

Original Research Article

Real-Time Volume Rendering of FCC Datasets Using Box-Splines on the Mobile Platforms

ABSTRACT

Visualizing volumetric datasets using real-time volume rendering technique involves a large number of interpolation operations that are computationally expensive. This situation used to restrict real-time volume rendering methods to be used only on high-end graphics workstations or special-purpose hardware. This paper presents a real-time direct volume rendering (DVR) implementation of face centered cubic (FCC) datasets with box-spline interpolation on mobile devices. The latest version of OpenGL ES (Open Graphics Library for Embedded System) (3.0) is used for implementation to leverage cutting-edge 3D graphics technology, and it shows interactive performance (2.40 frame per second (FPS)) for moderate-sized volume datasets (64×64×64).

Aims To present a real-time direct volume rendering (DVR) implementation of face centered cubic (FCC) datasets with box-spline interpolation on mobile devices.

Study design: Study is based on research conducted in computer lab, University of Seoul, South Korea.

Place and Duration of Study: Computer Science Lab, Department of Computer Science, The graduate College, University of Seoul, South Korea, between June 2014 and April 2015.

Methodology: The latest version of OpenGL ES (Open Graphics Library for Embedded System) (3.0) is used for implementation to leverage cutting-edge 3D graphics technology, and it shows interactive performance (2.40 frame per second (FPS)) for moderate-sized volume datasets (64×64×64).

Results: In our implementation, we calculated the opacity using front-to-back compositing whereby the viewing rays are traversed from the eye point into the volume. We also compared different volume sizes, having the same density.

Conclusion: We have presented a real-time volume rendering technique for FCC datasets on mobile devices that efficiently evaluate spline value. Our work has proven that mobile devices constitute a valid program to achieve interactive volume visualization, despite the fact that the rendering capabilities are concentrated in comparison to desktop solutions, due to their inherent autonomy limitations.

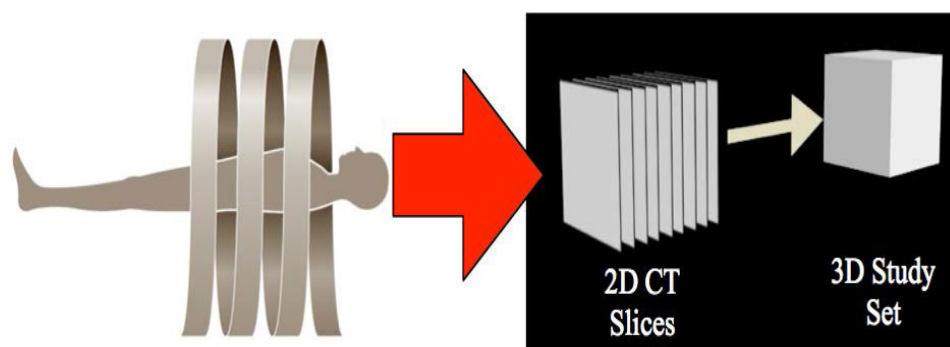
Keywords: [Volume rendering, Ray-casting, FCC lattice, GPU, Box-splines]

1. INTRODUCTION

Volume rendering is a method of visualizing a three dimensional volumetric data as a two dimensional image with a given view point. Such visualization is significant to gain accurate insight into the tremendous quantity of data. But it is impossible to provide this type of data with conventional rendering techniques due to which, the volume rendering has made its own line of business.

Volume rendering can be employed by any industry or field of research involved with 3D datasets. Till now, the largest area of volume rendering research and its usage is performed

21 by the medical industry. Medical imaging was one of the first applications of volume
 22 rendering, and has gone on to be the driving force behind most of the volume rendering
 23 research over the past two decades. The two most normally used medical datasets for
 24 volume rendering are CT (computed tomography) scans and MRI (magnetic resonance
 25 imaging) images. Today's CT scanners and MRI machines typically generate scans of 512 x
 26 512 or 1024 x 1024 pixels. The slices can then be merged into a single 3D representation
 27 and used in volume rendering as shown in Figure 1.



28
 29
 30
 31 **Fig. 1. Illustration of the CT process.**
 32

33 It was in the late 2000s that interactive volume visualization became possible in high range
 34 desktop and laptops by exploiting the texture functionality [1]. Nowadays, laptops are being
 35 replaced with smaller embedded devices like tablets and smartphones. Considering the
 36 current trend towards mobile information systems and ubiquitous graphical devices, native
 37 volume rendering on mobile devices has become an application with various potential uses;
 38 especially last few years have witnessed dramatic improvements that how much
 39 computation and communication power can be jammed into such a little gimmick. However,
 40 despite big improvements, the mobile terminals are still clearly less capable than desktop
 41 computers in many ways. They operate at a lower speed; the display is smaller in size with
 42 lower resolution besides there is less memory for running the program and for storing them,
 43 and you can use the device for a short time because the battery will finally work away. The
 44 interactive 3D rendering on these devices is quite a challenging job, however gradually the
 45 public presentation of several mobile devices has been improving day by day. Recently,
 46 some novel devices are equipped with 3D dedicated graphics chips and a high resolution
 47 color display as well. In addition, many 3D graphics APIs have been developed such as
 48 OpenGL ES, and they are backed by almost every latest mobile device. Due to these
 49 advancements, the mobile devices become a possible platform for volume visualization and
 50 this made it a hot topic for the researches in this fields.

51 This paper is going to look how to achieve real-time volume visualization on mobile devices
 52 that possess fixed imaginations and not viewed as the appropriate platform for visualization.
 53 The present techniques of mobile volume visualization leverages tensor-based volume
 54 reconstruction methods such as trilinear or tri-cubic interpolations, while for our algorithm we
 55 adopted the 6-direction cubic box-spline with the FCC datasets due to its high quality and
 56 reasonable performance.

57 The work is sequenced as follows. Section 2 describes the previous work, section 3
 58 describes briefly background and section 4 will discuss implementation details, it goes on to

59 trace the results we obtained in section 5. Section 6 concludes the paper with some
60 proposition for future employment.

61

62 **2. BACKGROUND**

63

64 GPU-based direct volume rendering has been used for visualization of medical and scientific
65 datasets. Numerous approaches have been proposed in the literature [2], [3]. Kruger et al.
66 [4] presented one of the first GPU (Graphics Processing Unit) implementations. Their
67 approach used proxy geometry, most often resembling the data set bounding box to specify
68 ray parameters either through an analytical approach or by rendering the proxy geometry
69 into a texture. GPU ray casting has been actively investigated but mostly limited to datasets
70 on the Cartesian lattice. Kim [5] implemented a real time GPU iso-surface ray caster for FCC
71 datasets. By GPU preprocessing, they achieved superior performance, efficient evolution of
72 spline value and its gradient and efficient empty space skipping.

73 In respect of mobile devices, interactive direct volume rendering is still a largely unexplored
74 field. On mobile devices, 3D textures are supported through OpenGL ES 3.0 feature.
75 Lamberti et al. [6] and Jeong et al. [7] tried to attempt a rendering server that carries out
76 rendering of the volume and streams the resulting image to the mobile client over a network
77 but unfortunately these server-based solutions required a persistent and fast network
78 connection. Moser and Weiskopf [8] introduced a technique for volume rendering on mobile
79 devices that adopts the 2D texture slicing scheme with a rendering speed of 1.5 frames per
80 second. ImageVis3D [9] is an IOS application for the interactive visualization of very large
81 volumetric datasets, that also uses a 2D textures slicing scheme as well as the GPU ray
82 casting scheme. Then Congote et al [10] implemented a ray-based technique using WebGL,
83 obtaining a frame rate of around 2-3 FPS.

84 Mensmann et al. [11] demonstrated that CUDA™ programming model is suitable for volume
85 ray casting and more efficient than a shader-based implementation. With the introduction of
86 loops in shaders in the Shader Model™ 3, a single-pass approach was pioneered by
87 Stegmaier et al. [12]. Similar to other fragment shader based approaches, first a full screen
88 quad is rendered on screen in order to invoke the fragment shader. Then, the ray casting
89 fragment shader is applied. Using the assigned texture coordinates, the ray directions for
90 sampling of volume are determined. Finally, the volume is traversed front-to-back. Such a
91 single-pass approach shows great potential although more improvement is needed,
92 especially for mobile devices where every additional texture lookup degrades performance
93 considerably.

94 Petkov et al. [13] showed the superiority of the lattice-Boltzmann method on the FCC lattice
95 compared to the Cartesian lattice. Leveraging the isotropic structure of the FCC lattice, Qiu
96 et al. [14] proposed an efficient global illumination method on the FCC lattice by discretizing
97 photon tracing. The six-direction box-spline filter on the FCC lattice was first proposed by
98 Entezari [15] and later investigated in detail by Kim et al. [16].

99 In particular, there is limited work done on real-time volume visualization on mobile devices.
100 One major issue is the technical limitation of mobile devices, which poses challenges for
101 volume rendering visualization methods.

102 **3. DIFFERENT TECHNIQUES**

103

104 In this section we first describe Direct Volume Rendering (DVR) technique, then FCC lattice,
105 and box spline. For a deep insight of volume rendering, refer to the book by Klaus et al [17]

106 and for the perfect theory of box-spline, refer to the book by de Boor et al. [18]. And lastly we
107 will review OpenGL ES 3.0.

108 **3.1 Direct Volume Rendering**

109
110 Volume rendering is frequently used method for the 3D visualization of medical images;
111 based on transparency and coloration of voxel. A medical image is composed of a set of
112 voxels, each voxel having a grey level that represents a physical property of the tissue. The
113 approaches in DVR are based on the law of physics like emission, absorption and scattering.
114 In that respect several different optical models used for light interaction with volume
115 densities of absorbing, glowing, reflecting, and scattering material [19].

116 Before the description of the actual algorithm and implementation, the concepts related to
117 the real-time volume rendering algorithms such as opacity calculation, illumination and
118 shading and composition are briefly discussed as follows.

119 **3.1.1 Opacity Calculation**

120
121
122 Most volume rendering methods calculate the opacity at discrete locations on the rays; as
123 the sample locations on the ray will generally not coincide with the voxel locations, therefore
124 some re-sampling filter will have to be used.

125 Similarly the grey value data has to be converted to an opacity value and the easiest way to
126 calculate this opacity is to assign an opacity to all possible grey values by using a look-up
127 table to directly convert a grey value to an opacity value. This approach can be used when
128 the object of interest has a grey value that is higher (or lower) than its surrounding. The grey
129 values that occur inside the object that is visualized will generally be assigned a high
130 opacity, while the grey values that occur in the surrounding will be assigned a low or zero
131 opacity (fully transparent). Consequently, we will only see the object of interest.

132 Some other methods assign an opacity value to the length of the volume gradient. This
133 approach makes regions with a large volume gradient, such as the transition area between
134 two tissues, opaque or semi-transparent, while regions with a small volume gradient, for
135 instance inside an object, will be made fully transparent. The grey value based and the
136 volume gradient based opacity calculation methods can also be combined, for instance by
137 multiplying the two opacity values resulting from each method.

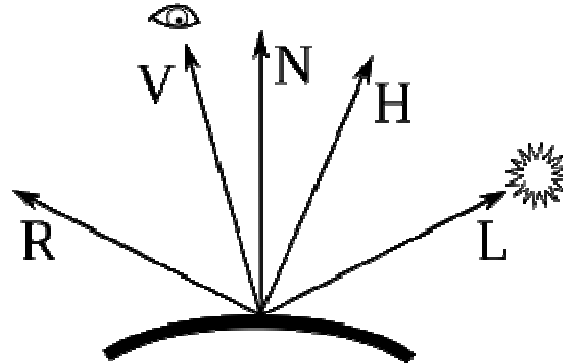
138 **3.1.2 Illumination and Shading**

139
140 Shading model can be used to calculate the effect of light source shining on a surface. The
141 most common shading model used in volume rendering applications is the Blinn- Phong
142 surface shading model which approximates point light source illumination. It consider three
143 different light characteristics - an ambient light that represents the average incoming light
144 from all directions, a diffuse light that represents the isotropically scattered direct lighting,
145 and a specular light that represents the amount of light reflected toward the eye and the
146 sharpness of the highlights. Phong illumination model can be defined as follows

$$147 \quad C_o = C_a K_a O_d + C_p [K_d O_d (\bar{N} \cdot \bar{D}) + K_s O_s (\bar{R} \cdot \bar{V}^n)]$$

148 where C_o is the resulting color, C_a is ambient color, K_a is the ambient reflection
149 coefficient, O_d is the diffuse color of the point on an object, C_p is the color of point light

150 source, K_d is the diffuse reflection coefficient, $\bar{N}$ is the normalized normal vector, $\bar{L}$ is the
 151 normalized light direction, K_s is the specular reflection coefficient, O_s is the specular color, $\bar{R}$
 152 is the normalized reflection vector and $\bar{V}$ is the normalized view point vector. [20]



153

154 **Fig. 2. Blinn-Phong shading model.**

155 The surface normal required by this model is approximated as the normalized gradient of the
 156 scalar field at the sample being shaded. The gradient vector is mathematically the first order
 157 derivative of a discrete volume. The derivative can be defined as:

158
$$\nabla f(x, y, z) = \left[\frac{df}{dx}, \frac{df}{dy}, \frac{df}{dz} \right]$$

159 Volumes are typically shaded as though the light reaches a sample without being attenuated
 160 by the portion of the volume between it and the light. Adding a visibility term, or volumetric
 161 shadows, to this shading model is straight forward. The color of the interpolated, classed,
 162 and surface shaded sample is multiplied by one minus the opacity between it and the light
 163 source.

164 3.1.3 Composition

165

166 When the color of each sample is determined by shading, and the transparency is
 167 determined, the next step is to evaluate the volume rendering equation. This evaluation is
 168 performed by a process called compositing which can be done either back to front or front to
 169 back. The model proposed by Levoy and Drebin et al.[21] uses the under operator to
 170 composite interpolated, classified, and shaded samples taken from the front of the volume
 171 towards the back with reference to the eye. The under operator is defined as:

172
$$C_{out} = C_{old} + (1 - \alpha_{old}) \alpha_{in} C_{in}$$

173

174
$$\alpha_{out} = \alpha_{old} + (1 - \alpha_{old}) \alpha_{in}$$

175 Similarly, if the volume is composed from back to front, with respect to the eye, the over
 176 operator is defined as:

$$C_{out} = C_{in} \alpha_{in} + (1 - \alpha_{in}) C_{old}$$

178

$$\alpha_{out} = \alpha_{in} + (1 - \alpha_{in}) \alpha_{old}$$

180 Front to back compositing is equivalent to back to front, but it has an added advantage.
181 Since the composition is done towards the back and the current transparency is known at all
182 times, the compositing can be stopped early when the translucency is full means nothing
183 behind that point is going to affect the final image. This is one of the more powerful
184 optimizations that can be done for several rendering techniques.

185 In addition to the above compositing schemes, alternative approaches are sometimes used.
186 For example, maximum intensity projection (MIP) and x-ray or weighted sum projections are
187 often applied in medical imaging applications. This compositing scheme is independent of
188 traversal order; i.e. it may be applied in a back to front, a front to back, or any other order.

189 3.2 The FCC Lattice

190

191 A three-dimensional lattice L is determined by all of the integer linear combinations of a non-
192 singular generator matrix G :

$$L : \{Gj : j \in \mathbb{Z}^3, G \in \mathbb{R}^{3 \times 3}, \det G \neq 0\}$$

193

194 The FCC lattice $\mathbb{Z}_{FCC}$ is defined as a subset of the Cartesian lattice $\mathbb{Z}^3$ where the total of its
195 constituents is even, i.e.

$$\mathbb{Z}_{FCC} := \{j \in \mathbb{Z}^3, G \in \mathbb{R}^{3 \times 3}, \det G \neq 0\}$$

196

197 or by generator matrix

$$G_{fcc} := \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

198

199 $\mathbb{Z}_{fcc}$ Can also be defined as follows:

$$\mathbb{Z}_{fcc} := \{(x, y, z) \in \mathbb{Z}^3 : x + y + z \text{ is even}\}$$

200

201 The FCC lattice shows better sampling efficiency than the Cartesian lattice [22]. While the
202 BCC lattice is the optimal 3D sampling Lattice for band-limited and isotropic signals, the FCC
203 lattice is optimal when the signal is sampled at a depressed rate, as indicated by Künsch et
204 al. [23].

205 3.3 Box Splines

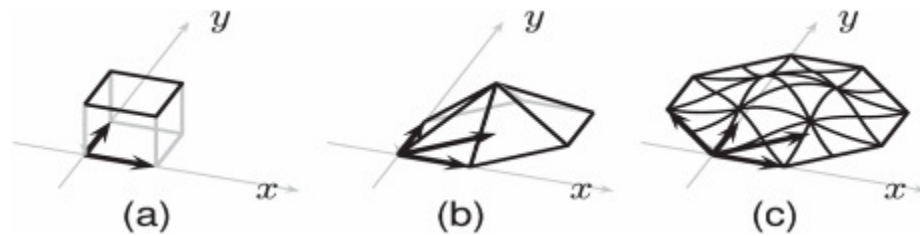
206

207 Box splines were introduced by de Boor at al. [24] as multivariate generalizations of uniform
208 B-splines and have turned out to be remarkably useful.

209 A box-spline is a piecewise polynomial with a finite support and certain continuity and is
 210 uniquely defined by a direction matrix. Given an $n \times m$ (usually $n \leq m$) direction matrix Ξ ,
 211 a box-spline M_Ξ can be constructed by taking consecutive directional convolutions along
 212 each column direction (Figure 3). In other words, starting from the base case ($n = m$)

$$213 \quad M_\Xi(x) := \frac{1}{|\det \Xi|} X_\Xi(X), \quad X \in \mathbb{R}^n$$

214



215

216

217 **Fig. 3. Construction of box-splines with direction matrices.**

218 (a) $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, (b) $\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$ and (c) $\begin{bmatrix} 1 & 0 & 1 & -1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$ via consecutive directional convolutions.

219 Where Ξ is invertible and $X_\Xi(X)$ is the characteristic function on the half-open
 220 parallelepiped $\Xi[0,1]^m$. A box-spline defined by the direction matrix $\Xi \cup \{\xi\}$ can be
 221 recursively defined as

$$222 \quad M_{\Xi \cup \{\xi\}}(x) := \int_0^1 M_\Xi(X - t\xi) dt, \quad \xi \in \mathbb{R}^n.$$

223 Given a discrete dataset on the Cartesian lattice, we can reconstruct a continuous spline $s(x)$
 224 by a convolution of the dataset $V: \mathbb{Z}^n \rightarrow \mathbb{R}$ and a box-spline filter M_Ξ :

$$225 \quad s(x) := V * M_\Xi := \sum_{j \in \mathbb{Z}^n} V(j) M_\Xi(X - j)$$

226 While the theory put forth by de Boor et al. [25] is based on shifts on the Cartesian lattice,
 227 box-splines on non-Cartesian lattices can be easily obtained by change-of-variables [26].

228

229 3.4 OpenGL ES 3.0

230

231 OpenGL ES (Embedded system) is an application programming interface (API) for an
 232 advanced 3D graphics targeted at handheld and embedded devices. Open GL ES (3.0) was
 233 released in 2012 August and is back-ward compatible with OpenGL ES 2.0, while OpenGL
 234 ES 2.0 was successful but significant features that enabled techniques such as shadow
 235 mapping, volume rendering, GPU based particle animation, geometry instancing, texture
 236 compression, and gamma correction were missing. OpenGL ES 3.0 brought these features
 237 to mobile devices while taking care of the constraints of embedded system.

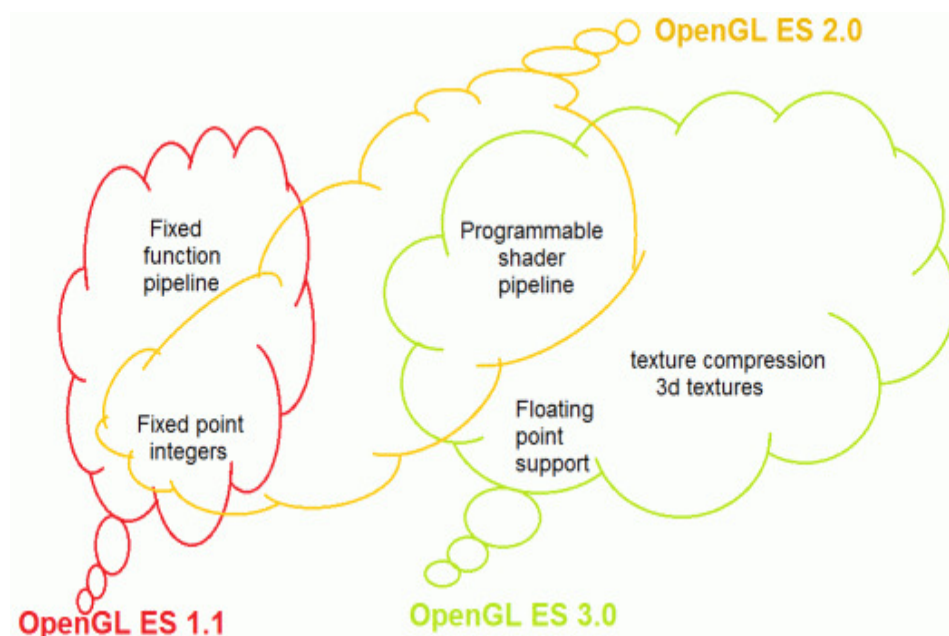


Fig. 4. OpenGL ES.

OpenGL ES 3.0 introduced many new features related to texturing i.e. 3D textures, 2D texture arrays, seamless cube maps and many more. There is likewise a major update in shading language. For brief description, refer to the book by Ginsburg et al. [27].

The EGL (Embedded-system Graphics library) Native Platform Graphics Interface is an additional layer that can be used with OpenGL ES and the underlying windows system. The EGL API implements the features such as power management, support for multiple rendering contexts in a process, sharing objects across rendering contexts in a process, and a mechanism to get function pointers to EGL or OpenGL ES extension functions supported by a given implementation.

Android is an operating system for smartphones, tablets and laptops from the Google-sponsored Open Handset Alliance. OpenGL ES 3.0 API specification is supported by Android 4.3 (API 18) and higher. Latest version of Android 4.4 (KitKat®) was released in October 2013, it doesn't only optimize older phones and tablets but also provided better gaming experience than ever and optimized to run on a greater range of devices as well. Android 4.4 KitKat features additional range of support among other processor types such as NVIDIA Tegra and Samsung's Exynos so that the graphics rendering will be improved dramatically.

Concluding all aforementioned, Android 4.4 KitKat has better 3D graphics rendering, more CPU supports and enhanced Internet connection.

4. IMPLEMENTATION

This part identifies the components of the real-time volume rendering algorithm we adopted for an efficient evaluation for the spline on the FCC lattice, that's been introduced by Kim et al.

al. [5]. They introduced a ray casting algorithm that is fully implemented as a shader program.

266

267 4.1 Implementation Setting

268

269 The implementation of the real-time volume rendering is based on android NDK (Native
270 Development Kit) using C language. OpenGL ES 3.0 and EGL 1.4 are used as graphics
271 API's as provided in Intel's Multimedia Accelerator Software Development Kit. Many older
272 devices do not support OpenGL ES 3.0 so for the test program we chose Google Nexus7
273 tablet with android version 4.4 KitKat.

274 4.2 Evaluation of Spline on the GPU

275 For the evaluation of a spline, it is significant to know the polynomial structure induced by the
276 knot planes of M_{cc} since all the stages in the same polynomial piece share the stencil, which
277 is the relative position of finite data on Z_{fcc} required for valuation. Kim et al. [5] already
278 analyzed the spline structure. For the complete theory of GPU evaluation refer to the Kim et
279 al. [5].

280 There are seven knot planes generated by M_{fcc} , three of which are axis-aligned and
281 decompose the whole space into cubes; $\{j + [0, 1)^3 : j \in \mathbb{Z}^3\}$. Depending on the lower
282 corner j of each cube, we can split the cubes into two groups:

$$283 \quad \{j + [0, 1)^3 : j \in Z_{fcc}\} \quad \text{and} \quad \{j + [0, 1)^3 : j \in (\mathbb{Z}^3 \setminus Z_{fcc})\}.$$

284 Notice that each group can be identified by calculating the parity of j ; $j(1) + j(2) + j(3)$. Then
285 by the remaining four knot planes, cubes in each group are decomposed into five tetrahedral
286 $\{\tau_0, \tau_1, \tau_2, \tau_3, \tau_4\}$ and $\{\tau_5, \tau_6, \tau_7, \tau_8, \tau_9\}$ in Figure 5. Notice that each tetrahedron τ_j in the
287 second group (Figure 5 (f) – (j)) can be transformed to τ_{j-5} with a reflection with regard to
288 the origin followed by a translation by $(1, 1, 1)$:

$$\tau_{j-5} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \tau_j + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \quad \text{for } j \in \{5, 6, 7, 8, 9\}.$$

289

290

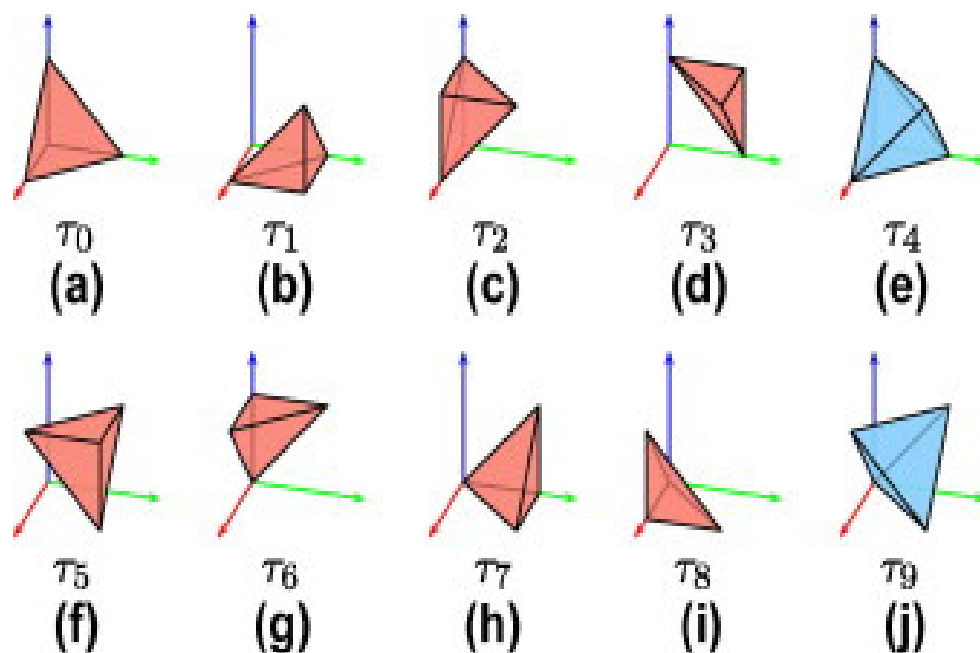


Fig. 5. Shift-invariant tetrahedral polynomial pieces induced by M_{cc} for (top) even and (bottom) odd parity cubes.

4.3 Challenges

During this algorithm, we came across a number of problems because real-time rendering on a mobile device is subject to a number of limitations.

1. In the fragment shader, we came across with a floating point texture problem. We can accept two options either to use `texelFetch()` or `texture()` and both are quite different in functionality. `texture` is the usual texture access function, which handles filtering and normalized $[0,1]$ texture coordinates whereas the `texelFetch()` directly access a texel in the texture without using un-normalized coordinates. In our implementation we were not able to fork out using `texelFetch()` so we decided to use `texture()`.
2. The limited size of graphics memory is another major problem for volume rendering: 16 MB is typically very small for volume datasets. Simply here we cannot render with a 16MB dataset hence the datasets we used is about 32KB~622KB.
3. Computational power is confined by usually low processor speed. Therefore, we cannot expect to run long shaders originally written for desktop PCs on a mobile device at interactive frame rates. We specifically crafted the shaders to avoid complex computations and conditional branches. The mightiness of this real-time volume rendering algorithm was both high speed and high quality, even though our implementation is not optimized for speed.
4. With 3D textures we were unable to employ a floating point texture (GL_R32F, GL_R32UI etc.). Floating point textures have a special designated range of internal

formats (RGBA_16F, RGBA_32F, etc.). Regular textures store fixed-point data, so reading from them gives you [0, 1] range values. Contrary, floating point textures give you floating point numbers as a result. Also not all hardware supports filtering of floating point textures so is the case here, our hardware doesn't support it as well so we decided to use unsigned-byte (GL_R8) and it worked.

5. RESULTS

The main drawback we have faced is the limited texture memory hence our implementation failed to compute the dataset having more than 86x86x86 size of voxels. So all the datasets had to be resampled in order to achieve more meaningful comparison, to serve this purpose we used MATLAB [30].

In our implementation, we calculated the opacity using front-to-back compositing whereby the viewing rays are traversed from the eye point into the volume.

Table 1 shows the performance comparison for the same size datasets. Figure 8 shows the rendering results of linear interpolation and our implementation side by side. For reference images we included desktop generated images using box spline interpolation, shown on the left hand side. The rendering results seen on the right hand side are rendered with linear interpolation and the images in the center are rendered with our box-spline interpolation. It is seen that our results looks more blurry here, it is because we are using a very small dataset with a low resolution, to claim this we carried out our test on desktop with a high resolution dataset. The result we got in desktop is shown in figure 7, where we used the same density datasets. Now it is clearer that with a high resolution dataset FCC box-spline interpolation has better performance than linear interpolation.

In the comparison of rendering with the FCC box-spline interpolation, it is conspicuous that our algorithm has good image quality, even though it employs only half of the dataset but still getting along with linear interpolation. Hence spline evaluation is an expensive operation and have limited support, we get a low frame rate. Further, it should be brought into amount that linear interpolation is hardwired on the graphics hardware according to what, it is much more time efficient. Table 1 shows the performance comparison (fps) of both techniques.

We also compared different volume sizes, having the same density. Table 2 shows the performance comparison. Figure 6 compared the rendering results, it can be seen easily that linear interpolation has more jiggling whereas our method has more smooth and clear results. Conclusively our algorithm showed better quality due to the diminished number of data fetches, and thus the lower complexity of the spline pattern.

Table 1. Performance (in fps: frames per second) comparison

Dataset	Size	Linear interpolation (fps)	FCC box-spline interpolation (fps)
Daisy	64x60x56	50.40	2.40

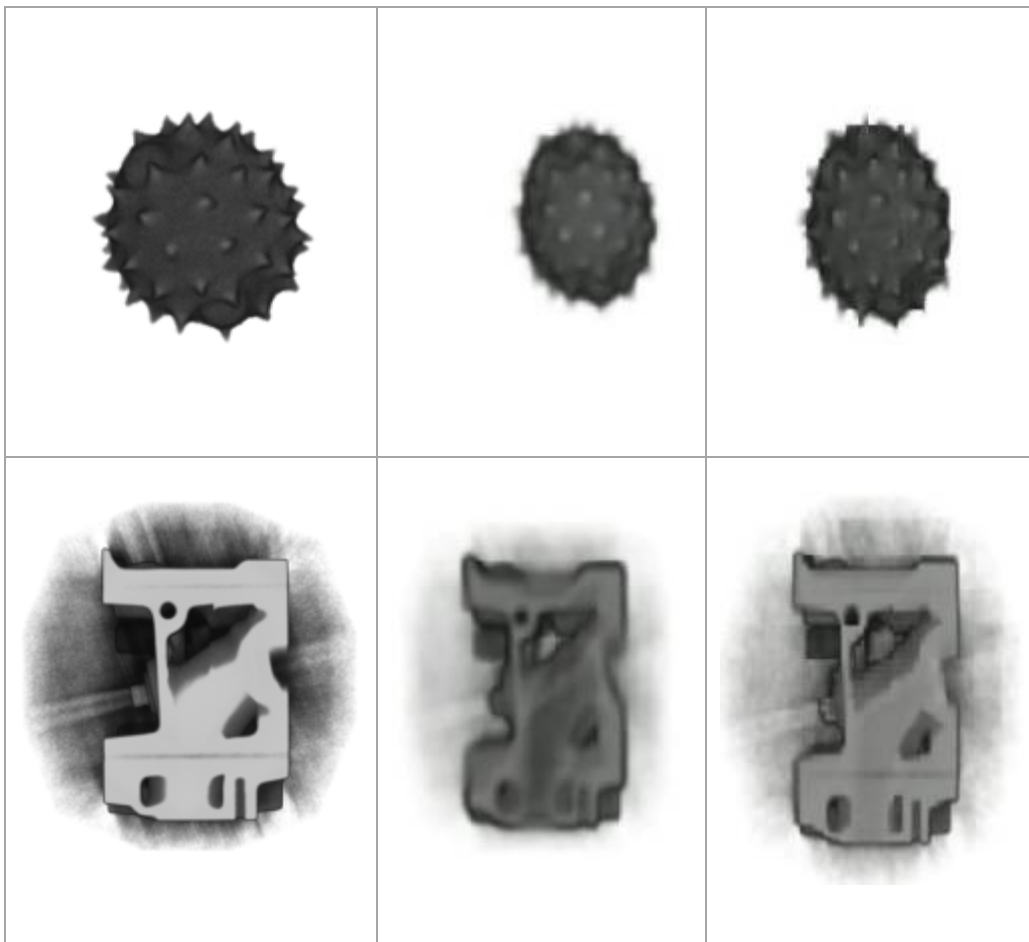
Engine	64×64×64	46.79	2.40
Foot	64×64×64	48.40	2.40

353
354
355
356

Table 2. Performance comparison (same density of different volume sizes)

Dataset	Size Linear interpolation	Speed Linear interpolation (fps)	Size FCC box- spline interpolation	Speed FCC box-spline interpolation (fps)
Engine	65×65×65	45.20	82×82×82	2.40
Daisy	49×46×43	64.80	62×58×54	2.80
Foot	65×65×65	47.59	82×82×82	2.40

357
358





359

360

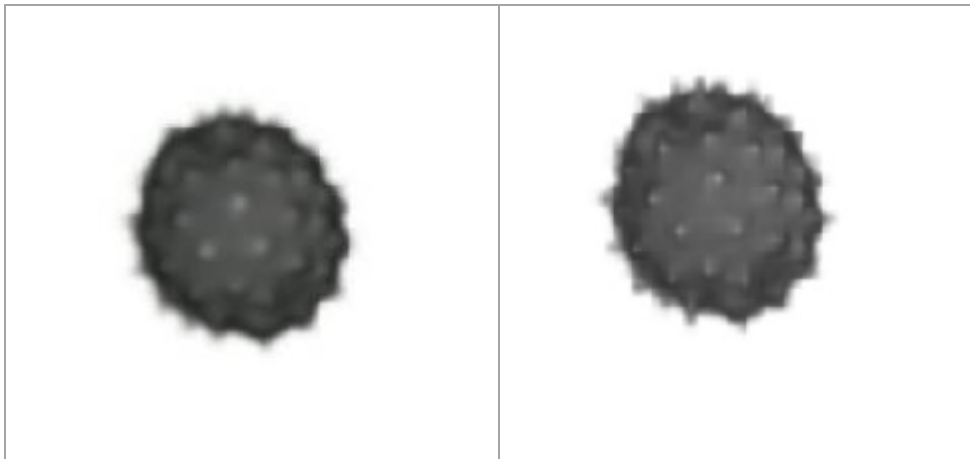
361

362

363

364

Fig. 6. Rendering images: Desktop generated on the left side, FCC datasets with box-spline interpolation in the center and linear interpolation on the right side



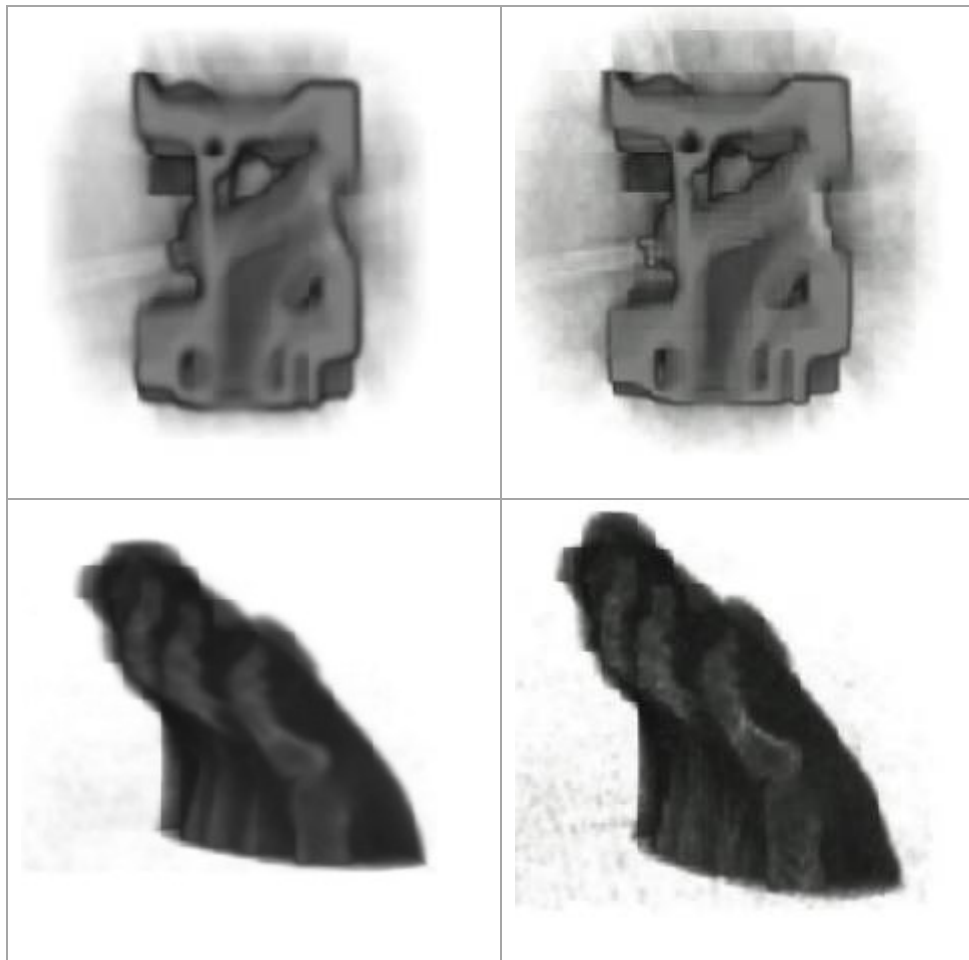


Fig. 7. Rendering images: performance comparisons for table 2.FCC datasets with box-spline interpolation on the left side and linear interpolation on the right side

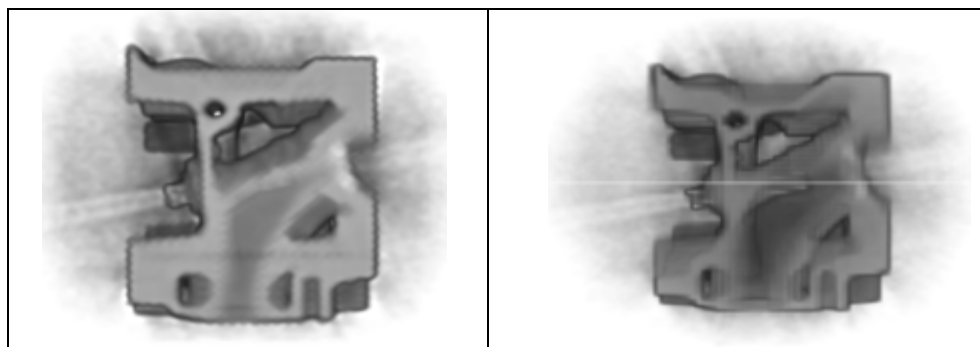


Fig. 8. Desktop generated images: Linear interpolation and Box-spline interpolation.

6. CONCLUSION

We have presented a real-time volume rendering technique for FCC datasets on mobile devices that efficiently evaluate spline value. Our work has proven that mobile devices constitute a valid program to achieve interactive volume visualization, despite the fact that the rendering capabilities are concentrated in comparison to desktop solutions, due to their inherent autonomy limitations. This work is a clear contribution to the literature. However, there comes a challenge for speed and storage, though it paves a new research topic for future.

As future work, our current research is centered on the betterment of the rendering performance and quality based on a continuous search of new techniques as well suited to this kind of devices.

REFERENCES

- [1] Engel, K., Kraus, M., Ertl, T., "High-quality pre-integrated volume rendering using Hardware-accelerated pixel shading", Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics hardware. HWWS '01, ACM Vol. 9, issue 16, (2001)
- [2] K. Engel, M. Hadwiger, J. M. Kniss, A. Lefohn, C. R. Salama and D. Weiskopf, "Real-time volume graphics", A.K. Peters Publisher, 2005.
- [3] B. Preim and D. Bartz, "Visualization in Medicine", Elsevier Inc. Publisher, 2007
- [4] Krüger, J. and Westermann, R., "Acceleration techniques for GPU-based volume rendering", Proceedings of IEEE Visualization, pp. 287–292, (2003)
- [5] M. Kim, "GPU isosurface raycasting of FCC datasets", Graphical Models, Vol. 75, no. 2, pp. 90–101, Mar. 2013.
- [6] Lamberti, F. and Sanna, A., "A solution for displaying medical data models on mobile devices", In SEPADS'05, pp. 1–7, Stevens Point, Wisconsin, USA. (WSEAS) (2005).
- [7] Jeong, S. and Kaufman, A. E., "Interactive wireless virtual colonoscopy" The Visual Computer, Vol. 23, issue 8, pp. 545–557, (2007).
- [8] Moser, M. and Weiskopf, D., "Interactive Volume Rendering on Mobile Devices", In Workshop on Vision, Modelling, and Visualization VMV '08, pp. 217–226, (2008).
- [9] ImageVis3D (2011). Imagevis3d: A real-time volume rendering tool for large data. Scientific computing and imaging institute (sci).
- [10] Congote, J., Segura, A., Kabongo, L., Moreno, A., Posada, J., and Ruiz, O., "Interactive visualization of volumetric data with webgl in real-time", Proceedings of the Web3D '11, pp. 137–146. ACM, (2011)
- [11] Jörg Mensmann, Timo Ropinski, Klaus H. Hinrichs., "An Advanced Volume Raycasting Technique using GPU Stream Processing", GRAPP: International Conference on Computer Graphics Theory and Applications, pp. 190-198 – 2010

- 413 [12] S. Stegmaier, M. Strengert, T. Klein and T. Ertl, "A simple and flexible volume
414 rendering framework for graphics-hardware-based raycasting", Proceedings of The Fourth
415 International Workshop on Volume Graphics, pp. 187 – 241, 2005
- 416 [13] Kaloian Petkov, Feng Qiu, Zhe Fan, Arie E. Kaufman, Klaus Mueller, "Efficient LBM
417 visual simulation on face-centered cubic lattices", IEEE Transactions on Visualization and
418 Computer Graphics 15, pp. 802–814. ISSN 1077-2626, (2009)
- 419 [14] Feng Qiu, Fang Xu, Zhe Fan, Neophytou Neophytos, Arie Kaufman, Klaus Mueller,
420 "Lattice-based volumetric global illumination", IEEE Transactions on Visualization and
421 Computer Graphics 13, Vol. 6, pp. 1576–1583, 2007.
- 422 [15] Alireza Entezari, Optimal, "Sampling Lattices and Trivariate Box Splines", PhD
423 Thesis, Simon Fraser University, 2007
- 424 [16] Minho Kim, Alireza Entezari, Jörg Peters, "Box spline reconstruction on the face-
425 centered cubic lattice", IEEE Transactions on Visualization and Computer Graphics 14, Vol.
426 6, pp. 1523–1530, 2008
- 427 [17] Klaus Engel, Markus Hadwiger, Joe M.Kniss, Christ of Rezk-Salama, Daniel
428 Weiskopf, Real-Time Volume Graphics.
- 429 [18] Carl de Boor, Klaus Höllig, Sherman Riemenschneider, Box Splines, Springer
430 Verlag New York, Inc., 1993.
- 431 [19] Nelson Max., "Optical Models for Direct Volume Rendering", IEEE transection on
432 Visualization and Computer Graphics, pp. 1077-2626, (1995).
- 433 [20] Barthold Lichtenbelt, Randy Crane, and Shaz Naqvi, "Introduction to Volume
434 Rendering", Prentice-Hall, 1998
- 435 [21] Levoy, M., "Display of surfaces from volume data", IEEE Computer Graphics &
436 Applications 8, Vol. 5, pp. 29-37, (1988)
- 437 [22] John Horton Conway, Neil J.A. Sloane, Sphere Packings, Lattices and Groups, third
438 ed., Springer-Verlag New York, Inc., New York, NY, USA, 1998
- 439 [23] Hans R. Künsch, Erik Agrell, Fred A. Hamprecht, "Optimal lattices for sampling",
440 IEEE Transactions on Information Theory 51 Vol. (2), 634–647, (2005)
- 441 [24] C. de Boor, K.A. Hollog, S. Riemenschneider, "Box Splines", Applied Mathematical
442 Sciences, Springer-Verlag, pp. 159–174, 1998
- 443 [25] Carl de Boor, Klaus Höllig, Sherman Riemenschneider, Box Splines, Springer
444 Verlag New York, Inc., 1993.
- 445 [26] Minho Kim, Jörg Peters., "Symmetric box-splines on root lattices", Journal of
446 Computational and Applied Mathematics, 235 Vol. (14), pp. 3972–3989, (2011)
- 447 [27] Dan Ginsburg, Budirijanto Purnomo. Open GL ES 3.0 Programming Guide. Second
448 Edition.

449 [28] Christian John Noon., “A Volume Rendering Engine for Desktops, Laptops, Mobile
450 Devices and Immersive Virtual Reality Systems using GPU-Based Volume Recasting”, PhD
451 thesis, Iowa State University,2012

452 [29] S. Roettger, “Volume library (online)”, January 2012.URL <http://www9.informatik.uni-erlangen.de/External/vollib/>.

454 [30] MATLAB, version 8.10.604 (R2013a). The Math Works Inc. Natick, Massachusetts,
455 2013

456