Beyond Trilinear Interpolation: Higher Quality for Free

BALÁZS CSÉBFALVI, Budapest University of Technology and Economics, Hungary

In volume-rendering applications, it is a de facto standard to reconstruct the underlying continuous function by using trilinear interpolation, and to estimate the gradients for the shading computations by calculating central differences on the fly. In a GPU implementation, this requires seven trilinear texture samples: one for the function reconstruction, and six for the gradient estimation. In this paper, for the first time, we show that the six additional samples can be used not just for gradient estimation, but for significantly improving the quality of the function reconstruction as well. As the additional arithmetic operations can be performed in the shadow of the texture fetches, we can achieve this quality improvement for free without reducing the rendering performance at all. Therefore, our method can completely replace the standard trilinear interpolation in the practice of GPU-accelerated volume rendering.

CCS Concepts: • **Computing methodologies** → **Volumetric models**; **Image processing**; **Texturing**.

Additional Key Words and Phrases: GPU-accelerated volume rendering, Catmull-Rom spline interpolation.

ACM Reference Format:
Balázs Csébfalvi. 2019. Beyond Trilinear Interpolation: Higher Quality for Free. *ACM Trans. Graph.* 38, 4, Article 56 (July 2019), 8 pages. https://doi.org/10.1145/3306346.3323032

1 INTRODUCTION

Volumetric data sets are usually obtained by sampling a trivariate function that represents a continuous 3D phenomenon. In order to visualize the original continuous phenomenon, the underlying function needs to be reconstructed between the sample positions. This requires a convolution of the discrete samples by an interpolation filter if the sample positions are assumed to be defined by the grid points of a regular sampling grid. In the practice of volume rendering, trilinear interpolation is one of the most popular resampling techniques, as it represents a widely accepted trade-off between image quality and rendering speed. This is especially true for GPU-accelerated volume rendering [Cabral et al. 1994; Engel et al. 2006; Krüger and Westermann 2003; Westermann and Ertl 1998], where a hardwired implementation of trilinear texture fetching is available. Although higher-order filters, such as the tricubic B-spline [Marschner and Lobb 1994; Mitchell and Netravali 1988] or the tricubic Catmull-Rom spline [Catmull and Rom 1974; Keys 1981], guarantee higher image quality [Marschner and Lobb 1994], they are significantly slower to evaluate even if their state-of-the-art GPU implementations [Csébfalvi 2018; Ruijters et al. 2008; Sigg and Hadwiger 2005] are used. In this paper, we propose a resampling technique that is as efficient as a trilinear interpolation combined with on-the-fly central differencing, but still provides much higher reconstruction quality (see Figure 1). Therefore, it can potentially become a new standard tool for volume resampling.

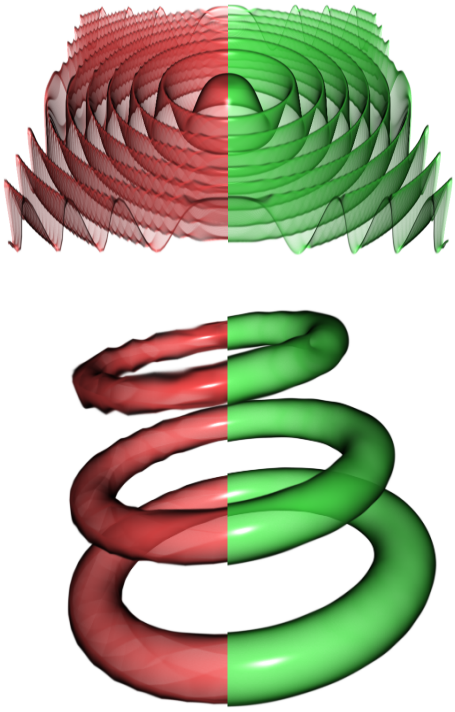


Fig. 1. Semitransparent rendering of implicit surfaces using trilinear interpolation (red) and our method (green) for reconstructing the underlying trivariate function from its discrete sampled representation. Our method significantly reduces the artifacts introduced by the trilinear interpolation but still guarantees the same rendering efficiency. The gradients are estimated from six trilinear samples in both cases, but our method reuses these samples for improving the quality of the function reconstruction.

2 RELATED WORK

Theoretically, higher-order reconstruction filtering has been thoroughly studied in the literature taking both sampling-theoretical [Li et al. 2004; Marschner and Lobb 1994; Mitchell and Netravali 1988; Theußl et al. 2000] and approximation-theoretical aspects [Blu et al. 1999; Blu and Unser 1999a,b; Condat et al. 2005; Csébfalvi 2008; Möller et al. 1997, 1998] into account. However, only few papers propose fast GPU implementations for higher-order filters. In 2005, Sigg and Hadwiger published a pioneering work [Sigg and Hadwiger 2005] on an efficient evaluation of tricubic B-spline filtering on the

Author's address: Balázs Csébfalvi, Budapest University of Technology and Economics, Department of Control Engineering and Information Technology, Magyar tudósok körútja 2, Budapest, H-1117, Hungary, cseb@iit.bme.hu.

0730-0301/2019/7-ART56 $15.00
https://doi.org/10.1145/3306346.3323032

GPU. Their method exploits that, in modern GPUs, the cost of a trilinear texture fetch is nearly the same as that of a nearest-neighbor texture fetch. Instead of a brute-force implementation, which would take 64 nearest-neighbor texture samples, tricubic B-spline filtering was proposed to be evaluated as a linear combination of only eight trilinear samples. As this approach drastically reduces the number of texture fetches, which is the bottleneck in GPU implementations in general, a significant speed-up can be achieved. Ruijters et al. [Ruijters et al. 2008] improved this method by evaluating the sample positions for the eight trilinear samples on the fly rather than precalculating them in a lookup table. Recently, Csébfalvi recognized that it is not trivial to adapt the method by Sigg and Hadwiger [Sigg and Hadwiger 2005] to filter kernels that take also negative values, and redesigned the original algorithm for an efficient Catmull-Rom spline interpolation [Csébfalvi 2018]. In spite of all these optimizations, GPU-accelerated higher-order filtering is mainly used for rendering isosurfaces by ray casting [Csébfalvi 2018; Hadwiger et al. 2009, 2005; Sigg and Hadwiger 2005], where gradients are necessary to evaluate only for the first-hit surface points. To use these techniques also for semitransparent volume rendering, gradients need to be evaluated for each sample position along the viewing rays. However, an on-the-fly central differencing or the calculation of the analytic derivatives would require 48 and 24 additional texture fetches, respectively [Hadwiger et al. 2005]. This is a significant extra cost, which dramatically reduces the rendering performance. Alternatively, the derivatives can also be precalculated and stored for each voxel [Westermann and Ertl 1998]. In this case, the GPU can simultaneously interpolate between the function values and the derivatives, but this approach increases the memory requirements by a factor of four. Overall, all previous GPU-accelerated higher-order filtering techniques [Csébfalvi 2018; Hadwiger et al. 2005; Ruijters et al. 2008; Sigg and Hadwiger 2005] achieve higher quality either at the cost of higher storage requirements or at the cost of lower rendering performance. In contrast, it is important to emphasize that using our method is not a trade-off, as it is the first third-order filtering technique that does not require an extra storage cost and still guarantees the same rendering efficiency as a lower-quality trilinear filtering combined with on-the-fly central differencing.

3 OUR METHOD IN 1D

For the sake of clarity, we explain our method in 1D first, and then we discuss its 3D extension in Section 4. Without a loss of generality, assume that the samples of a continuous function $f(x)$ are known at integer positions and $f(x)$ needs to be approximately reconstructed from these discrete samples. A piecewise linear reconstruction at position x is defined as follows:

$$f(x) \approx f_{\text{lin}}(x) = f[\lfloor x \rfloor] \cdot (1-u) + f[\lfloor x \rfloor + 1] \cdot u, \tag{1}$$

where $u = x - \lfloor x \rfloor$. The first-order derivative of $f(x)$ can be estimated by calculating central differences on $f_{\text{lin}}(x)$ [Pratt 2001]:

$$f'(x) \approx d(x) = [f_{\text{lin}}(x+1) - f_{\text{lin}}(x-1)]/2. \tag{2}$$

The key idea of our method is to reuse the linear samples $f_{\text{lin}}(x-1)$, $f_{\text{lin}}(x)$, and $f_{\text{lin}}(x+1)$ to improve the reconstruction of $f(x)$ in terms of approximation order. To do so, first, we estimate the second-order derivative of $f(x)$ by using a discrete Laplacian operator [Pratt 2001]:

$$f''(x) \approx s(x) = f_{\text{lin}}(x-1) - 2f_{\text{lin}}(x) + f_{\text{lin}}(x+1). \tag{3}$$

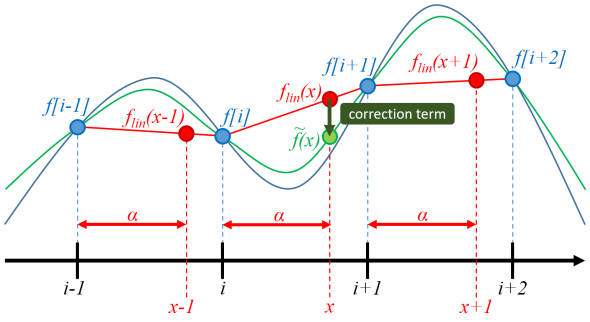


Fig. 2. Illustration of our method in 1D. The three linear samples $f_{\text{lin}}(x-1)$, $f_{\text{lin}}(x)$, and $f_{\text{lin}}(x+1)$ are used for derivative estimation and for improving the quality of the function reconstruction as well.

Based on the estimated second-order derivative, we modify the result of the linear interpolation $f_{\text{lin}}(x)$ between the discrete samples by adding a correction term in order to compensate the error of the piecewise linear approximation (see Figure 2). This correction term is positive if the second derivative is negative and, vice versa, it is negative if the second derivative is positive. More precisely, we propose the following reconstruction scheme:

$$f(x) \approx \tilde{f}(x) = f_{\text{lin}}(x) + \underbrace{s(x) \cdot \frac{1}{2}(\alpha^2 - \alpha)}_{\text{correction term}}, \tag{4}$$

where $x = i + \alpha$ for $i \in \mathbb{Z}$ and $\alpha \in [0, 1)$ being the integer and fractional parts of x, respectively. Note that, in 1D, our method formulated by Equation 4 leads to the same result as a Catmull-Rom spline interpolation:

$$\begin{aligned}
&f_{\text{lin}}(x) + s(x) \cdot \tfrac{1}{2}(\alpha^2 - \alpha) = \\
&f_{\text{lin}}(x) + [f_{\text{lin}}(x-1) - 2f_{\text{lin}}(x) + f_{\text{lin}}(x+1)] \cdot \tfrac{1}{2}(\alpha^2 - \alpha) = \\
&f[i] \cdot (1-\alpha) + f[i+1] \cdot \alpha + \\
&[f[i-1] \cdot (1-\alpha) + f[i] \cdot \alpha] \cdot \tfrac{1}{2}(\alpha^2 - \alpha) - \\
&[f[i] \cdot (1-\alpha) + f[i+1] \cdot \alpha] \cdot (\alpha^2 - \alpha) + \\
&[f[i+1] \cdot (1-\alpha) + f[i+2] \cdot \alpha] \cdot \tfrac{1}{2}(\alpha^2 - \alpha) = \\
&f[i-1] \cdot (-\tfrac{1}{2}\alpha^3 + \alpha^2 - \tfrac{1}{2}\alpha) + f[i] \cdot (\tfrac{3}{2}\alpha^3 - \tfrac{5}{2}\alpha^2 + 1) + \\
&f[i+1] \cdot (-\tfrac{3}{2}\alpha^3 + 2\alpha^2 + \tfrac{1}{2}\alpha) + f[i+2] \cdot (\tfrac{1}{2}\alpha^3 - \tfrac{1}{2}\alpha^2) = \\
&\sum_{l=-\infty}^{\infty} f[l] \cdot \varphi_{\text{CR}}(x-l),
\end{aligned} \tag{5}$$

where φ_{CR} is the well-known Catmull-Rom spline kernel [Catmull and Rom 1974; Keys 1981]:

$$\varphi_{\text{CR}}(t) = \begin{cases} \frac{3}{2}|t|^3 - \frac{5}{2}|t|^2 + 1 \text{ if } |t| \le 1 \\ -\frac{1}{2}|t|^3 + \frac{5}{2}|t|^2 - 4|t| + 2 \text{ if } 1 < |t| \le 2 \\ 0 \text{ otherwise.} \end{cases} \tag{6}$$

Thus, according to our evaluation scheme, a 1D Catmull-Rom spline interpolation can be equivalently evaluated from three samples linearly interpolated at positions $x-1$, x, and $x+1$.

4 3D EXTENSION

In 3D, the samples of a trivariate function $f(x,y,z)$ are assumed to be known for integer coordinates and $f(x,y,z)$ needs to be approximately reconstructed from these discrete samples. In volume-rendering applications, it is a de facto standard to use trilinear interpolation between the discrete samples and to estimate gradients by calculating central differences on the trilinearly reconstructed underlying function. A trilinear reconstruction is defined as follows:

$$\begin{aligned} f(x,y,z) \approx f_{\text{trilin}}(x,y,z) = \\ f[\lfloor x\rfloor,\lfloor y\rfloor,\lfloor z\rfloor]\cdot(1-u)\cdot(1-v)\cdot(1-w)+ \\ f[\lfloor x\rfloor+1,\lfloor y\rfloor,\lfloor z\rfloor]\cdot u\cdot(1-v)\cdot(1-w)+ \\ f[\lfloor x\rfloor,\lfloor y\rfloor+1,\lfloor z\rfloor]\cdot(1-u)\cdot v\cdot(1-w)+ \\ f[\lfloor x\rfloor+1,\lfloor y\rfloor+1,\lfloor z\rfloor]\cdot u\cdot v\cdot(1-w)+ \\ f[\lfloor x\rfloor,\lfloor y\rfloor,\lfloor z\rfloor+1]\cdot(1-u)\cdot(1-v)\cdot w+ \\ f[\lfloor x\rfloor+1,\lfloor y\rfloor,\lfloor z\rfloor+1]\cdot u\cdot(1-v)\cdot w+ \\ f[\lfloor x\rfloor,\lfloor y\rfloor+1,\lfloor z\rfloor+1]\cdot(1-u)\cdot v\cdot w+ \\ f[\lfloor x\rfloor+1,\lfloor y\rfloor+1,\lfloor z\rfloor+1]\cdot u\cdot v\cdot w, \end{aligned} \tag{7}$$

where $u = x-\lfloor x\rfloor$, $v = y-\lfloor y\rfloor$, and $w = z-\lfloor z\rfloor$. Using central differences [Pratt 2001], the gradient of $f(x,y,z)$ can be estimated as

$$\nabla f(x,y,z) = \begin{bmatrix} \frac{\partial}{\partial x}f(x,y,z) \\ \frac{\partial}{\partial y}f(x,y,z) \\ \frac{\partial}{\partial z}f(x,y,z) \end{bmatrix} \approx \begin{bmatrix} d_x(x,y,z) \\ d_y(x,y,z) \\ d_z(x,y,z) \end{bmatrix} = \tag{8}$$

$$\begin{bmatrix} [f_{\text{trilin}}(x+1,y,z) - f_{\text{trilin}}(x-1,y,z)]/2 \\ [f_{\text{trilin}}(x,y+1,z) - f_{\text{trilin}}(x,y-1,z)]/2 \\ [f_{\text{trilin}}(x,y,z+1) - f_{\text{trilin}}(x,y,z-1)]/2 \end{bmatrix}.$$

In order to extend our reconstruction scheme described by Equation 4 to 3D, we propose the following nonseparable extension:

$$\begin{aligned} f(x,y,z) \approx \tilde{f}(x,y,z) = f_{\text{trilin}}(x,y,z) + s_x(x,y,z)\cdot(\alpha^2-\alpha)/2 \\ + s_y(x,y,z)\cdot(\beta^2-\beta)/2 \\ + \underbrace{s_z(x,y,z)\cdot(\gamma^2-\gamma)/2}_{\text{correction terms}} \end{aligned} \tag{9}$$

where $x = i+\alpha$, $y = j+\beta$, and $z = k+\gamma$ for $i,j,k \in \mathbb{Z}$ and $\alpha,\beta,\gamma \in [0,1)$ being the integer and fractional parts of x, y, and z, respectively, while the second-order partial derivatives are estimated by using the discrete Laplacian operator along the three major axes separately:

$$\begin{bmatrix} \frac{\partial^2}{\partial x^2}f(x,y,z) \\ \frac{\partial^2}{\partial y^2}f(x,y,z) \\ \frac{\partial^2}{\partial z^2}f(x,y,z) \end{bmatrix} \approx \begin{bmatrix} s_x(x,y,z) \\ s_y(x,y,z) \\ s_z(x,y,z) \end{bmatrix} = \tag{10}$$

$$\begin{bmatrix} f_{\text{trilin}}(x-1,y,z) - 2f_{\text{trilin}}(x,y,z) + f_{\text{trilin}}(x+1,y,z) \\ f_{\text{trilin}}(x,y-1,z) - 2f_{\text{trilin}}(x,y,z) + f_{\text{trilin}}(x,y+1,z) \\ f_{\text{trilin}}(x,y,z-1) - 2f_{\text{trilin}}(x,y,z) + f_{\text{trilin}}(x,y,z+1) \end{bmatrix}.$$

Note that we use exactly the same six additional trilinear samples for evaluating the three correction terms in Equation 9 that are used for gradient estimation in Equation 8. It is easy to see that our method formulated by Equation 9 is interpolating as the correction terms are equal to zero at the grid points, while the remaining trilinear interpolation exactly reproduces the original discrete samples.

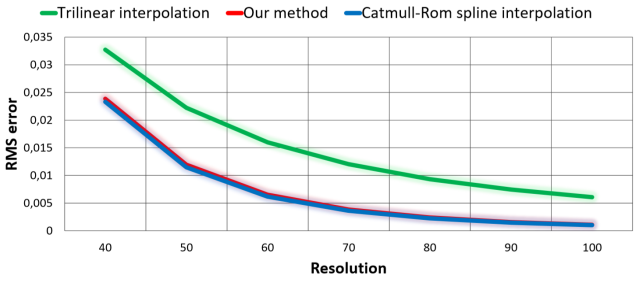


Fig. 3. RMS error of the reconstructed ML signal depending on the sampling resolution, which is set to be the same along the three major axes. The error curve of our method is practically the same as that of the Catmull-Rom spline interpolation, which is significantly below the error curve belonging to trilinear interpolation.

5 POLYNOMIAL RECONSTRUCTION AND APPROXIMATION ORDER

In this section, we show that our method is equivalent to Catmull-Rom spline interpolation in a sense that it is also able to exactly reproduce quadratic polynomials. Such a quadratic polynomial is defined as

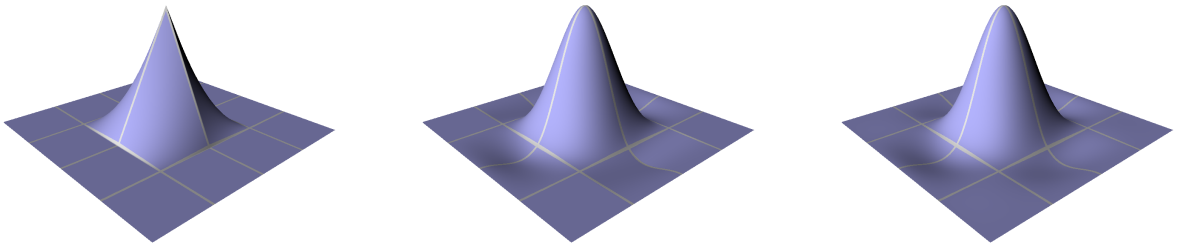
$$f(x,y,z) = Ax^2+By^2+Cz^2+Dyz+Exz+Fxy+Gx+Hy+Iz+J. \tag{11}$$

Let us first express the result of the trilinear interpolation by substituting Equation 11 into Equation 7:

$$\begin{aligned} f_{\text{trilin}}(x,y,z) = f_{\text{trilin}}(i+\alpha,j+\beta,k+\gamma) = \\ f[i,j,k]\cdot(1-\alpha)\cdot(1-\beta)\cdot(1-\gamma)+ \\ f[i+1,j,k]\cdot\alpha\cdot(1-\beta)\cdot(1-\gamma)+ \\ f[i,j+1,k]\cdot(1-\alpha)\cdot\beta\cdot(1-\gamma)+ \\ f[i+1,j+1,k]\cdot\alpha\cdot\beta\cdot(1-\gamma)+ \\ f[i,j,k+1]\cdot(1-\alpha)\cdot(1-\beta)\cdot\gamma+ \\ f[i+1,j,k+1]\cdot\alpha\cdot(1-\beta)\cdot\gamma+ \\ f[i,j+1,k+1]\cdot(1-\alpha)\cdot\beta\cdot\gamma+ \\ f[i+1,j+1,k+1]\cdot\alpha\cdot\beta\cdot\gamma = \\ \\ Ai^2+A2i\alpha+A\alpha+Bj^2+B2j\beta+B\beta+Ck^2+C2k\gamma+C\gamma+ \\ Dyz+Exz+Fxy+Gx+Hy+Iz+J. \end{aligned} \tag{12}$$

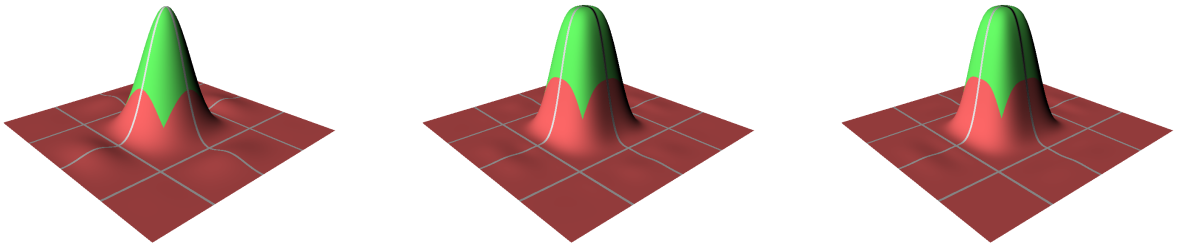
Note that s_x, s_y, and s_z defined by Equation 10 exactly reproduce the second-order homogeneous partial derivatives of $f(x,y,z)$, since substituting Equation 12 into Equation 10, we obtain:

$$\begin{bmatrix} s_x(x,y,z) \\ s_y(x,y,z) \\ s_z(x,y,z) \end{bmatrix} = \begin{bmatrix} 2A \\ 2B \\ 2C \end{bmatrix} = \begin{bmatrix} \frac{\partial^2}{\partial x^2}f(x,y,z) \\ \frac{\partial^2}{\partial y^2}f(x,y,z) \\ \frac{\partial^2}{\partial z^2}f(x,y,z) \end{bmatrix}. \tag{13}$$



(a) Trilinear filter. (b) Our tricubic filter. (c) Catmull-Rom spline filter.

Fig. 4. 2D cross sections ($x \in [-2, 2]$, $y \in [-2, 2]$, $z = 0$) of impulse responses corresponding to trilinear interpolation (a), our method (b), and tricubic Catmull-Rom spline interpolation (c). Note that the difference between the impulse response of our method and that of the Catmull-Rom spline interpolation is hardly recognizable.



(a) Trilinear filter.
Postaliasing: 0.042498
Oversmoothing: 0.746361

(b) Our tricubic filter.
Postaliasing: 0.069355
Oversmoothing: 0.558539

(c) Catmull-Rom spline filter.
Postaliasing: 0.073159
Oversmoothing: 0.533303

Fig. 5. 2D cross sections ($\nu_x \in [-2, 2]$, $\nu_y \in [-2, 2]$, $\nu_z = 0$) of frequency responses corresponding to trilinear interpolation (a), our method (b), and tricubic Catmull-Rom spline interpolation (c). Frequency coefficients corresponding to the passband and the stopband are depicted in green and red, respectively. The postaliasing and oversmoothing effects were measured as proposed by Marschner and Lobb [Marschner and Lobb 1994]. According to these measures, our method shows practically the same frequency-domain behavior as the Catmull-Rom spline interpolation.

Therefore, we can evaluate Equation 9 as follows:

$$
\begin{aligned}
&f_{\text{trilin}}(x, y, z) + A \cdot (\alpha^2 - \alpha) + B \cdot (\beta^2 - \beta) + C \cdot (\gamma^2 - \gamma) = \\
&Ai^2 + A2i\alpha + A\alpha + Bj^2 + B2j\beta + B\beta + Ck^2 + C2k\gamma + C\gamma + \\
&Dyz + Exz + Fxy + Gx + Hy + Iz + J + \\
&A \cdot (\alpha^2 - \alpha) + B \cdot (\beta^2 - \beta) + C \cdot (\gamma^2 - \gamma) = \\
&A(i + \alpha)^2 + B(j + \beta)^2 + C(k + \gamma)^2 + \\
&Dyz + Exz + Fxy + Gx + Hy + Iz + J = \\
&Ax^2 + By^2 + Cz^2 + Dyz + Exz + Fxy + Gx + Hy + Iz + J = \\
&f(x, y, z).
\end{aligned}
\tag{14}
$$

Thus, our method described by Equation 9 can indeed exactly reproduce quadratic polynomials. As a consequence, according to the Strang-Fix conditions [Strang and Fix 1971], it guarantees a third-order approximation just as the standard tensor-product extension of the Catmull-Rom spline interpolation. The approximation order determines how fast the error of the approximation converges to zero if the sampling frequency is increased [Blu and Unser 1999b]. In Figure 3, the asymptotic error behavior of our method is compared to that of the trilinear and Catmull-Rom spline interpolations. Note that our method results in practically the same error curve as the Catmull-Rom spline interpolation. The error curves were generated by measuring the RMS error between the analytically defined Marschner-Lobb (ML) test signal [Marschner and Lobb 1994] and its reconstructions corresponding to different sampling resolutions.

6 IMPULSE RESPONSE

Equation 5 implies that the correction term in Equation 4 can be expressed by the following convolution:

$$
s(x) \cdot \frac{1}{2}(\alpha^2 - \alpha) = \sum_{l=-\infty}^{\infty} f[l] \cdot \varphi_{\text{corr}}(x - l), \tag{15}
$$

where the corresponding filter is $\varphi_{\text{corr}}(t) = \varphi_{\text{CR}}(t) - \varphi_{\text{lin}}(t)$ for $\varphi_{\text{lin}}(t) = \max(0, 1 - |t|)$ being the tent filter belonging to linear interpolation. Note that, in 3D (see Equation 9), each correction term can be expressed by separate convolutions along the three major axes, where the correction kernel $\varphi_{\text{corr}}(t)$ is used along one axis and the linear kernel $\varphi_{\text{lin}}(t)$ is used along the other two axes. Thus, the 3D impulse response of our method defined by Equation 9 can be expressed as

$$
\begin{aligned}
&\varphi(x, y, z) = \varphi_{\text{lin}}(x)\varphi_{\text{lin}}(y)\varphi_{\text{lin}}(z) + \varphi_{\text{corr}}(x)\varphi_{\text{lin}}(y)\varphi_{\text{lin}}(z) + \\
&\varphi_{\text{lin}}(x)\varphi_{\text{corr}}(y)\varphi_{\text{lin}}(z) + \varphi_{\text{lin}}(x)\varphi_{\text{lin}}(y)\varphi_{\text{corr}}(z).
\end{aligned}
\tag{16}
$$

It is interesting to mention that $\varphi(x, y, z)$ is equal to the Catmull-Rom spline along the major axes. For example, for $y, z = 0$, we obtain $\varphi(x, y, z) = \varphi(x, 0, 0) = \varphi_{\text{lin}}(x) + \varphi_{\text{corr}}(x) = \varphi_{\text{CR}}(x)$, since $\varphi_{\text{lin}}(y) = \varphi_{\text{lin}}(z) = \varphi_{\text{lin}}(0) = 1$ and $\varphi_{\text{corr}}(y) = \varphi_{\text{corr}}(z) = \varphi_{\text{corr}}(0) = 0$ in Equation 16. Furthermore, along a grid line, only those discrete samples have a contribution to the reconstructed function that are intersected by the given grid line. As a consequence, our method is equivalent to a Catmull-Rom spline interpolation along the edges of the cubic cells, and fits a tricubic polynomial onto the edge profiles in between. Figure 4 shows the impulse response $\varphi(x, y, z)$ of our method compared to that of the trilinear interpolation and the Catmull-Rom spline interpolation. In fact, $\varphi(x, y, z)$ represents a very good approximation of the 3D separable extension of the Catmull-Rom spline. Between these kernels we measured an RMS error of 0.0036, while between the trilinear kernel and the tricubic Catmull-Rom spline we measured an order of magnitude higher RMS error of 0.0269.

7 FREQUENCY RESPONSE

It is easy to derive the frequency response $\hat{\varphi}(\nu_x, \nu_y, \nu_z)$ of our method by simply taking the Fourier transform of Equation 16:

$$\hat{\varphi}(\nu_x, \nu_y, \nu_z) = \hat{\varphi}_{\text{lin}}(\nu_x)\hat{\varphi}_{\text{lin}}(\nu_y)\hat{\varphi}_{\text{lin}}(\nu_z) + \hat{\varphi}_{\text{corr}}(\nu_x)\hat{\varphi}_{\text{lin}}(\nu_y)\hat{\varphi}_{\text{lin}}(\nu_z) + \hat{\varphi}_{\text{lin}}(\nu_x)\hat{\varphi}_{\text{corr}}(\nu_y)\hat{\varphi}_{\text{lin}}(\nu_z) + \hat{\varphi}_{\text{lin}}(\nu_x)\hat{\varphi}_{\text{lin}}(\nu_y)\hat{\varphi}_{\text{corr}}(\nu_z), \tag{17}$$

where the frequency response of the linear tent filter is $\hat{\varphi}_{\text{lin}}(\nu) = \frac{\sin^2(\pi\nu)}{(\pi\nu)^2}$, while the frequency response of the correction kernel is $\hat{\varphi}_{\text{corr}}(\nu) = \hat{\varphi}_{\text{CR}}(\nu) - \hat{\varphi}_{\text{lin}}(\nu)$. Thus, the well-known Fourier transform of the Catmull-Rom spline [Mitchell and Netravali 1988] completes the definition of $\hat{\varphi}(\nu_x, \nu_y, \nu_z)$:

$$\hat{\varphi}_{\text{CR}}(\nu) = \frac{3}{(\pi\nu)^2}[\text{sinc}^2(\nu) - \text{sinc}(2\nu)] + \frac{1}{(\pi\nu)^2}[-3\text{sinc}^2(2\nu) + 2\text{sinc}(2\nu) + \text{sinc}(4\nu)]. \tag{18}$$

Figure 5 shows the frequency response $\hat{\varphi}(\nu_x, \nu_y, \nu_z)$ of our method compared to that of the trilinear interpolation and the Catmull-Rom spline interpolation. Note that our method shows practically the same frequency-domain behavior as the Catmull-Rom spline interpolation. Furthermore, just as the frequency response of the tricubic Catmull-Rom spline, $\hat{\varphi}(\nu_x, \nu_y, \nu_z)$ guarantees at least third-order zero-crossings at integer coordinates except the origin. Let us first consider the case when $\nu_x, \nu_y, \nu_z \in \mathbb{Z}$ and at least two coordinates are not equal to zero. In Equation 17, each of the four terms ensures at least fourth-order zero-crossings as, for non-zero integer positions, $\hat{\varphi}_{\text{lin}}$ and $\hat{\varphi}_{\text{CR}}$ are well-known to result in zero-crossings of order two and three, respectively. In other cases, when at least two coordinates are equal to zero, $\hat{\varphi}(\nu_x, \nu_y, \nu_z)$ takes the same values as a tensor-product extension of $\hat{\varphi}_{\text{CR}}$. For example, if $\nu_y, \nu_z = 0$ then $\hat{\varphi}(\nu_x, \nu_y, \nu_z) = \hat{\varphi}(\nu_x, 0, 0) = \hat{\varphi}_{\text{lin}}(\nu_x) + \hat{\varphi}_{\text{corr}}(\nu_x) = \hat{\varphi}_{\text{CR}}(\nu_x)$, since $\hat{\varphi}_{\text{lin}}(\nu_y) = \hat{\varphi}_{\text{lin}}(\nu_z) = \hat{\varphi}_{\text{lin}}(0) = 1$ and $\hat{\varphi}_{\text{corr}}(\nu_y) = \hat{\varphi}_{\text{corr}}(\nu_z) = \hat{\varphi}_{\text{corr}}(0) = 0$ in Equation 17. Therefore, $\hat{\varphi}(\nu_x, 0, 0)$ is equal to the frequency response of the Catmull-Rom spline at ν_x, and as such results in the same third-order zero-crossings for $\nu_x \in \mathbb{Z}$ and $\nu_x \neq 0$. Thus, for arbitrary integer coordinates except the origin, $\hat{\varphi}(\nu_x, \nu_y, \nu_z)$ guarantees at least third-order zero-crossings, which is the frequency-domain formulation of the well-known Strang-Fix condition [Strang and Fix 1971] for a third-order approximation. This is equivalent to the spatial-domain formulation that requires an exact reconstruction of quadratic polynomials as described in Section 5.

8 GPU IMPLEMENTATION

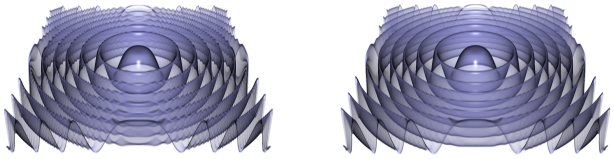
Equation 9 is straightforward to evaluate on the GPU using the following GLSL code:

```
vec4 resampleGradientAndDensity(vec3 position)
{
    float sample = texture(samplerUnit, position).r;

    vec3 step = 1.0 / size;
    vec3 sample0, sample1;

    sample0.x = texture(samplerUnit,
        vec3(position.x - step.x, position.y, position.z)).r;
    sample1.x = texture(samplerUnit,
        vec3(position.x + step.x, position.y, position.z)).r;
    sample0.y = texture(samplerUnit,
        vec3(position.x, position.y - step.y, position.z)).r;
    sample1.y = texture(samplerUnit,
        vec3(position.x, position.y + step.y, position.z)).r;
    sample0.z = texture(samplerUnit,
        vec3(position.x, position.y, position.z - step.z)).r;
    sample1.z = texture(samplerUnit,
        vec3(position.x, position.y, position.z + step.z)).r;
    vec3 scaledPosition = position * size - 0.5;
    vec3 fraction = scaledPosition - floor(scaledPosition);
    vec3 correctionPolynomial = (fraction * (fraction - 1.0)) / 2.0;
    sample += dot((sample0 - sample * 2.0 + sample1),
        correctionPolynomial);

    return vec4(normalize(sample1 - sample0), sample);
}
```

Variable `size` is a uniform `vec3` parameter of the fragment shader that represents the resolution of the input volume data along the three major axes. Note that, without the lines edited in blue, function `resampleGradientAndDensity` implements the standard resampling scheme, where the underlying function is reconstructed by using a trilinear texture sampling, while the gradients are estimated by calculating central differences between trilinear samples. We propose to add the blue lines to the GLSL code in order to improve the quality of the function reconstruction according to Equation 9. This modification does not require new texture samples, so the number of necessary texture fetches still remains seven. Moreover, the additional arithmetic operations are performed in the shadow of the seven trilinear texture fetches. Therefore, our method does not reduce the rendering performance at all, but still improves the numerical accuracy by increasing the approximation order from two to three. Thus, there is no reason left to still use traditional trilinear interpolation combined with on-the-fly central differencing, since using the proposed modification a quality improvement is provided for free.

Figure 6 shows a semitransparent volume rendering of the ML signal using trilinear interpolation and our method. The test signal is represented by 100^3 discrete samples. While there is no difference in terms of rendering efficiency, our method already reconstructs the signal almost perfectly at the given sampling frequency, but the trilinear interpolation still introduces artifacts. The corresponding error images in Figure 7 show a significant improvement in numerical accuracy as well. Figure 8 shows a semitransparent volume rendering of a Christmas tree CT scan using trilinear interpolation and our method. Trilinear interpolation introduces severe staircase

Volume Rendering using On-the-fly Gradient Estimation



Trilinear interpolation. Rendering speed: 11 fps.

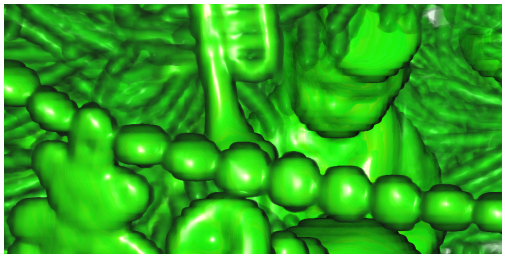
Our method. Rendering speed: 11 fps.

Fig. 6. Semitransparent volume rendering of the ML signal using **on-the-fly gradient estimation**. The frame rates were measured on an nVidia GeForce GT 720M graphics card.

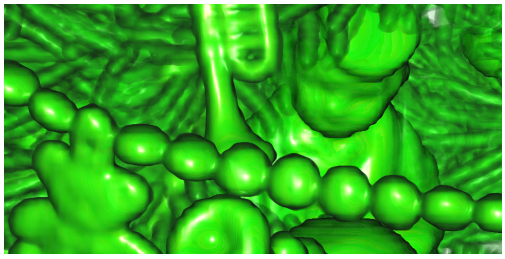
Trilinear interpolation.

Our method.

Fig. 7. Over-emphasized error images corresponding to semitransparent volume rendering using **on-the-fly gradient estimation**. The ground truth is the volume rendering of the analytical test signal using the same transfer function and the same sampling rate along the viewing rays. Error of zero is mapped to white, while darker pixels correspond to larger error.



Trilinear interpolation. Rendering speed: 7 fps.

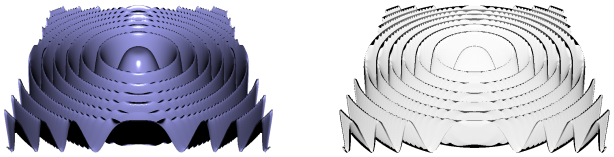


Our method. Rendering speed: 7 fps.

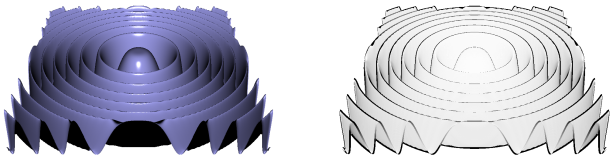
Fig. 8. Semitransparent volume rendering of a Christmas tree CT scan ($512 \times 499 \times 512$ voxels) using **on-the-fly gradient estimation**.

artifacts, which can be significantly reduced by using our method. This is especially apparent at the contours of the objects.

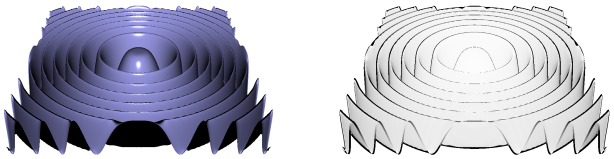
Isosurface Rendering using Precalculated Derivatives



Trilinear interpolation. Rendering speed: 92 fps.



Our method. Rendering speed: 75 fps.



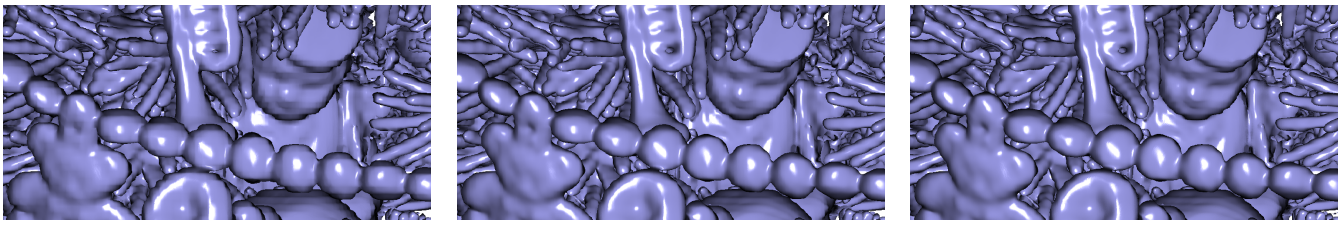
Optimized Catmull-Rom spline interpolation [Csébfalvi 2018]. Rendering speed: 28 fps.

Fig. 9. Left-hand side: Isosurface rendering of the ML signal using **precalculated derivatives**. Right-hand side: The corresponding over-emphasized error images. In each pixel, we measured the error as the distance between the ray/surface intersection points belonging to the approximate isosurface and the analytically defined isosurface. Error of zero is mapped to white, while darker pixels correspond to larger error.

It is important to emphasize that, without using precalculated gradients, an optimized Catmull-Rom spline interpolation [Csébfalvi 2018] cannot be competitive to our method as an on-the-fly central differencing would require 48 additional trilinear texture fetches. An analytic derivative calculation [Sigg and Hadwiger 2005] combined with an arbitrary tricubic filter cannot be competitive either, since it would require 24 additional trilinear texture fetches.

9 ISOSURFACE RENDERING

For isosurface rendering, gradients do not need to be evaluated for each sample position, but only for the first intersection points. Therefore, the bottleneck is the function reconstruction rather than the gradient reconstruction. If there is sufficient texture memory available for storing a four times larger volume representation, our method can be very efficiently adapted to isosurface rendering. We propose to precalculate the three second-order homogeneous partial derivatives rather than the three first-order derivatives. In this case, Equation 9 can be evaluated by taking a single four-component trilinear texture sample, simultaneously interpolating between the function and second-order derivative samples. This can be easily implemented by using the following GLSL code, which assumes that the estimated second-order derivatives are divided by two (to avoid



Trilinear interpolation. Rendering speed: 10 fps.

Our method. Rendering speed: 7.5 fps.

Optimized Catmull-Rom spline interpolation [Csébfalvi 2018]. Rendering speed: 1.5 fps.

Fig. 10. Isosurface rendering of a Christmas tree CT scan ($512 \times 499 \times 512$ voxels) using **precalculated derivatives**.

a division in the shader program) and stored in red, green, and blue channels, while the function values are stored in the alpha channel:

```
float resampleDensity(vec3 position)
{
    vec4 sample = texture(samplerUnit, position);
    vec3 scaledPosition = position * size - 0.5;
    vec3 fraction = scaledPosition - floor(scaledPosition);
    vec4 correctionPolynomial =
        vec4(fraction * (fraction - 1.0), 1.0);
    return dot(sample, correctionPolynomial);
}
```

Here we exploit that s_x, s_y, and s_z in Equation 10 are defined, in fact, by two consecutive convolutions corresponding to the trilinear interpolation, and the discrete Laplacian operator. Due to the commutative property of the convolution, the order of these two convolutions can be reversed. Therefore, first, the second-order derivatives are precalculated at the grid points by using the discrete Laplacian operator, and afterwards s_x, s_y, and s_z are evaluated by separate trilinear interpolations in between. Together with the trilinear interpolation of the function values, this requires a single four-component trilinear texture fetch.

Figure 9 shows an isosurface of the ML signal rendered by trilinear interpolation, our method, and the separable Catmull-Rom spline interpolation. Note that our method is competitive to trilinear interpolation in terms of rendering efficiency, but results in much higher image quality, while compared to the optimized Catmull-Rom spline interpolation, it is 2.5 times faster providing practically the same image quality. Figure 9 shows the corresponding error images.

If isosurfaces contained in higher-resolution real-world data need to be rendered, trilinear texture fetching becomes more expensive as the caching efficiency is lower for larger data sets in general. Therefore, increasing the data resolution, our method becomes more and more efficient than a Catmull-Rom spline interpolation as it requires only one trilinear texture fetch rather than eight trilinear texture fetches. Figure 10 and Figure 11 show isosurfaces of real-world volume data sets that can be rendered already five times faster by using our method rather than an optimized Catmull-Rom spline interpolation.

10 CONCLUSION

We have introduced a novel GPU-based volume-resampling technique that guarantees the same rendering efficiency as trilinear interpolation combined with on-the-fly central differencing, but improves image quality by increasing the approximation order from two to three. Our technique is able to exactly reproduce quadratic polynomials and their derivatives, while the standard trilinear interpolation can exactly reproduce only linear polynomials. As our method provides quality improvement for free, it is recommended to be integrated into every application that used trilinear interpolation combined with on-the-fly central differencing so far. Considering the fact that trilinear interpolation is currently widely used in various visualization and 3D modeling applications, our results can potentially make a significant practical impact.

When our method is used for isosurface rendering, where gradients are necessary to evaluate only for the first-hit surface points, it is slightly slower than a trilinear interpolation if the second-order derivatives are assumed to be precalculated. However, rendering isosurfaces in real-world CT data, our method was measured to be five times faster than the state-of-the-art GPU optimization [Csébfalvi 2018] of the Catmull-Rom spline interpolation.

Theoretically, the interpolation technique proposed in this paper can also be interpreted as a nonseparable extension of the well-known Catmull-Rom spline, which is equivalent to the separable extension in terms of approximation order and frequency-domain behavior. However, our nonseparable extension scheme is more efficient to implement, as it yields a much more compact reconstruction kernel that covers only 32 discrete samples rather than 64 discrete samples.

ACKNOWLEDGMENTS

This work has been supported by projects OTKA K–124124 and VKSZ-14 PET/MRI 7T. The Christmas tree data set is courtesy of the Institute of Computer Graphics and Algorithms, Vienna University of Technology. The carp data set is courtesy of Michael Scheuring, Computer Graphics Group, University of Erlangen.

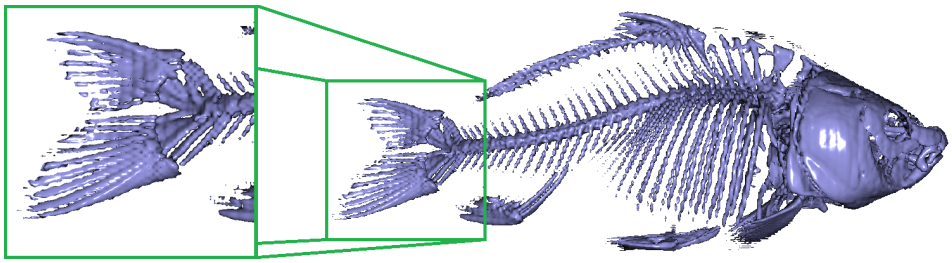
REFERENCES

T. Blu, P. Thévenaz, and M. Unser. 1999. Generalized Interpolation: Higher Quality at no Additional Cost. In *Proceedings of IEEE International Conference on Image Processing*. 667–671.

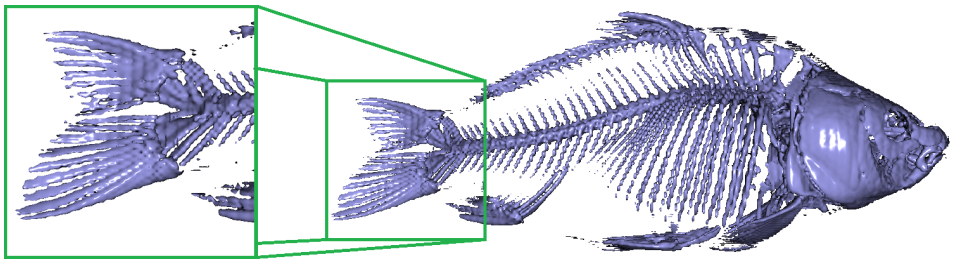
T. Blu and M. Unser. 1999a. Approximation Error for Quasi-Interpolators and (Multi-)Wavelet Expansions. *Applied and Computational Harmonic Analysis* 6, 2 (1999), 219–251.

T. Blu and M. Unser. 1999b. Quantitative Fourier Analysis of Approximation Techniques: Part I - Interpolators and Projectors. *IEEE Transactions on Signal Processing* 47, 10 (1999), 2783–2795.

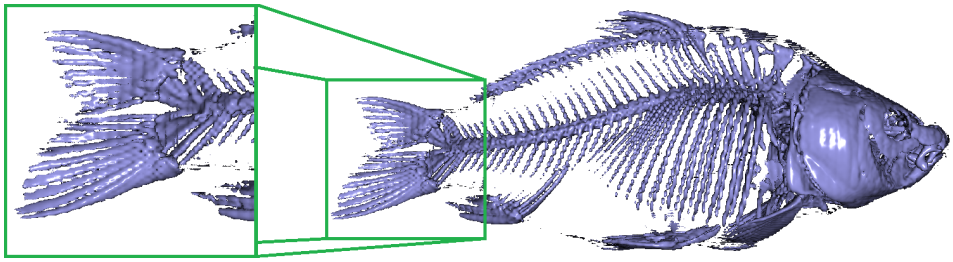
B. Cabral, N. Cam, and J. Foran. 1994. Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware. In *Proceedings of IEEE Symposium on Volume Visualization*. 91–98.



Trilinear interpolation. Rendering speed: 7.5 fps.



Our method. Rendering speed: 5.5 fps.



Optimized Catmull-Rom spline interpolation [Csébfalvi 2018]. Rendering speed: 1.1 fps.

Fig. 11. Isosurface rendering of a carp CT scan ($128 \times 128 \times 256$ voxels) using **precalculated derivatives**. Note that, due to its lower oversmoothing effect, our method can better reconstruct the high-frequency details than the standard trilinear interpolation. In terms of detail preservation, our method is equivalent to the Catmull-Rom spline interpolation, but it is five times faster to evaluate.

E. Catmull and R. Rom. 1974. A Class of Local Interpolating Splines. *Computer Aided Geometric Design* (1974), 317–326.

L. Condat, T. Blu, and M. Unser. 2005. Beyond Interpolation: Optimal Reconstruction by Quasi-Interpolation. In *Proceedings of the IEEE International Conference on Image Processing*. 33–36.

B. Csébfalvi. 2008. An Evaluation of Prefiltered Reconstruction Schemes for Volume Rendering. *IEEE Transactions on Visualization and Computer Graphics* 14, 2 (2008), 289–301.

B. Csébfalvi. 2018. Fast Catmull-Rom Spline Interpolation for High-Quality Texture Sampling. *Computer Graphics Forum (Proceedings of EUROGRAPHICS)* 37, 2 (2018), 455–462.

K. Engel, M. Hadwiger, J. Kniss, C. Reck-Salama, and D. Weiskopf. 2006. *Real-Time Volume Graphics*. AK Peters Ltd.

M. Hadwiger, P. Ljung, C. Reck-Salama, and T. Ropinski. 2009. Advanced Illumination Techniques for GPU-based Volume Raycasting. In *ACM SIGGRAPH 2009 Course Notes*.

M. Hadwiger, C. Sigg, H. Scharsach, K. Bühler, and M. Gross. 2005. Real-Time Ray-Casting and Advanced Shading of Discrete Isosurfaces. *Computer Graphics Forum* 24, 3 (2005), 303–312.

R. G. Keys. 1981. Cubic Convolution Interpolation for Digital Image Processing. *IEEE Transactions on Acoustics, Speech, and Signal Processing* ASSP-29, 6 (1981), 1153–1160.

J. Krüger and R. Westermann. 2003. Acceleration Techniques for GPU-based Volume Rendering. In *Proceedings of IEEE Visualization*. 38–45.

A. Li, K. Mueller, and T. Ernst. 2004. Methods for Efficient, High Quality Volume Resampling in the Frequency Domain. In *Proceedings of IEEE Visualization*. 3–10.

S. Marschner and R. Lobb. 1994. An Evaluation of Reconstruction Filters for Volume Rendering. In *Proceedings of IEEE Visualization*. 100–107.

D. Mitchell and A. Netravali. 1988. Reconstruction Filters in Computer Graphics. In *Proceedings of SIGGRAPH*. 221–228.

T. Möller, R. Machiraju, K. Mueller, and R. Yagel. 1997. Evaluation and Design of Filters Using a Taylor Series Expansion. *IEEE Transactions on Visualization and Computer Graphics* 3, 2 (1997), 184–199.

T. Möller, K. Mueller, Y. Kurzion, R. Machiraju, and R. Yagel. 1998. Design of Accurate and Smooth Filters for Function and Derivative Reconstruction. In *Proceedings of IEEE Symposium on Volume Visualization*. 143–151.

W. K. Pratt. 2001. *Digital Image Processing*. John Wiley & Sons, Inc.

D. Ruijters, B. M. ter Haar Romeny, and P. Suetens. 2008. Efficient GPU-Based Texture Interpolation using Uniform B-Splines. *Journal of Computer Tools* 13, 4 (2008), 61–69.

C. Sigg and M. Hadwiger. 2005. Fast Third-Order Texture Filtering. In *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*. Matt Pharr (ed.), Addison-Wesley, 313–329.

G. Strang and G. Fix. 1971. A Fourier Analysis of the Finite Element Variational Method. In *Constructive Aspects of Functional Analysis*. 796–830.

T. Theußl, H. Hauser, and M. E. Gröller. 2000. Mastering Windows: Improving Reconstruction. In *Proceedings of IEEE Symposium on Volume Visualization*. 101–108.

R. Westermann and T. Ertl. 1998. Efficiently Using Graphics Hardware in Volume Rendering Applications. In *Proceedings of SIGGRAPH*. 169–176.