Interactive Direct Rendering of Trivariate B-spline Scalar Functions

Alon Raviv and Gershon Elber
Department of Computer Science
Technion, Israel Institute of Technology
Haifa 32000, Israel
E-mail: {alonr | gershon}@cs.technion.ac.il

Abstract

This paper presents a direct rendering paradigm of trivariate B-spline functions, that is able to incrementally update complex volumetric data sets, in the order of millions of coefficients, at interactive rates of several frames per second, on modern workstations. This incremental rendering scheme can hence be employed in modeling sessions of volumetric trivariate functions, offering interactive volumetric sculpting capabilities.

The rendering is conducted from a fixed viewpoint and in two phases. The first, preprocessing, stage accumulates the effect that the coefficients of the trivariate function have on the pixels in the image. This preprocessing stage is conducted offline and only once per trivariate and viewing direction. The second stage conducts the actual rendering of the trivariate functions. As an example, during a volumetric sculpting operation, the artist can sculpt the volume and get a displayed feedback, in interactive rates.

1 Introduction

Volume Rendering, as a subfield of *Volume Visualization*, is concerned with the rendering of volumetric data. In contrast to other methods of volumetric visualization, in *Volume Rendering* the volumetric data is processed directly.

A common way of representing the volumetric data is via a three-dimensional grid of *voxels*. Just like a two dimensional grid of pixels, a grid of voxels is a three-dimensional grid that consists of cubes that are denoted as volumetric cells or voxels. A two dimensional image is typically represented by assigning a scalar or a vector value to each pixel.

Similarly, a three-dimensional volumetric data set is represented by assigning scalar or vector values to the voxels. A three-dimensional grid of uniform voxels is also called a *voxmap*.

One approach at *Volume Rendering* is based upon algorithms which cast rays into the volume, one for each pixel in the two dimensional viewing plane. The intensities that are accumulated for each such ray form the output image. As a consequence, these algorithms are view space oriented and view point dependent. As an example, [17] and [10] present algorithms for ray casting. [17] stops the ray whenever an opaque surface is encountered, while [10] computes the color and opacities of the casted rays from the scalar values which are assigned to the voxels. Optimizations include the termination of the ray tracing as a function of the encountered opacity [11], using spatial coherence of the data [11], controlling the number of casted rays and the number of samples per ray [12] and performing sparse samples along the ray's path [2].

In virtually all existing *Volume Rendering* work, the volume is represented as a piecewise constant or sometimes as a piecewise linear trivariate function. Clearly, higher order trivariate functions can yield a smoother result and better represent freeform volumes [15]. One noticeable exception that do attempt to directly handle higher order trivariate functions is [1]. Chang, Rockwood and He [1] render *freeform volumes* instead of voxels. A freeform volume is a trivariate polynomial or rational function defined over a parametric domain that is a volume. In order to visualize the volume, the functions defined over the volume are evaluated in [1] to produce a dense set of sampled points, in object space. These points are assigned function values which are converted to intensity and opacity values. Rendering takes place by computing the intensity or opacity integral along the ray of each pixel. This computation of the integral is conducted in a back to front order and takes into account the intensities and opacities of the points which affect each pixel. Nevertheless, the intensity integral is computed numerically according to a discrete set of samples along the path of the ray inside the trivariate, instead of the exact computation over the continuous trivariate representation. In addition, interactive manipulations can not be performed because only the final value of the integral for each pixel is saved instead of the whole data set of the sampling points. Hence, changes of

individual control points require the reevaluation of a whole set of rays.

In [7], a robust method to extract the boundaries of scalar trivariate functions is proposed via the computation of the scalar field of the determinant of the Jacobian of the trivariate function. This approach is clearly more accurate than the traditional conversion into a piecewise linear or even a piecewise constant approximation that is combined with polygonal iso-surface extraction methods, such as Marching Cubes [13]. The direct manipulation of the higher order function not only provides accuracy but is also more robust, better handling the cracking, or black holes, problem as well as degeneracies due to sampling that are common to the marching Cubes method. Unfortunately, the approach of [7] is slow and cannot be used in interactive sessions.

Two examples of interactive rendering of geometry using ray casting while manipulating the geometry, are [18] and [14]. In [18], Wang and Kaufman use a local ray casting algorithm to render three-dimensional objects which are interactively modified by the user. The objects are defined via a three-dimensional density function whose value at a certain location represents the density of the object's material at that location. The density function is defined over a voxmap. Modifying the object is translated into changes in the values of the density function. In order to render the object, a ray is casted from each pixel towards the voxmap to determine the pixel's color. Wang and Kaufman measure the minimal distance of the ray from the object using the density function defined over the volume. This distance is used to create anti-aliased pixel color, thus saving the need for image space super-sampling. In addition, since only a small part of the object's volume is modified in a typical modifying operation, rays are casted only for the pixels which are affected by the modified region. As a consequence, interactive speed can be reached. Nevertheless, since the object is represented by a piecewise linear density function, the color computed for each ray is only an approximation limited by the sampling rate of the density function. In [14], Mizuno, Okada and Toriwaki also use a ray casting algorithm to render three-dimensional objects which are interactively modified by the user. The objects in [14] are represented and modeled via intersection lists between the object and each viewing ray. The intersection points at the head of each list, which are the points of the object visible to the viewer, are used to generate

the image of the object. Each time the object is modified, the rays that intersect the modified volume are re-casted and their intersection lists are updated. Interactive speed is achieved due to two reasons. The first is the relative low number of lists which have to be updated - only the lists of the view lines which intersect the modified volume. The second is the typical small size of these lists - as stems from the fact that, frequently, a ray stubs a single object at few locations only. Only intersection points between the object and the viewing rays are considered. Hence, opacity information of the object can not be employed, and only an opaque face of the object is rendered.

In [8], a comprehensive and detailed review of the field of *Volume Visualization* is given and more on the topic can be found there.

1.1 What is a Trivariate

Before presenting our rendering technique, it seems appropriate to briefly consider the kind of representation that is being used for representing the rendered objects.

Just like an (explicit) scalar bivariate surface that assigns a scalar z_0 value to every point (x_0, y_0) in the XY plane (see Figure 1), an (explicit) scalar trivariate function $q(x, y, z)$ assigns a scalar, w_0, value to every point, (x_0, y_0, z_0), in the three-dimensional space.

Like all tensor product B-spline functions, scalar trivariate tensor product B-spline functions [16, 6] have a control-mesh that consists of scalar coefficients, $P_{ijk} \in \mathbb{R}$. These trivariate B-spline functions are of the form:

$$q(u, v, w) = \sum_{i=0}^{l-1} \sum_{j=0}^{m-1} \sum_{k=0}^{n-1} P_{ijk} B_{i,o_u}(u) B_{j,o_v}(v) B_{k,o_w}(w), \tag{1}$$

where $B_{i,o_u}(u)$, $B_{j,o_v}(v)$, and $B_{k,o_w}(w)$ are the B-spline basis functions of orders o_u, o_v, and o_w respectively, P_{ijk} are the scalar coefficients in a volumetric mesh of size $l \times m \times n$, and $q(u, v, w)$ is a scalar function. In the ensuing discussion and due to their simplicity, we assume the B-spline basis functions are uniform.

The (rendered) objects are defined as implicit forms that are represented with the aid of trivariate tensor product B-spline functions. An iso-surface at level q_0 could be

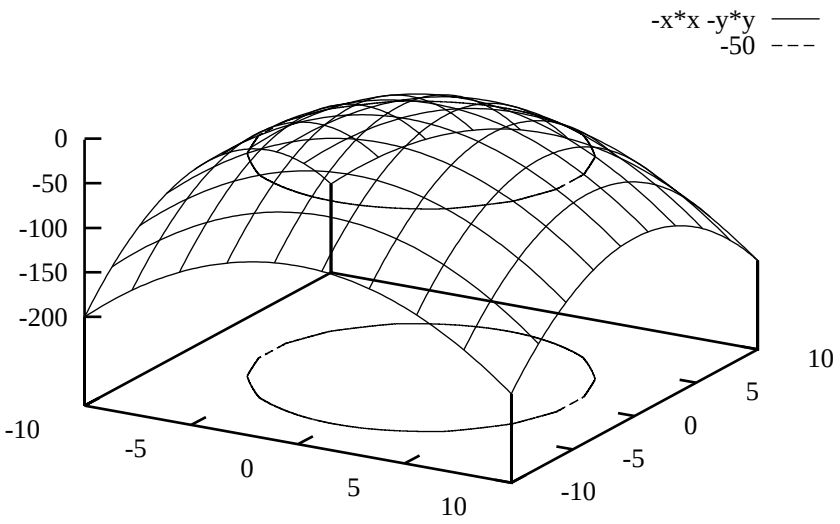


Figure 1: An example of an explicit bivariate function, $f(x, y) = -(x^2 + y^2)$. The bivariate function assigns a height to each $(x,\ y)$ location on the plane. The constant set of $f(x, y)$ is an implicit curve, $f(x, y) = -50$ in this example.

extracted from the representation, solving for $q(u, v, w) = q_0$. Alternatively, the trivariate field could be considered as a translucency field and the opacity of the object should be integrated along the line of sight. Trivariate based representations are employed, for example, in [1, 7, 15, 16], and trivariates from all these sources could be rendered using the approach presented in this work.

1.2 The Presented Work

In this work, we present an accurate interactive *Volume Rendering* technique which visualizes the opacity of an object that is represented as an implicit trivariate function. As in [1], and in contrast with [14], the end result is continuous and the degree of continuity can be controlled via the degree of the trivariate functions. Nonetheless and while [1] also handles and renders trivariate functions, in this work we strive for interactive rendering rates of several frames per second while we assume a fixed viewing direction. We will

demonstrate that every second one can incrementally update thousands of coefficients of the trivariate functions, in a sculpted scene with complex trivariate(s) that consists of millions of coefficients, hence providing the means for interactive manipulation and editing of trivariate volumes.

A line integral is computed, as an off-line preprocess, along the path of each casted ray. The evaluation of the integral is performed by summing up a collection of the scalar trivariate B-spline functions. During the preprocessing stage, the B-spline basis functions, which blend each control coefficient of the trivariate function, are pre-evaluated and their effect on each individual pixel is stored. During an interactive stage, the values of the control coefficients are modified via, for example, a sculpting or modeling environment. The modified values and the data generated in the preprocessing stage are exploited together toward the reevaluation of an updated trivariate representing the image of the object, in interactive rates.

This paper is organized as follows. Section 2 describes the preprocessing stage during which the effect of the B-spline basis functions on individual pixels is evaluated and Section 3 depicts how the interactive rendering is being conducted. Section 4 portrays some examples of rendered objects using the presented approach, and finally, in Section 5 we conclude and discuss future work.

2 Off-Line Preprocessing Towards Interactive Rendering of Trivariates

In order to conduct the rendering process of the object in interactive rates, the influence of the B-spline basis functions on each individual pixel is computed during the preprocessing stage. Section 2.1 discusses the ray casting algorithm which is used for computing the contribution and generating the coefficients for all pixels involved, whereas Section 2.2 elaborates on the computation and storage of the data and the image.

2.1 Ray Casting

The ray casting approach typically casts rays from pixels in the screen into a volumetric data set. The quantity that is accumulated for each ray originating in each pixel is converted into color or intensity and is assigned to the pixel. For example, a ray can accumulate the density of the volumetric data set along it's path, letting a higher density to be converted into a brighter color.

A ray casting algorithm typically visualizes the volumetric data set from a fixed viewpoint, which is selected by the user. Once a viewpoint has been chosen, visualization of the object and ray casting is conducted from the chosen viewpoint.

In the proposed scheme, the object is visualized by integrating the density of the trivariate tensor product B-spline functions along the path of each casted ray. Denote by P_{ijk} the value of a single scalar control coefficient in the control mesh of a trivariate tensor product B-spline function (See Equation (1)). Then,

$$\mathcal{V}_{ijk} = (i,\ i + o_u) \times (j,\ j + o_v) \times (k,\ k + o_w), \tag{2}$$

where o_u, o_v, and o_w are the orders of the B-spline basis functions on each dimension respectively, is the volume in which P_{ijk} has a positive contribution to the value of the trivariate tensor product B-spline function. In other words, $\mathcal{V}_{ijk}$ is the *support volume* in which $P_{ijk}B_{i,o_u}(u)B_{j,o_v}(v)B_{k,o_w}(w) > 0$. Since uniform knot vectors are employed, the parametric domain $(i,\ i + o_u) \times (j,\ j + o_v) \times (k,\ k + o_w)$ equals the subset of the three-dimensional space which is occupied by $\mathcal{V}_{ijk}$. Denote by $\mathcal{R}_{p_l q_l}(r)$ the ray casted through the pixel (p, q):

$$\mathcal{R}_{p_l q_l}(r) : [-\infty, \infty] \longrightarrow I\!R^3. \tag{3}$$

Since $\mathcal{R}_{p_l q_l}(r)$ is a straight line, $\mathcal{R}_{p_l q_l}(r)$ is a linear vector function of r. Employ an arc length parameterization, $\overline{\mathcal{R}}_{p_l q_l}(s)$, to $\mathcal{R}_{p_l q_l}(r)$ such that $s \in [0,\ L]$ for $\overline{\mathcal{R}}_{p_l q_l}(s) \subset \mathcal{V}_{ijk}$. That is, the ray penetrates $\mathcal{V}_{ijk}$ at $\overline{\mathcal{R}}_{p_l q_l}(0)$ and leaves $\mathcal{V}_{ijk}$ at $\overline{\mathcal{R}}_{p_l q_l}(L)$,

$$\overline{\mathcal{R}}_{p_l q_l}(s) = (u(s), v(s), w(s)), \tag{4}$$

for some linear functions $u(s), v(s), w(s)$. Note that the domain $[0,\ L]$ might be empty if $\overline{\mathcal{R}}_{p_l q_l} \bigcap \mathcal{V}_{ijk} = \emptyset$.

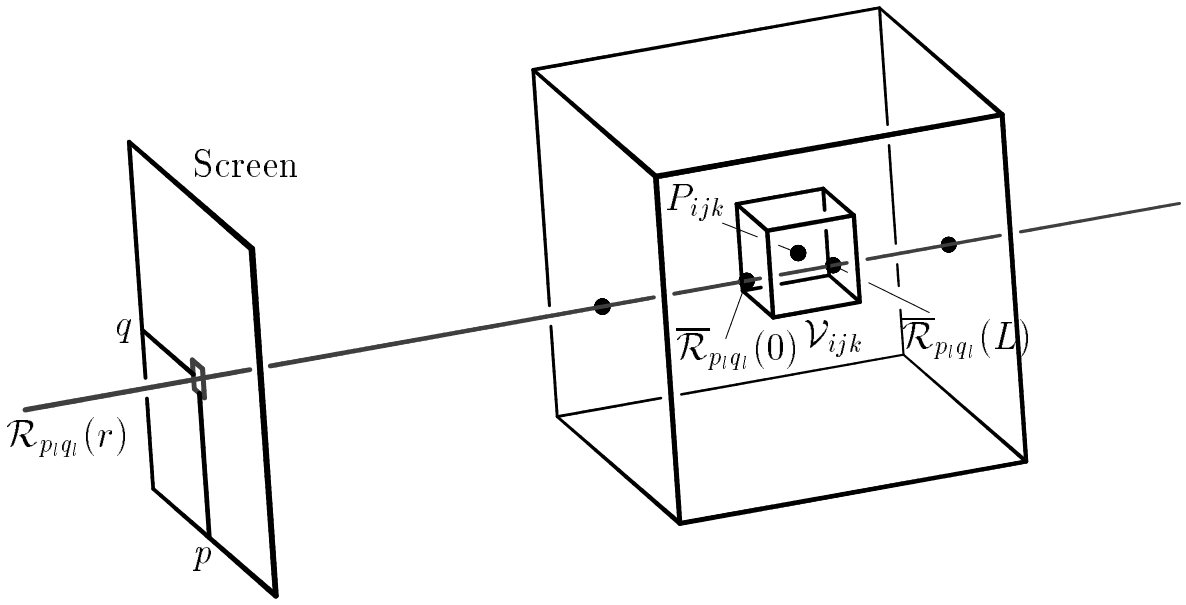


Figure 2: A ray $\mathcal{R}_{p_l q_l}(r)$ intersecting the support volume, $\mathcal{V}_{ijk}$, of a single control coefficient P_{ijk}. The large box denotes the entire volume of the trivariate function representing the object. The dots on $\mathcal{R}_{p_l q_l}(r)$ denote the intersection points between $\mathcal{R}_{p_l q_l}(r)$ and the entire volume as well as the support volume of P_{ijk}.

The density of the trivariate function is integrated along the intersection domain of $\mathcal{R}_{p_l q_l}$ and $\mathcal{V}_{ijk}$, $\overline{\mathcal{R}}_{p_l q_l}(s)$, $s \in [0, L]$. Figure 2 shows how a ray $\mathcal{R}_{p_l q_l}(r)$ intersects the volume, $\mathcal{V}_{ijk}$, affected by a single control coefficient, P_{ijk}.

2.2 Preprocessing of the Trivariates

In order to visualize the object, the orientation, or location and direction of each casted ray, is first determined according to the indices of the pixel through which the ray is casted. If an orthographic projection is employed, all of the casted rays are parallel, and therefore they all have the same direction. This direction is set to follow the view direction that is prescribed by the user. If a perspective projection is employed, the direction of each ray is determined by the viewpoint that is prescribed by the user and the location of the pixel that the ray is casted through.

Once the orientation of some ray is computed, the weighted contribution of each

control coefficient of the trivariate to the ray is evaluated and accumulated. Toward this end the following observation is crucial:

> **Observation 1** *During the interactive stage, only the values of the scalar coefficients, P_{ijk}, are modified. The B-spline basis functions that blend each coefficient, P_{ijk}, remain unchanged. Hence, the weighted contribution of each control coefficient to each ray is fixed and can be computed a-priori in the preprocessing stage, given a fixed view direction or viewpoint.*

As a consequence, during the interactive stage, the actual density of a pixel is computed via a blending between the precomputed fixed weighted contributions and the current values of all the coefficients of the trivariates.

Recalling Equation (2), $\mathcal{V}_{ijk}$ is the volume in which P_{ijk} has a positive contribution to the value of the trivariate function. The contribution of P_{ijk} to the density of some ray $\mathcal{R}_{p_l q_l}$ depends on the existence and form of the intersection between $\mathcal{R}_{p_l q_l}$ and $\mathcal{V}_{ijk}$. The weight of the contribution of P_{ijk} to the density of $\mathcal{R}_{p_l q_l}$, $w_{p_l q_l}(P_{ijk})$, is given by

$$w_{p_l q_l}(P_{ijk}) = \int_0^L B_i(u(s))B_j(v(s))B_k(w(s))ds, \tag{5}$$

where $B_i(u), B_j(v), B_k(w)$ are the B-spline basis functions blending P_{ijk} and $u(s), v(s), w(s)$ and L are defined in Equation (4). Due to the fact that the object is represented by trivariate B-spline functions, $w_{p_l q_l}(P_{ijk})$ can be computed analytically, because there exists a closed from for the product and integral of B-spline functions [3, 4] .

Let the set $\mathcal{S}_{\mathcal{R}_{p_l q_l}}$ be,

$$\mathcal{S}_{\mathcal{R}_{p_l q_l}} = \{P_{ijk} \mid w_{p_l q_l}(P_{ijk}) > 0\}. \tag{6}$$

In other words, $\mathcal{S}_{\mathcal{R}_{p_l q_l}}$ holds all the control coefficients whose weighted contribution to the density of $\mathcal{R}_{p_l q_l}$ is positive.

The intensity of light passing through translucent material decreases exponentially. See [9] for more information. The result of the computation of the density of an object along a path of a ray depends, in the general case, on the order of the integration along the path of the ray. For example, the color assigned to a ray that passes through a red

material and then through a green material is different from the color that is assigned to a ray that passes through these materials in an opposite order. Nonetheless, in the presented case, the densities accumulated along the casted rays are always mapped to a level of gray, i.e. all objects are assumed uniform in their color. Therefore, we have the following observation to our aid:

> **Observation 2** *The order of accumulating the effect of the control coefficients, P_{ijk}, in integrating the density along some ray is irrelevant and has no influence on the final result.*

Examining the total density that is accumulated by ray $\mathcal{R}_{p_l q_l}$ of pixel $(p,\ q)$ equals,

$$I_{p_l q_l} = e^{-\sum_{P_{ijk} \in \mathcal{S}_{\mathcal{R}_{p_l q_l}}} w_{p_l q_l}(P_{ijk}) P_{ijk}} = \frac{1}{\Pi_{P_{ijk} \in \mathcal{S}_{\mathcal{R}_{p_l q_l}}} e^{w_{p_l q_l}(P_{ijk}) P_{ijk}}}, \tag{7}$$

Observation 2 is made clear due to the multiplicative nature of the density function in Equation (7). The inverse of $I_{p_l q_l}$ is simply a product of terms of the form $e^{w_{p_l q_l}(P_{ijk}) P_{ijk}}$.

As mentioned above, during the interactive stage, the values of the control coefficients, P_{ijk}, are modified. Therefore, the weighted contribution of each control coefficient to each ray must be stored, or else it would be impossible to compute the change in the ray's total density as a result of the change in the value of a control coefficient. Towards this end, the preprocessing stage creates a special data structure for each control coefficient, P_{ijk}. This data structure holds references to all the rays for which the weighted contribution of the control coefficient, P_{ijk}, is positive (See Figure 3). These n rays are identified via the indices, $\{(p_l,\ q_l)\}_{l=1}^{n}$, of their respective pixels in the image plane. In other words, only the rays, $\mathcal{R}_{p_l q_l}$, through these n pixels have a positive weighted contribution from P_{ijk}, $w_{p_l q_l}(P_{ijk})$, and hence only these positive weights, $w_{p_l q_l}(P_{ijk})$, are recorded for P_{ijk}. Due to the considerable memory costs that can be expected in storing such information, the weighted contribution of the control coefficients must be reduced to the bare minimum. Two different approaches are employed, in our case, to reduce the size of this data structure:

1. By placing a bound on the maximal value of the weighted contribution of a single control coefficient to some ray, we can map this real valued set to fixed point

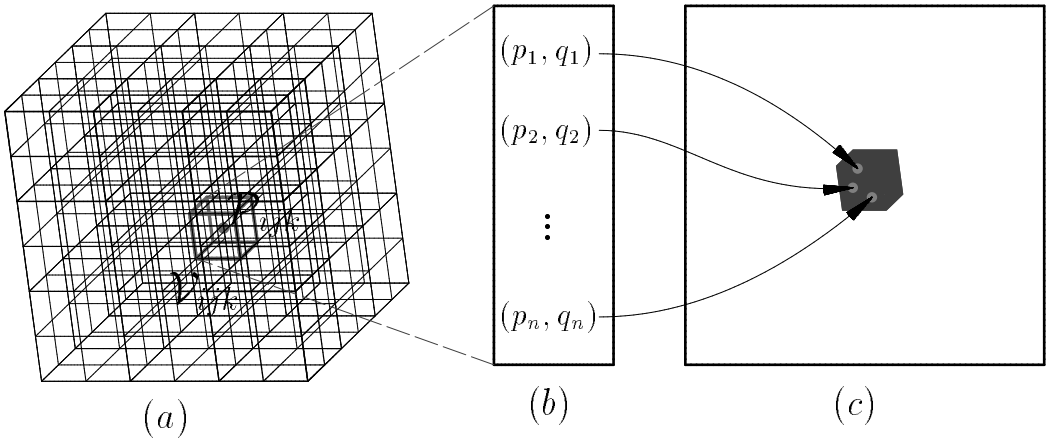


Figure 3: The preprocessing data structure contains for each coefficient in the trivariate, $\mathcal{P}_{ijk}$, and its support volume, $\mathcal{V}_{ijk}$, seen in gray in (a), a list of indices of rays, (p_l, q_l), $l = 0, \cdots, n$, in (b), that are affected by $\mathcal{P}_{ijk}$. These rays affect the pixels in the image plane as is seen in (c), inside the projection of the support volume.

representation. This maximal value is the solution of the following maximization problem:

$$\mathcal{C} = \max_{u(s),v(s),w(s)} (w_{p_l q_l}(P_{ijk})) = \max_{u(s),v(s),w(s)} \left(\int_0^L B_i(u(s)) B_j(v(s)) B_k(w(s)) ds \right), \quad (8)$$

where $B_i(u),\ B_j(v),\ B_k(w)$ are the B-spline basis functions blending a single control coefficient, P_{ijk}, over the volume $\mathcal{V}_{ijk}$, and $u(s),\ v(s),\ w(s)$ and L are defined in Equation (4). This maximization problem is discussed in Section 2.3.

2. Error dispersing in the fixed point representation. Let $\mathcal{D}$ be the number of different possible values in the fixed point representation, and $\mathcal{C}$ be the solution of Equation (8). Then, $\delta_c = \frac{\mathcal{C}}{\mathcal{D}}$ is the length of the quantization interval. In order to alleviate the effects of the truncation error, random noise in the form of a number between 0 and δ_c is added to the weighted contributions of the control coefficients before they are being converted into their fixed point counterparts. Thus, these weighted contributions are randomly scattered around the middle of the quantization interval.

2.3 Finding a Supremum for a Single Control Coefficient's Contribution

In this section, we shall compute a supremum for the weight of the contribution of a single control coefficient to a single ray ($\mathcal{C}$ in Equation (8)). As defined in Section 2.1, $\mathcal{V}_{ijk}$ is the support volume over which a control coefficient, P_{ijk}, has a positive contribution. The intersection interval between a ray and $\mathcal{V}_{ijk}$ is determined by the direction of the ray. The contribution of the control coefficient is given in Equation (5).

We shall compute the supremum for the case in which there are *cubic B-spline basis functions* on all three-dimensions. This case is selected due to the high usability of the cubic basis functions, an order we employ in most of the work presented here. For other orders, the computation would follow similar lines.

For cubic basis functions, we have,

$$\mathcal{V}_{ijk} = [i, i+4) \times [j, j+4) \times [k, k+4), \tag{9}$$

or the *support volume* of P_{ijk}. Since uniform cubic B-spline basis functions are employed, the parametric domain of each basis function, on each dimension, is $[0, 4)$. Let $R_{p_l q_l}$ be a ray which intersects $\mathcal{V}_{ijk}$. Denote by $h_i,\ h_j,\ h_k > 0$ the projections of $R_{p_l q_l} \bigcap \mathcal{V}_{ijk}$ on the axes of the parametric domain, and let $h_{max} = \max(h_i,\ h_j,\ h_k) \leq 4$. Then, without loss of generality, assume that $h_k = h_{max}$, and we have,

$$\int_0^L B_i(u(s))B_j(v(s))B_k(w(s))ds \leq \frac{2}{3}\frac{2}{3}\int_0^L B_k(w(s))ds, \tag{10}$$

because $B_i(t) \leq \frac{2}{3},\ \forall t$, for uniform cubic B-spline basis functions. For a uniform cubic B-spline function defined over the parametric domain $[0, 4)$,

$$\int_0^4 B_k(t)dt = 1. \tag{11}$$

Along the k direction, and in the worst case, the entire positive domain of $B_k(t)$ is affecting $R_{p_l q_l}$ and hence we have $B_k(t)\ |_{t=0,4} = B_k(\frac{4s}{L})\ |_{s=0,L}$, or $w(s) = \frac{4s}{L}$. Therefore,

$$\begin{aligned} \int_0^L B_k(w(s))ds &= \int_0^L B_k\left(\frac{4s}{L}\right)ds \\ &= \int_0^L B_k\left(\frac{4s}{L}\right)\frac{ds}{dt}dt \end{aligned}$$

$$
\begin{aligned}
&= \int_0^4 B_k(t)\frac{L}{4}dt \\
&= \frac{L}{4}\int_0^4 B_k(t)dt \\
&= \frac{L}{4}. \qquad (12)
\end{aligned}
$$

Finally, in the worst case, the intersection interval between $R_{p_l q_l}$ and $\mathcal{V}_{ijk}$ is the main diagonal of $\mathcal{V}_{ijk}$, and $L = 4\sqrt{3}$. Hence,

$$\mathcal{C} \leq \frac{2}{3}\frac{2}{3}\int_0^L B_k(w(r))dr = \frac{4}{9}\frac{L}{4} \leq \frac{4}{9}\sqrt{3}. \qquad (13)$$

Figure 4 illustrates two extreme cases. For cases where h_{max} does not cover all of the parametric domain of B_k, the contribution will be smaller because the integral will be computed only on a sub-interval of the parametric domain of B_k. Hence, $\mathcal{C} \leq \frac{4}{9}\sqrt{3}$, in Equation (8). While this bound of $\mathcal{C}$ is clearly not tight, the example of the axis parallel ray in Figure 4 yields

$$\int_0^L B_i(u(s))B_j(v(s))B_k(w(s))ds = \int_0^L B_i(2)B_j(2)B_k(s)ds = \frac{2}{3}\frac{2}{3}\int_0^4 B_k(s)ds = \frac{4}{9},$$

or a $\sqrt{3}$ factor from the established bound for $\mathcal{C}$ of $\frac{4}{9}\sqrt{3}$.

Concluding the preprocessing stage, it should be noted that while the resulting data structure is large, a single data structure could serve the three front, side, and up views. Having a symmetric trivariate with similar knot sequences and orders in u, v, and w, by swapping the x, y, and z indices of the coefficients, one can employ the same preprocessing data structure for all three views.

3 Interactive Rendering

As depicted in Section 2.2, each control coefficient, P_{ijk}, in each trivariate patch holds all the references to the rays for which the weighted contribution of the control coefficient is positive. As already stated, during the interactive stage, the object is modified by changing the values of control coefficients in the trivariate functions' control meshes. Thus, after a reshaping operation is conducted, the exact subset of control coefficients which were modified is known. For each control coefficient in this subset, the densities

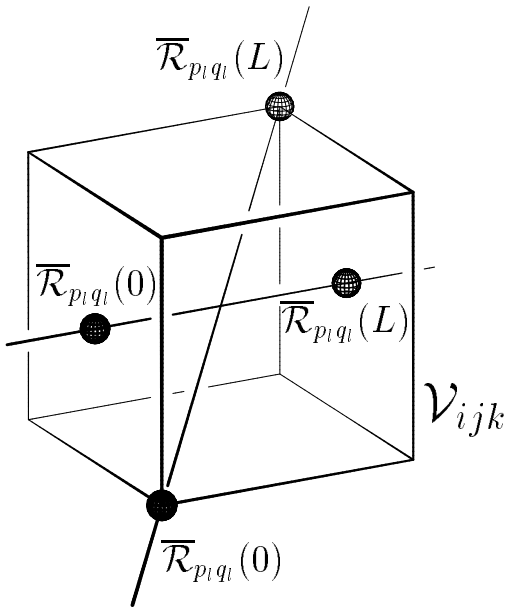


Figure 4: Two extreme cases of an intersection between a ray and $\mathcal{V}_{ijk}$.

of all the rays for which the control coefficient holds references to, must be updated. Let the old and new values of the coefficient P_{ijk} be denoted by P_{ijk}^{old} and P_{ijk}^{new}, respectively. Then, the density of a single ray is updated due to change in the coefficient P_{ijk} by

$$I_{p_l q_l}^{new} = I_{p_l q_l}^{old} \frac{e^{-P_{ijk}^{new} w_{p_l q_l}(P_{ijk})}}{e^{-P_{ijk}^{old} w_{p_l q_l}(P_{ijk})}} = I_{p_l q_l}^{old} e^{-(P_{ijk}^{new} - P_{ijk}^{old}) w_{p_l q_l}(P_{ijk})}. \tag{14}$$

Again, recall that Equation (14) holds due to the multiplicative nature that has been observed in Equation (7). Thus, updating the value of pixel $(p,\ q)$ due to a change in a single coefficient, P_{ijk}, requires a small and fixed set of operations that is independent of the rest of the coefficients affecting pixel $(p,\ q)$.

The introduced visualizing algorithm can employ transparency to convey the object. The generated image is a gray level image, in which a pixel with a brighter color denotes an object with a higher density along the path of the ray. This transparency computation can be combined with shading. In order to generate the shading information, normals must be derived. These normals can be derived out of the trivariate tensor product B-spline functions which represent the object.

Let the boundary of the object be an iso-surface of the scalar trivariate. This bound-

ary is detected by the first location along a ray's path in which the value of the trivariate function representing the object is above a predefined threshold. Then, the normals are computed on this iso-surface that forms the boundary of the object. Unfortunately, complete rays must be re-casted in order to reevaluate the iso-surface and the normals, every time some coefficient changes. Clearly, reshaping operations can now become much slower. Nevertheless, due to the fact that only a small region of the object is typically modified by each modeling operation, only the rays that affect some P_{ijk} in the modified region, should be recasted. The evaluations of normals continue to yield interactive modeling speeds of several frames per second for fairly complex trivariate mesh resolutions of approximately a hundred coefficients cubed, as will be demonstrated in Section 4

Finally, the shaded image and the transparent image of the object can be linearly blended together, allowing the user to control the shading based upon the illumination or upon the object's density.

4 Examples

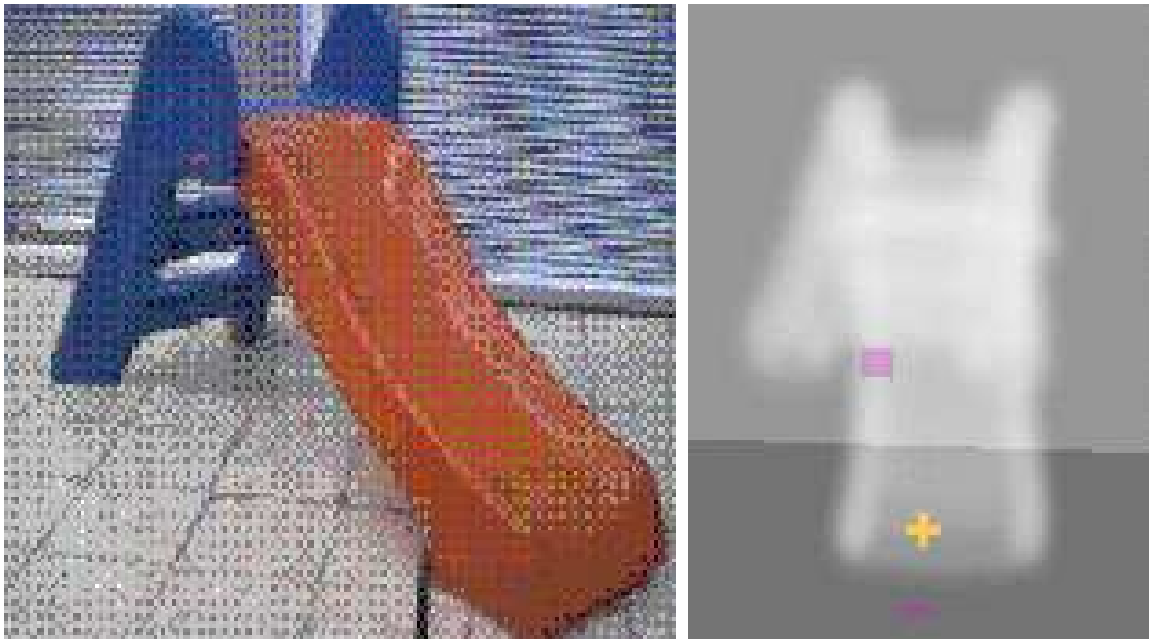
This section presents examples of objects that were rendered using the visualization scheme introduced in this work. As already stated, any trivariate function could be similarly rendered, such as the function from [1, 7, 15, 16].

The developed direct rendering tool is based on the Glut [5] toolkit from SGI that is compatible with both SGI and PC WinNT environments. All the examples in this section were recorded off an Onyx SGI having a reality engine graphics board. While the data structures are large and take significant amount of time to compute, they are computed only once. Further, these data structures allow us to achieve rendering updates of several frames per second. One byte per weighted contribution has been used in all presented examples. The size of the two dimensional images for all presented examples is 512×512 pixels. All the objects in the presented examples were modeled by assigning the coefficients of the trivariate function a value of zero at a location with no material, a value of one hundred at a location with a material of the highest density,

and intermediate values at locations with material of intermediate densities. We seek the iso-surface at which the value of the trivariate scalar function is fifty, locations that are considered to be on the boundary of the object. Thus, locations at which the value of the trivariate function is above fifty are inside the object, and locations at which the value of the trivariate function is below fifty are outside the object.

Figure 5 shows a kid's slide visualized by the presented direct volume visualization technique. The original real slide is presented in the photograph of Figure 5 (a). The computerized model of the slide was reverse engineered from the real slide, using the approach presented in [15], into a scalar trivariate function whose transparent image is shown in Figure 5 (b). Figure 6 shows the same slide object shaded. Figure 6 (a) shows a blend between the gray level transparent image and the shaded image, and Figure 6 (b) shows the shaded image of the slide, with normals computed along the iso-surface boundary. Figure 7 shows the same slide object with the addition of another trivariate patch used to model the caption on the slide's slope. Preprocessing the trivariate from the displayed viewing direction took about four hours on an 180Mhz R10000 SGI machine. As described in Section 2.2, the data computed during the preprocessing stage consists of the weighted contributions of each control coefficient of the trivariate functions to each ray casted from a pixel in the screen. We shall refer to this data as the *preprocessing data*. The size of the preprocessing data of the slide object shown in Figure 5 is about 250 *MB*. The size of the control mesh of the trivariate function representing the slide object is $68 \times 74 \times 107$ coefficients. Rendering a complete gray level transparent image of the whole trivariate function representing the slide, on an 180Mhz R10000 SGI machine, takes about 119 seconds. Direct rendering of a shaded image of the same trivariate takes about 29 seconds. The time required for an update and a rerender due to an interactive change of a single control coefficient, when no shading information is generated, is about 3.5×10^{-5} seconds. For direct rendering with shading, about 1.2×10^{-4} additional seconds are required for each control coefficient.

Figure 8 shows a chair visualized by the presented direct volume visualization technique. The model of the chair was also reverse engineered from the real chair presented in Figure 8 (a) using the approach presented in [15], into a trivariate function whose

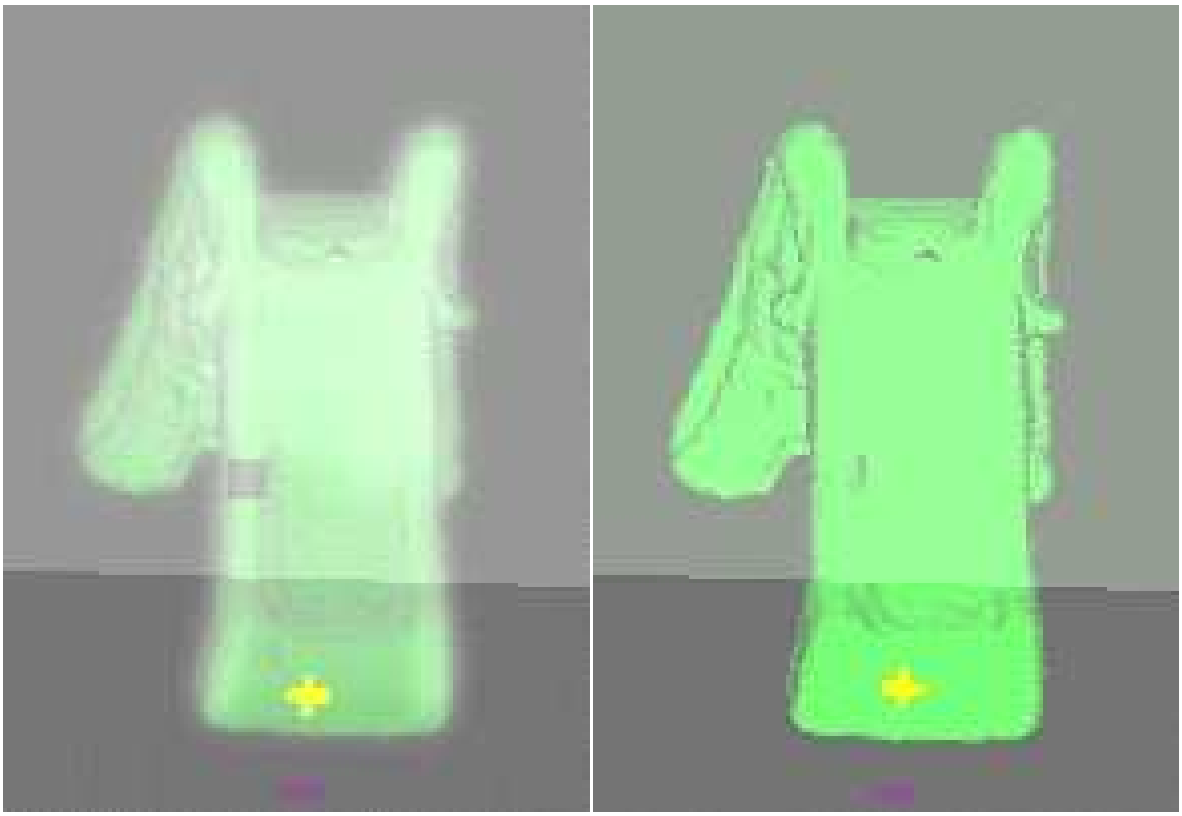


(a) (b)

Figure 5: A photograph of a real kids slide that is reverse engineered into a trivariate scalar function representation is presented in (a). The reverse engineered model is visualized transparently in (b) by the presented direct volume visualization technique. The cross marker seen in (b) is the cursor of the tool used to model the shape.

transparent image is shown in Figure 8 (b). Figure 8 (c) shows the same chair, this time shaded. Preprocessing the trivariate from the displayed viewing direction took about seven hours. The size of the preprocessing data is about 400 *MB*. The size of the control mesh of the trivariate function representing the chair object is $91 \times 90 \times 80$ coefficients. Rendering a gray level transparent image of the whole trivariate function representing the chair, on an 180Mhz R10000 SGI machine, takes about 186 seconds. Rendering a shaded image of the same trivariate takes about 42 seconds. The time required for an update and a rerender due to an interactive change of a single control coefficient, when no shading information is generated, is about 4.5×10^{-5} seconds. For rendering with shading information, about 1.4×10^{-4} additional seconds are required for each control coefficient.

Figure 9 (a) shows a trivariate model of a human face reconstructed from data generated by a three-dimensional scanner, and then visualized by the presented direct



(a) (b)

Figure 6: The kids' slide from Figure 5 shown in different levels of shading. (a) shows a blending between the gray level transparent image and the shaded opaque image, and (b) shows the shaded opaque image of the slide. The cross marker seen in (a) and (b) is the cursor of the tool used to model the shape

volume visualization technique. Figure 9 (b) shows the human face shaded, and with normals computed along the iso-surface boundary. Preprocessing the visualization data took about twenty hours. The size of the preprocessing data is about one *GB*. The size of the control mesh of the trivariate function representing the human face object is $121 \times 121 \times 121$ coefficients. Rendering a gray level transparent image of the whole trivariate function representing the human face, on an 180Mhz R10000 SGI machine, takes about 491 seconds. Rendering a shaded image of the same trivariate takes about 65 seconds. The time required for an update and a rerender due to an interactive change of a single control coefficient, when no shading information is generated, is about 4.6×10^{-5}

(a) (b)

Figure 7: The kids' slide from Figure 5 with the addition of another trivariate patch used to model the caption on the slope of the slide. (a) shows the slide visualized transparently and (b) shows the shaded opaque image of the slide.

seconds. For rendering with shading information, about 1.9×10^{-4} additional seconds are required for each control coefficient. Finally, Figure 10 presents the model of the face, after some editing operations, adding two horns to the face.

Tables 1 and 2 summarize all the results of this section. In all examples, the time to update a single coefficient, in both the transparent and the shaded mode, was in the order of less than a millisecond. Hence, hundreds if not thousands of coefficients could be updated per second while achieving a rendering rate of at least a frame per second. In the above examples, sculpting tools that affects less than a hundred coefficients at a time, where typically used. Having an object of about one hundred cubed coefficients, the tool typically affects 4^3 or 5^3 coefficients. Having about ten samples per second

(a) (b) (c)

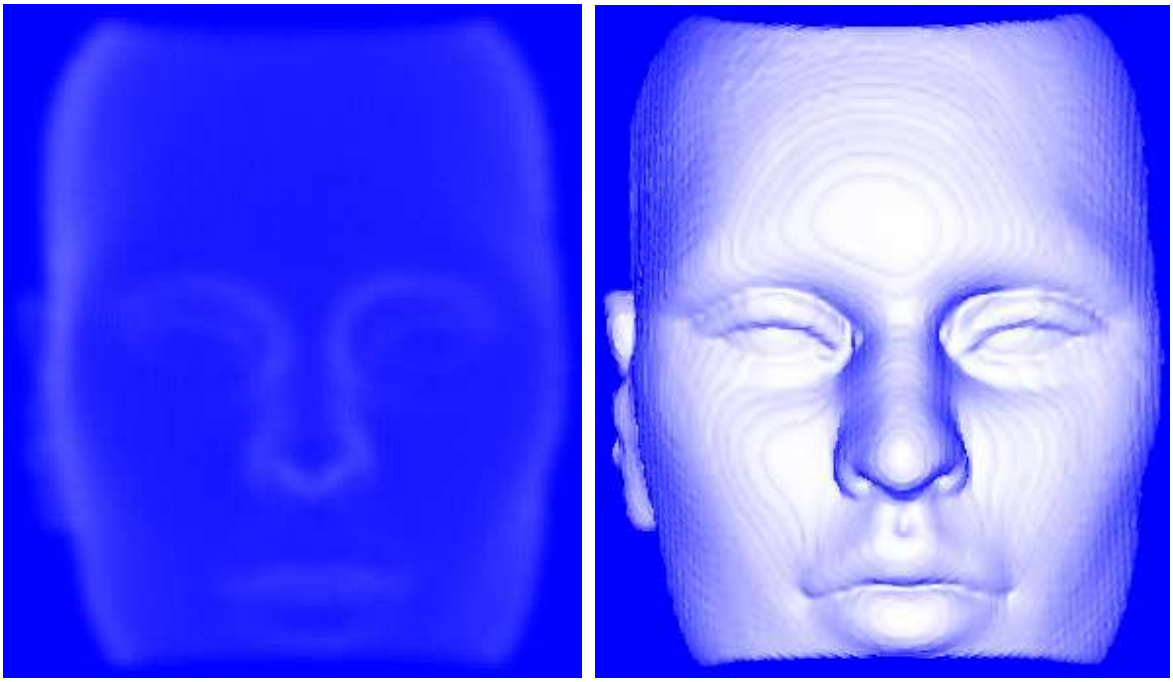
Figure 8: A photograph of a real garden chair that is reverse engineered into a trivariate scalar function representation is presented in (a). In (b) and (c), the reverse engineered model of the chair is visualized by the presented direct volume visualization technique. (b) shows a transparent image of the chair, and (c) shows a shaded opaque image. The cross marker seen in (b) and (c) is the cursor of the tool used to model the shape.

Object	Figure(s)	Pre-processing Time	Pre-processing Data Size	Mesh Size of Trivariate
Slide	5, 6	4 hours	250Mbyte	$68 \times 74 \times 107$
Chair	8	7 hours	400Mbyte	$91 \times 90 \times 80$
Face	9	20 hours	1Gbyte	$121 \times 121 \times 121$

Table 1: A summary of the preprocessing results shown in this section, computed on an SGI Onyx with 180Mhz R10000 cpu.

of the motion of the tool and one ends up with about a thousand coefficients that are updated per second.

The size of the computed data structure, in the preprocessing stage is large. Recall that a single data structure could support the three views of front, side, and up for a trivariate function with identical knot sequences and orders in the three direction. In fact, due to the independence on the order of the coefficients in the line integral, six different views ($\pm x$, $\pm y$, $\pm z$) could be derived by swapping the x, y, and z coefficients.



(a) (b)

Figure 9: A transparent image of the human face visualized by the presented direct volume visualization technique. (a) shows a transparent image of the face, and (b) shows a shaded opaque image.

Object	Rendering of a whole Volume (Secs.)		Rendering of a Single Coefficient (Secs.)	
	No Shading	With Shading	No Shading	With Shading
Slide	118	29	3.5×10^{-5}	1.55×10^{-4}
Chair	182	42	4.5×10^{-5}	1.85×10^{-4}
Face	491	65	4.6×10^{-5}	2.36×10^{-4}

Table 2: A summary of the rendering times for the objects shown in this section, computed on an SGI Onyx with 180Mhz R10000 cpu.

Moreover, these views need not necessarily be axes parallel and a single preprocessing data structure could serve six different views, if the knot sequences and orders are identical in the three axes.

Have a preprocessing data structure with a significant size to handle, the Onyx machine we have employed has 192 *MB* of real memory and therefore this machine

Figure 10: The same object as in Figure 9 after further modeling of two horns.

was unable to hold this entire preprocessing data set in real memory. As a result, local sculpting operations could be made interactively while large interactive motion commands experienced delays due to the memory swapping needs.

5 Conclusions and Future Work

In this work, we have presented a three-dimensional interactive rendering scheme of uniform trivariate B-spline functions. The continuous trivariate representation yields several advantages like the ability to transparently visualize the object by employing accurate analytic computations of the object's density along the casted rays, the capacity to transparently visualize objects with a non-uniform density, and the capability to interactively render the modifications which the object undergoes during a modeling process.

Possible future work includes:

- Transparently visualizing the object from arbitrary viewpoints - As stated in this

work, the visualization of the object can be currently conducted from a fixed viewpoint. Updating the transparent image of the object after some modeling operation involves simple arithmetic computations and can be performed interactively. By preprocessing and preparing the necessary data structures from several viewpoints, one can interactively update several images from several viewpoints during the modeling process. Consequently, the user can interactively select a viewpoint from which he/she would like to view the object. Nevertheless, the size of the preprocessed data is large and effort should be invested in minimizing this size.

- Transparently visualizing objects with a non-uniform color - As described in Section 2.2, Equation (7) holds due to the fact that the rendered objects are considered to have a uniform color. Therefore, the order in which the control coefficients are integrated is of no importance, and the transparent image of the object can be interactively updated as depicted by Equation (14). Finding an efficient method for integrating the coefficients according to their distance from the user, for visualization and interactive rendering, will enable one to transparently visualize objects of various colors or properties.

References

[1] Y.K. Chang, A.P. Rockwood, Q. He. Direct Rendering of Freeform Volumes. *Computer Aided Design*, 27, pp 553-558, July 1995.

[2] D. Cohen and Z. Sheffer. Proximity clouds - an Acceleration Technique for 3D Grid Traversal. *The Visual Computer*, 11, pp 27-38, 1994.

[3] G. Elber and E. Cohen. Second Order Surface Analysis Using Hybrid Symbolic and Numeric Operators. *Transactions on Graphics*, Vol 12, No 2, pp 160-178, April 1993.

[4] G. E. Farin. Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide. *Academic Press*, Inc. 4th Edition 1996.

[5] Glut 3.0. http://reality.sgi.com/employees/mjk_asd/glut3/glut3.html.

[6] J. Hoschek and D. Lasser. Fundamentals of Computer Aided Geometric Design *A.K. Peters* 1993.

[7] K. I. Joy and M. A. Duchaineau. Boundary Determination for Trivariate Solids. The Seventh *Pacific Graphics* conference of Computer Graphics and Applications, pp 82-91, Seoul, Korea, October 1999.

[8] A.F. Kaufman. Volume Visualization. *IEEE Computer Society Press* 1990.

[9] P. Lena, F. Lebrun and F. Mignard. Observational Astrophysics. *Springer* 1998.

[10] M. Levoy. Display of Surfaces from Volume Data. *IEEE Computer Graphics and Applications*, 8, 5, pp 29-37, May 1988.

[11] M. Levoy. Efficient Ray Tracing of Volume Data. *ACM Transactions on Graphics*, 9, 3, pp 245-261, July 1990.

[12] M. Levoy and R. Whitaker. Gaze-Directed Volume Rendering. *Computer Graphics*, 24, 2, pp 217-223, March 1990.

[13] W.E. Lorensen and H.E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *Computer Graphics, Siggraph* 21(4), pp. 163-169, July 1987.

[14] S. Mizuno, M. Okada and J. Toriwaki. Virtual Sculpting and Virtual Woodcut Printing. *Visual Computer* 10, pp 39-51, 1998.

[15] A. Raviv and G. Elber. Three Dimensional Freeform Sculpting Via Zero Sets of Scalar Trivariate Functions. *Computer Aided Design*, 32, 8/9, pp 513-526, July/August 2000. Also the fifth ACM/IEEE Symposium on Solid Modeling and Applications, Ann Arbor, Michigan, pp 159-166, June 1999.

[16] T.W. Sederberg and S.R. Parry. Free-Form Deformation of Solid Geometric Models. *Computer Graphics, Siggraph* 20(4), pp 151-159, August 1986.

[17] H.K. Tuy and L.T. Tuy. Direct 2-D Display of 3-D Objects. *IEEE Computer Graphics and Applications*, 4, 10, pp 29-33, October 1984.

[18] W. Wang and A.F. Kaufman. Volume Sculpting. *Symposium on Interactive 3D Graphics Proceedings, ACM Press*, pp 151-156, April 1995.