. Thus the silhouette pixels in the screen space can be detected via central second order difference of depth values. If the difference is greater than a threshold, the pixel is regarded as a potential silhouette pixel. Of course, it may also be singular point pixel or self-intersection point pixel. Anyway, it is a conservative but a feasible solution. The pixels on the patch boundaries can be determined straightforwardly.

After all the pixels to be anti-aliased are gathered, each of them is subdivided into 4 × 4 sub-pixels, and we select 4 sub-pixels of them as shown in Figure 6. Another rendering pass for these sub-pixels is performed, which has the same flowchart as in Figure 2. An example of anti-aliasing rendering effect is shown in Figure 6.

5 GPU implementations in CUDA

The proposed algorithm is implemented in NVIDIA CUDA. CUDA provides a C language interface for general-purpose programming on the NVIDIA GPU. Due to its efficient scatter-gather mechanism, CUDA exposes many important parallel computing capabilities of the NVIDIA GPU so as to facilitate data-parallel computations, such as scan, reduce and sort, etc. All the operations have been released in CUDPP [39].

Figure 7 shows the depth layer distributions of the opaque and semi-transparent surfaces in depth peeling. There are total 15 and 21 layers of patches for the opaque and semi-transparent renderings. In the naive solution, a pixel will correspond to a thread in each peeling step. With the increase of peeled depth, the pixel number will decrease accordingly. Thus many GPU cores will be on idle. To fully explore the parallel capability of GPU, the “Compact” operation should be employed to reduce the pixel number in each peeling step. Besides it, the “Compact” operation can also cull the empty queues in the 1st step in Figure 2 and RGSS anti-aliasing. The “Compact” operation is accomplished by using the parallel prefix sum (Scan) operation in the CUDPP.

It is obviously that we need process different numbers of pixels in each peeling step, which belong to different patches, as shown in Figure 7. The situation is very different from those of the general algebraic surfaces rendering algorithms in [7,32]. In their settings, only one algebraic surface is considered. Thus the routines, such as function evaluation, functional composition, etc., can share the algebraic surface

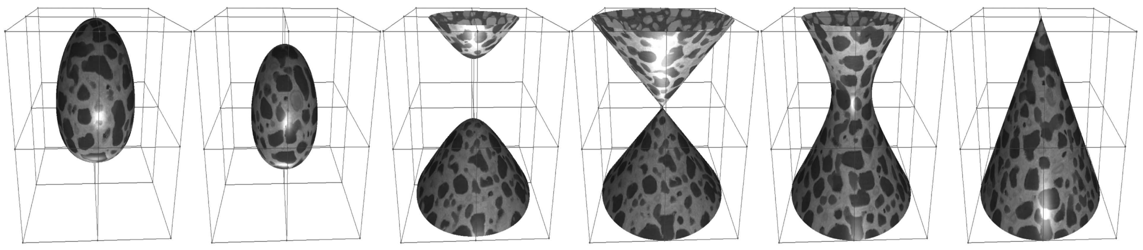


Figure 8 Morphing between three ABS surfaces of degree 6, where the 1st, 4th and 6th ones are key surfaces. The morphing is generated by interpolating the control coefficients of the key surfaces on the fly. It can achieve around 50–60 fps with 512×512 resolution.

data in the GPU. The approach cannot be extended trivially for rendering the ABS surface with many patches. To compute the intersection between a ray and an algebraic patch of degree $3 \times 3 \times 3$, 4^3 coefficients should be loaded into the on-die memory at least. If all pixels were processed in parallel trivially, i.e., each pixel will load 4^3 coefficients, it would cause heavy burden for the bandwidth and register memory of GPU. The more registers per thread allocated, the less threads worked in parallel are available. Furthermore, the simultaneous memory access of adjacent pixels may not be coalesced into a single memory transaction, which will lead to performance penalty. To address these problems, a patch-index based clustering scheme is proposed for each peeled depth, which is adopted in the 3rd step in Figure 2. Firstly, each pixel is associated with a patch index in the 3DDDA traversal. Then the pixels are sorted and clustered according to their associated patch indices. These clusters are launched in blocks. In this way, the threads in each block can share the same patch data, which can be loaded into the on-chip shared memory. This scheme provides very efficient memory access as fast as those of registers. Meanwhile the registers are saved greatly and could be applied to deeper parallelism. Therefore, the "sorting" operation is a key step in our GPU implementation, which is accomplished via the radix sort routine provided by CUDPP [39].

6 Results and comparison

The algorithm has been implemented on a PC with an Intel Core2 Q9550 2.83 GHz CPU, 4 GB Memory and an NVIDIA GeForce GTX 280 GPU. The rendering resolution of all examples is 512×512. The corresponding ABS surfaces almost fill the screen. Since the algebraic surface is capable of processing dynamic topology, an example of topological morphing between three key ABS surfaces is given in Figure 8, where the key ABS surfaces have different topologies. Since the proposed algorithm is purely on-the-fly, it can be applied to render dynamic ABS surfaces and explore their topological changes.

There are five algorithms, which are closely related to real-time rendering piecewise algebraic surface. They are analytical approach [11], Bézier clipping approach [6], B-spline knot insertion approach [7], interval arithmetic approach [24] and ray marching approach [32].

The analytical approach is designed for rendering the piecewise algebraic patches whose degrees are not greater than 4 and whose domains are tetrahedrons. Z-sorting algorithm is adopted to determined the first hit point [11]. Of course, the analytical root finding algorithm can be replaced by a numerical algorithm so that the approach can be extended to render the piecewise algebraic patches of degrees greater than 4. To render the ABS surface, the functional composition and the BPI root finding should be incorporated into the original analytical approach. However z-sorting algorithm need to compute all the hit points. Thus all the univariate polynomials along each view ray will be stored for the potential root finding, which is very storage consuming. For the example in Figure 7, almost 75% of the univariate polynomials have no root. To render the ABS surface with 720P resolution, it will exhaust all of 1GB memory of GTX280! Thus the scalability of the modified algorithm is limited. On the other hand, our depth peeling approach always reuse small amount of memory in each peeling step, which is very memory efficient.

Due to the limited precision of floating point in GPU, the numerical instability will occur for the interval arithmetic when a lot of MAD (multiplication and addition) operations are consumed to evaluate the tri-variate tensor product Bernstein polynomials. Furthermore, the small but necessary interval tolerance in the root finding algorithm may cause numerical artifacts. Thus the interval arithmetic approach [24] is not applicable to the ABS surface rendering. The adaptive marching points method proposed by Singh and Narayanan [32] isolate the root via rule of signs and approximate the root via bisection. However rule of signs is not a robust root isolation method. The bisection method can approximate first hit point efficiently, however it cannot determine the non-intersection case efficiently. The latter one is very common when a ray traverses a piecewise ABS surface. Furthermore the threshold estimation for each step is not a trivial work for the piecewise algebraic patches, since they are patch-dependent. Thus we did not implement the method for the ABS surface.

Comparisons between the BPI algorithm and the other root finding algorithms on CPU, such as RPoly, Laguerre's method, Bézier clipping, B-spline knot insertion, high degree clipping, GeoClip etc. can refer to the survey [36]. Among these methods, the Bézier clipping and the B-spline knot insertion algorithms are implemented and compared with our approach on GPU [6,7]. The original B-spline knot insertion approach is designed for a single algebraic patch [7]. Thus it contains a lot of pre-computations and transformations on CPU, which is infeasible for the ABS surface case. As the comparison, our implementation of the B-spline knot insertion approach is almost same with our proposed algorithm except for the root finding part, which is replaced as the B-spline knot insertion algorithm.

The method in [26] is 2–4 times faster than the method in this paper since it adopted different root-finding technique. However, the proposed method in this paper has three main advantages as follows, which is designed to solve the problems in [26]. The BPI method is more robust than the Newton-Raphson method, because the robustness of the Newton-Raphson method greatly depends on its initial guess of the first step. In many ill-conditioned cases, the Newton-Raphson method cannot guarantee the convergence of iterations. In contrast, the unconditioned convergence of the BPI method is proved in [25]. The method in [26] cannot display the ABS surfaces with semi-transparent effect, which is useful to reveal the complex topology of the ABS surfaces. We can adopt the proposed algorithm to find all the roots along one ray by the BPI method and depth peeling. The ABS surface is a kind of piecewise surface modeling approach, and the algorithm in [26] need extra strategies to process the boundaries between two adjacent patches, the silhouettes and the regions near silhouettes respectively. In this paper, we traverse the ABS surface patch by patch. Thus these additional steps are unnecessary.

We construct some globally C^1 and C^2 smooth ABS surfaces of degrees 6–9, as shown in Figure 9. The modeling approach of these examples goes beyond the scope of the paper and is omitted here. The termination criteria for the root-finding of Algorithm 1 is that $|f(t)| < \tau(= 1.0\text{e–}6)$ for single precision floating-point arithmetic on GPU. The degrees, knot information, numbers of patches of the ABS surfaces, the fps (frames per second) comparison statistics among the Bézier clipping approach [6], the B-spline knots insertion approach [7], our approach for opaque and semi-transparent surface rendering, are given in Table 1 in details. We can draw from the statistics that the proposed algorithm is faster than the other two algorithms. The speedups are 10%–35% and 4%–20% respectively, and the speedup decreases with the increase of depth complexity.

7 Conclusions and future work

In this paper, we present a GPU-based real-time ABS surface rendering algorithm by employing ray traversal, functional composition, and a novel root-finding algorithm—Bézier point insertion. The experimental results show that our algorithm is capable of ray casting the ABS surfaces of degrees 6-9 containing thousands of patches in real time, and the proposed root-finding algorithm is suitable for SIMD architecture of GPU and is superior to the Bézier clipping algorithm and the B-spline point insertion algorithm.

However, we can draw that the semi-transparent rendering is still slow, which is useful to exploit the complex topology of the algebraic surface. Another problem is how to efficiently determine whether an

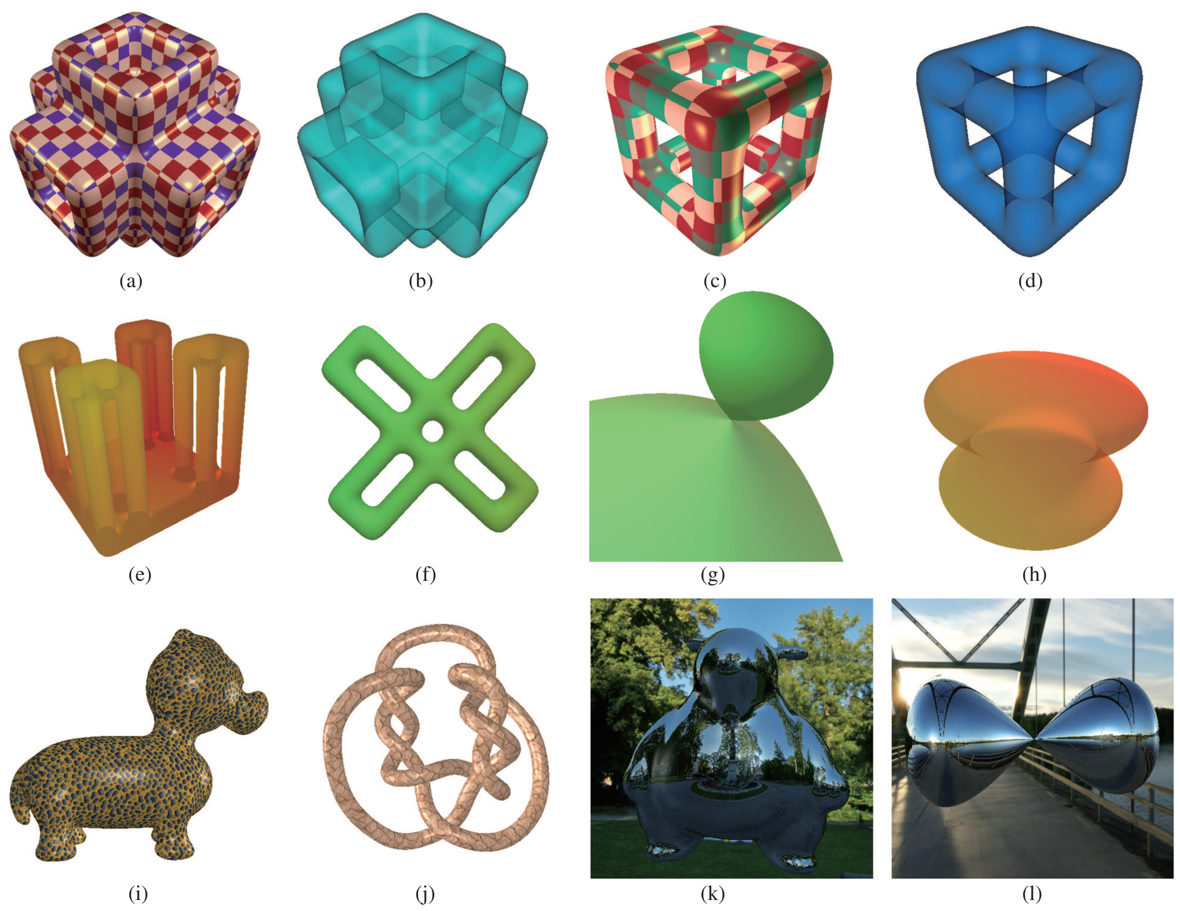


Figure 9 (Color online) The ABS surface examples.

Table 1 The statistics for the ABS surfaces in Figure 9. BC, BKI, BPI, BPI-α indicates the Bézier clipping approach, the B-spline knot insertion approach, our approach, our approach for semi-transparent rendering respectively. The runtime statistics are fps

ABS	Degree	Knot segments	Patches	BC	BKI	BPI	BPI-α
9(a)	$2 \times 2 \times 2$	$10 \times 10 \times 10$	448	9.17	9.78	10.21	4.35
9(c)	$2 \times 2 \times 2$	$8 \times 8 \times 8$	376	9.58	10.63	11.39	5.98
9(e)	$2 \times 2 \times 2$	$17 \times 17 \times 17$	1636	7.18	7.82	8.34	3.92
9(f)	$3 \times 3 \times 3$	$4 \times 18 \times 18$	1296	12.45	12.93	14.53	9.12
9(g)	$2 \times 2 \times 3$	$1 \times 1 \times 1$	1	16.92	17.72	20.03	14.83
9(h)	$2 \times 2 \times 2$	$25 \times 17 \times 17$	1020	15.27	15.98	17.85	10.44
9(i)	$2 \times 2 \times 2$	$23 \times 19 \times 19$	1063	15.28	16.20	18.10	10.46
9(j)	$2 \times 2 \times 2$	$37 \times 38 \times 38$	3622	9.53	10.17	11.09	6.40
9(k)	$2 \times 2 \times 2$	$17 \times 18 \times 18$	1083	13.79	14.33	16.11	9.87
9(l)	$3 \times 3 \times 3$	$1 \times 2 \times 2$	2	12.58	14.57	17.39	10.29

algebraic patch $F(x, y, z) = 0$ in Eq. (2) is null. In our setting, it is accomplished via sign check, which is too conservative. In future, we are looking forward to seeking more efficient algorithms to purge away no-root region before functional composition.

Acknowledgements

This work was supported by National Natural Science Foundation of China (Grant Nos. 60933007, 61170138), National Basic Research Program of China (973 Program)(Grant No. 2009CB320801), Program for New Century Excellent Talents in University (NCET-10-0728).

References

1 Patrikalakis N M, Kriezis G A. Representation of piecewise continuous algebraic surface in terms of B-splines. Visual Comput, 1989, 5: 360–374
2 Che X J, Liang X Z, Li Q. G1 continuity conditions of adjacent nurbs surfaces. Comput Aided Geom Des, 2005, 22: 285–298
3 Bajaj C L, Chen J, Xu G L. Modeling with cubic A-patches. ACM Trans Graph, 1995, 14: 103–133
4 Jüttler B, Felis A. Least-squares fitting of algebraic spline surfaces. Adv Comput Math, 2002, 17: 135–152
5 Tong W H, Feng Y Y, Chen F L. Hierarchical implicit tensor-product B-spline surface and its application in surface reconstruction (in Chinese). J Softw, 2006, 17: 11–20
6 Kanamori Y, Szego Z, Nishita T. GPU-based fast ray casting for a large number of metaballs. Comput Graph Forum, 2008, 27: 351–360
7 Reimers M, Seland J. Ray casting algebraic surfaces using the frustum form. Comput Graph Forum, 2008, 27: 361–370
8 Witkin A P, Heckbert P S. Using particles to sample and control implicit surfaces. In: Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 94), Orlando, 1994. 269–277
9 Lorensen W E, Cline H E. Marching cubes: A high resolution 3d surface construction algorithm. ACM SIGGRAPH Comput Graph, 1987, 21: 163–169
10 Bloomenthal J. An implicit surface polygonizer. In: Heckbert P, ed. Graphics Gems IV. San Diego: Academic Press Professional, Inc., 1994. 324–349
11 Loop C, Blinn J. Real-time GPU rendering of piecewise algebraic surfaces. In: Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 06), Boston, 2006. 664–670
12 Hanrahan P. Ray tracing algebraic surfaces. In: Proceedings of the 10th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 83), Detroit, 1983. 83–90
13 Kalra D, Barr A H. Guaranteed ray intersections with implicit surfaces. In: Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 89), Boston, 1989. 297–306
14 Mitchell D P. Robust ray intersection with interval arithmetic. In: Proceedings on Graphics Interface '90, Halifax, 1990. 68–74
15 Farouki R, Rajan V. On the numerical condition of polynomials in bernstein form. Comput Aided Geom Des, 1987, 4: 191–216
16 Lane J M, Riesenfeld R F. Bounds on a polynomial. Bit, 1981, 21: 112–117
17 Nishita T, Sederberg T W, Kakimoto M. Ray tracing trimmed rational surface patches. In: Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 90), Dallas, 1990. 337–345
18 Spencer M R. Polynomial real root finding in bernstein form. Dissertation for the Doctoral Degree. Provo: Brigham Young University, 1994
19 North N S. Intersection algorithms based on geometric intervals. Thesis for the Master's Degree. Provo: Brigham Young University, 2007
20 Bartoň M, Jüttler B. Computing roots of polynomials by quadratic clipping. Comput Aided Geom Des, 2007, 24: 125–141
21 Liu L G, Zhang L, Lin B B, et al. Fast approach for computing roots of polynomials using cubic clipping. Comput Aided Geom Des, 2009, 26: 547–559
22 Pabst H F, Springer J P, Schollmeyer A, et al. Ray casting of trimmed nurbs surfaces on the GPU. In: Proceedings of IEEE Symposium on Interactive Ray Tracing, Salt Lake City, 2006. 151–160
23 Knoll A, Wald I. Interactive ray tracing of arbitrary implicit functions. In: Proceedings of the 2nd IEEE/EG Symposium on Interactive Ray Tracing, Ulm, 2007. 11–18
24 Knoll A, Hijazi Y, Kensler A, et al. Fast ray tracing of arbitrary implicit surfaces with interval and affine arithmetic. Comput Graph Forum, 2009, 28: 26–40
25 Mørken K, Reimers M. An unconditionally convergent method for computing zeros of splines and polynomials. Math Comput, 2007, 76: 845–865
26 Wei F F, Feng J Q. Real-time ray casting of algebraic B-spline surfaces. Comput Graph, 2011, 35: 800–809
27 Fujimoto A, Tanaka T, Iwata K. Arts: accelerated ray-tracing system. IEEE Comput Graph Appl, 1986, 6: 16–26
28 Mammen A. Transparency and antialiasing algorithms implemented with the virtual pixel maps technique. IEEE Comput Graph Appl, 1989, 9: 43–55
29 Everitt C. Interactive order-independent transparency. White Paper, NVIDIA. 2001
30 Kajiya J T. Ray tracing parametric patches. In: Proceedings of the 9th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 82), Boston, 1982. 245–254
31 Sederberg T W, Zundel A K. Scan line display of algebraic surfaces. In: Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 89) Boston, 1989, 147–156
32 Singh J M, Narayanan P J. Real-time ray tracing of implicit surfaces on the GPU. IEEE Trans Vis Comput Graph,

2010, 16: 261–272

33 Nishita T, Nakamae E. A method for displaying metaballs by using bezier clipping. Comput Graph Forum, 1994, 13: 271–280

34 DeRose T D, Goldman R N, Hagen H, et al. Functional composition algorithms via blossoming. ACM Trans Graph, 1993, 12: 113–135

35 Feng J Q, Peng Q S. Bernstein polynomial composition through interpolation and its applications in curves and surfaces (in Chinese). J Softw, 2002, 13: 2014–2020

36 Wei F F, Zhou F, Feng J Q. Survey of real root finding of univariate polynomial equation in CAGD/CG (in Chinese). J Comput Aided Des Comput Graph, 2011, 23: 193–207

37 Rockwood A, Heaton K, Davis T. Real-time rendering of trimmed surfaces. SIGGRAPH Comput Graph, 1989, 23: 107–116

38 Beets K, Barron D. Super-sampling anti-aliasing analyzed. Technical Report, Beyond3D & 3dfx. 2000

39 Harris M, Owens J, Sengupta S, et al. http://www.gpgpu.org/developer/cudpp/

- Monograph
  Rordam M, Larsen F, Lausten N. An introduction to K-theory for C*-algebras. Cambridge: Cambridge University Press, 2000. 30—35
- Proceedings
  Qin H R. The subgroups of a finite order in K(Q). In: Bass H, Kuku A O, Pedrini C, eds. Algebraic K-theory and Its Application. Singapore: World Scientific, 1999. 600—607
- Conference proceedings
  Minor H E. Spillways for high velocities. In: Zurich V E, Minor H E, Hager W H, eds. Proceedings of International Workshop on Hydraulics of Stepped Spillways, Rotterdam, the Netherlands, 2000. 3—10
- Dissertation
  Liu G X. Classification of finite dimensional basic Hopf algebras and related topics. Dissertation for the Doctoral Degree. Hangzhou: Hangzhou University, 2005. 24—28
- Technical report
  Phillips N A. The Nested Grid Model. NOAA Technical Report NWS22. 1979
- Patent
  Zhang W P. Experiment Apparatus of Diffraction Imaging. China Patent, 02290557.X, 2003-12-03
- User manual
  Wang D L, Zhu J, Li Z K, et al. User Manual for QTKMapper Version 1.6, 1999
- Software
  Hemodynamics III: The ups and downs of hemodynamics. Version 2.2. Orlando: Computerized Educational Systems. 1993
- CD
  Anderson S C, Poulsen K B. Anderson's Electronic Atlas of Hematology. Philadelphia: Lippincott Wilkins, 2002
- Electronic version of a journal
  Christine M. Plant physiology: plant biology in the Genome Era. Science, 2003, 281: 331—332 [cited Sep. 23, 2003]. Available from: http://www. sciencemag. org/anatmorp.htm

Subscription information

ISSN print edition: 1674-733X

ISSN electronic edition: 1869-1919

Volume 57 (12 issues) will appear in 2014

Subscription rates

For information on subscription rates please contact:

Customer Service

China: sales@scichina.org

North and South America:

journals-ny@springer.com

Outside North and South America:

subscriptions@springer.com

Orders and inquiries:

China

Science China Press

16 Donghuangchenggen North Street, Beijing 100717, China

Tel: +86 10 64015683

Fax: +86 10 64016350

North and South America

Springer New York, Inc.

Journal Fulfillment, P.O. Box 2485

Secaucus, NJ 07096 USA

Tel: 1-800-SPRINGER or 1-201-348-4033

Fax: 1-201-348-4505

Email: journals-ny@springer.com

Outside North and South America:

Springer Distribution Center

Customer Service Journals

Haberstr. 7, 69126 Heidelberg, Germany

Tel: +49-6221-345-0, Fax: +49-6221-345-4229

Email: subscriptions@springer.com

Cancellations must be received by September 30 to take effect at the end of the same year.

Changes of address: Allow for six weeks for all changes to become effective. All communications should include both old and new addresses (with postal codes) and should be accompanied by a mailing label from a recent issue. According to § 4 Sect. 3 of the German Postal Services Data Protection Regulations, if a subscriber's address changes, the German Federal Post Office can inform the publisher of the new address even if the subscriber has not submitted a formal application for mail to be forwarded. Subscribers not in agreement with this procedure may send a written complaint to Customer Service Journals, Karin Tiks, within 14 days of publication of this issue.

Microform editions are available from: ProQuest. Further information available at
http://www.il.proquest.com/uni

Electronic edition

An electronic version is available at springer.com/scp

Production

Science China Press

16 Donghuangchenggen North Street, Beijing 100717, China

Tel: +86 10 64015683

Fax: +86 10 64016350

Printed in the People's Republic of China

Jointly Published by

Science China Press and Springer

Sponsored by
Chinese Academy of Sciences (CAS)
National Natural Science Foundation of China (NSFC)

Published by
Science China Press & Springer

ISSN 1674-733X