Modeling of Fuzzy Natural Phenomena with Particle System: A General Method for Interactive Visualization

Vladimir Beliaev, Natalia Zaytseva
Applied Mathematics Department,
St. Petersburg State Polytechnical University, St. Petersburg, Russia
{vladimir, nata}@d-inter.ru

Abstract

We present a general approach for rendering fuzzy objects modeled by a system of particles. At present time a lot of different techniques exist for rendering such objects. Often researchers who develop methods for modeling of complex fuzzy objects like smoke and clouds use there own methods for rendering these objects. This separate developing of rendering techniques has led the methods to loss of generality. We propose general approach based on using image-aligned sheet-buffer splatting method for objects visualization taking into account light scattering inside object volume. Such approach gives physically-based general method for visualization of modeled objects and should lie in base of rendering tool, which takes as input the set of colored particles (the color defines optical material properties) and density distribution function which specifies the way of material distribution inside particle's domain. So researchers do not need to develop rendering algorithm for visualization their results at the step of model developing, they can see results immediately after modeling.

Keywords: *Volume Rendering, Particle Systems, Image-Aligned Sheet-Buffer Volume Splatting, Light Scattering Models.*

1. INTRODUCTION

Physically-based methods of modeling so-called volume (fuzzy) effects like smoke, fog and clouds often give result as a set of particles. Every particle specifies some portion of substance, and is defined by *position* in 3D space, has *volume* and *density* value, which defines how much material is distributed in particle's domain. Optical properties of object's material are specified by particle's *color* and *optical model*, which is used for objects lighting. Thus natural object may consist of several materials; particles may have different colors inside one set. Note, that some modeling methods mean particle as a material point which has no volume, but only position. In this case size of particle's domain can be set due to required visual effects.

Volume rendering methods are very close to our needs, as they take sampled scalar field and reconstruct it. Every value of scalar field defines density material in that point of space. For simplicity of understanding we will call scalar value as particle (thus in the context these terms are equal).

Volume rendering (VR) methods were developed for medical needs and as a result some of them use the benefit of objects stativity. Note, that objects which are modeled by particles are dynamic – every frame particles change their position in the space. VR methods are divided into the following groups: cell-projection [14], texture-based [16], ray–casting [17], splatting [1] and shear-warp [15] method. Table 1 gives a brief comparison of these techniques. More detailed discussion you can find in [2].

Table 1: Comparison of general volume rendering algorithms.

	Ray casing	Splatting	Shear-Warp	Cell Projection	Texture -based
Processed scalar values	All	Required	Required	Required	All
Speed	+	++	+++	+	++++
Quality	++++	++++	++	++	++
Irregular grids	Yes	Yes	No	Yes	No
Hardware acceleration	No	Yes	Yes	Yes	Yes

The most suitable method for rendering dynamic objects, presented by set of particles is *splatting*, because it gives the best quality-speed trade-off and it is the only method which considers every particle independently and as a result it can be used for dynamic particles as well as for static ones. Other VR methods process static 3D grids with particles in nodes (density grids). So the only way to use these methods for dynamic set of particles is to build an irregular density grid at every frame and render it. Evidently, they are inefficient in our case.

Splatting is a popular algorithm for direct volume rendering; it was first proposed by Westover [1]. Splatting method reconstructs a continuous density function from the sampled scalar field (set of particles) using 3D reconstruction kernels. Reconstruction kernel is a radially symmetric function that distributes particle material inside particles' volume. For volume rendering, the continuous function is mapped to the screen as a superposition of pre-integrated 3D kernels, which are called 2D *footprint* or *splat*.

In this paper we present unified approach for rendering set of physical particles. Our method is based on image-aligned sheet-buffer splatting, which was presented by Mueller and Crawfis [3] to eliminate popping artifacts (incorrect illumination of several object parts) that appear due to changing of viewer's position (see Section 3 for details). We propose to use image-aligned sheet-buffer splatting for rendering different-colored particles inside one set without color popping artifact (it appears for penetrating particles with different colors). Also we use image-aligned sheet-buffer splatting for modeling the process of light propagation inside object volume from some light source taking into account light scattering. Additionally we propose to use volume textures, which are supported by modern GPUs, to store pre-integrated segments of reconstruction kernel instead of using the set of 2D images. This increases quality because current hardware allows us

to interpolate between segments automatically (see Section 4 for details).

The remainder of this paper is organized as follows: We first discuss related work in Section 2. We then briefly review the theory and main algorithm of splatting (Section 3), introduce color popping term in Section 4 and present image-aligned sheet-buffer splatting as way of eliminating this affect in Section 5. Next, we present our volume texture approach for effective storing images of particles segments in Section 6. In Section 7 we introduce our method of light propagation inside object's volume based on image-aligned sheet-buffer splatting and compare it the one based on classical splatting. Then we present results of rendering two objects. Finally, we conclude our work and outline future research directions Section 9.

2. RELATED WORK

2.1 Splatting

Splatting-based methods of volume rendering were intensively developed during last years.

First splatting was mentioned by Westover [1]. This method uses pre-integrated 2D splat for visualization every particle. Splat is a quad with 2D texture of pre-integrated 3D reconstruction kernel, and this quad is always perpendicular to viewing direction (in computer graphics it is also known as sprite). Correctness of such approach is based on the fact that reconstruction kernel is a radially symmetric function and it looks the same from every viewing position. So its image can be pre-calculated.

The main disadvantage of this method for us is incorrect processing of penetrating particles with different colors. Mueller and Crawfis [3] proposed image-aligned sheet-buffer splatting method to eliminate popping artifacts, which occur in classical splatting when viewer changes his position relative to object. In this method 2D splats are firstly accumulated in sheet images. Then images are composited[1] sequentially back-to-front with frame buffer giving the resulting image. Plane of sheet images is parallel to screen (in classical sheet-buffer method the plane is aligned with the grid face most parallel to the image plane - 3D density grid is often rectilinear due to scan nature of visualized data). We use image-aligned sheet-buffer splatting to solve the problem of visual artifacts, which appear when two dynamic particles with different colors penetrate each other (color popping artifact).

2.2 Lighting model

Max [4] showed that general lighting model for light propagation inside material should include emission, absorption and scattering of incoming light by elementary material particle. All volume rendering methods take into account only emission and absorption of light incoming from object's background. This approach is correct when visualized object is opaque (material is solid).

Emission and absorption is not enough for gaseous fuzzy objects we are talking about – light scattering is necessary. General approach to visualization of particles which simulate physical natural phenomena should take into account light scattering inside volume object. We consider volume objects that have no distinct surface, so we have no need to take into account light reflection from the surface. As a result our lighting model should include such optical effects as: **emission, absorption** and **scattering** from some light source through the object volume. These optical models were described by Max [4] in detail. Whereas emission and absorption are realized by standard interactive volume rendering methods light scattering is more complex process. There are a lot of papers dedicated to light scattering inside object volume [6], [7], but proposed methods are far from being interactive. Different approximations are used to process scattering interactively.

Scattering illumination models simulate the emission and absorption of light by a medium as well as scattering through the medium. *Single scattering* models simulate scattering through the medium in a single direction. This direction is usually the direction leading to the point of view. *Multiple scattering* models are more physically accurate, but must account for scattering in all directions (or a sampling of all directions), and therefore are much more complicated and expensive to evaluate. Approximation which takes into account only single scattering was used by Voss [9] and Nishita et al. [10]. Such approach results in too dark part of cloud which is opposite to light source (Figure 1). The reason is necessity to model multiple scattering for such objects as clouds.

Nishita et al. [11] showed that scattering of first and second order is sufficient. Dobashi et al. [5] developed two-pass method for single scattering. And Harris [8] generalized these ideas and used splatting for real-time modeling of light multiple scattering inside cloud. He used (classical) splatting for both passes: light propagation calculations and rendering particles with new light values.

Nulkar and Mueller [12] proposed to use image-aligned sheet-buffer splatting for generating shadows from volume object. Also they proved that image-aligned sheet-buffer splatting has the quality comparable with ray casting method (see Table 1).

Our idea consists in using image-aligned sheet-buffer splatting to model the process of light scattering from light source through the fuzzy object volume. Our goal is to improve quality and make speed-up of existing algorithm for light propagation through the set of particles.

Figure 1: Taking into account only single scattering of light inside cloud leads to abnormal dark parts on the opposite side from the light source **[4]**.

3. SPLATTING: THEORY AND ALGORITHM

All volume rendering algorithms evaluate the so-called volume rendering integral (VRI). We use notation and description from

[1] Compositing means the same as blending.

[13]. Denote a point in ray space by a column vector of three coordinates $\mathbf{x} = (x_0, x_1, x_2)^T$. Given a center of projection and a projection plane, these three coordinates are interpreted geometrically as follows: The coordinates x_0 and x_1 specify a point on the projection plane. The ray intersecting the center of projection and the point (x_0, x_1) on the projection plane is called a viewing ray. Using the abbreviation $\hat{\mathbf{x}} = (x_0, x_1)^T$, we refer to the viewing ray passing through (x_0, x_1) as $\hat{\mathbf{x}}$. The third coordinate x_2 specifies the Euclidean distance from the center of projection to a point on the viewing ray. The following notations are equal: $\mathbf{x}$, $(\hat{\mathbf{x}}, x_2)^T$, or $(x_0, x_1, x_2)^T$ to denote a point in ray space.

VRI describes the light intensity $I_\lambda(\hat{x})$ at wavelength λ that reaches the center of projection along the ray $\hat{x}$ with length L (Figure 2):

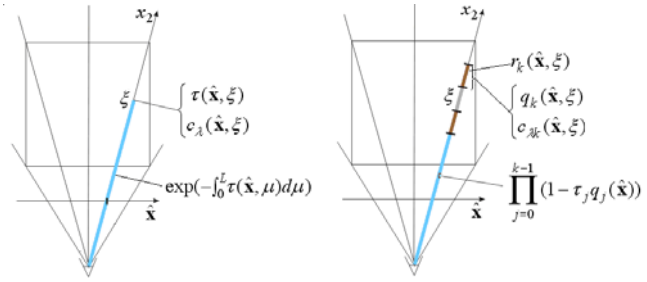

Figure 2: Volume rendering. **Left**: Illustrating the volume rendering equation in 2D. **Right**: Approximations in typical splatting algorithms**.**

VRI looks as follows:

$$I_\lambda(\hat{\mathbf{x}}) = \int_0^L c_\lambda(\hat{\mathbf{x}},\xi)\tau(\hat{\mathbf{x}},\xi)\exp(-\int_0^\xi \tau(\hat{\mathbf{x}},\mu)d\mu)d\xi\,, \quad (1)$$

where $\tau(\mathbf{x})$ is the extinction function that defines the rate of light occlusion at point, and $c_\lambda(\mathbf{x})$ is an emission coefficient, which defines quantity of light emitted by material in the point. VRI in discreet form looks as follows:

$$I_\lambda(\hat{\mathbf{x}}) = \sum_k c_{\lambda k}(\hat{\mathbf{x}})\tau_k q_q(\hat{\mathbf{x}})\prod_{j=0}^{k-1}(1-\tau_j q_j(\hat{\mathbf{x}}))\,, \quad (2)$$

where $q_k(\hat{\mathbf{x}})$ denotes $q_k(\hat{\mathbf{x}}) = r_k(\hat{\mathbf{x}}, x_2)dx_2$, where $r_k(x)$ is reconstruction kernel for particle's material. In theory reconstruction kernel should have infinite definition domain and spreads particle's material portion all over the space. But practically kernel function has finite definition area (local support) – it is truncated at some distance from its centre (when function value became lower than some threshold value). Locality of support is needed for one of simplifications, which reduce VRI (1) to discrete form (2).

More detailed description of VRI and its discreet version you can find in [4].

Note, that VRI takes into account emission and absorption lighting models. It means that every material elementary particle may absorb some portion of incoming light and emit some additional light (e.g. hot gases shine). But VRI doesn't take into account light scattering, which is very important for physically-based visualization of fuzzy objects. For example, the main lighting effect, which makes cloud visible for us, is sun light scattering inside cloud volume. This problem is examined in Section 7.

4. CLASSICAL SPLATTING: COLOR POPPING EFFECT

In classical splatting method all particles are sorted by distance to the viewer and then their splats are projected to the image plane and composited with frame buffer in back-to-front order (relative to viewer).

Consider situation when two particles with different colors are very close (see Figure 3). Splats are blended according to the following equation:

$$C_{new} = C\cdot\alpha + C_{frame-buffer}\cdot(1-\alpha)\,, \quad (3)$$

where C is the particle color, α is the particle's density – material quantity in particle's volume.

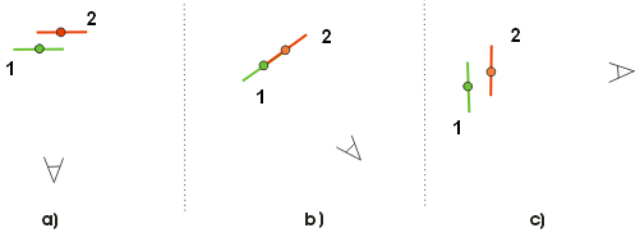

Figure 3: Sprites of two close particles turn due to viewer position (top view). That leads to changing of particle compositing order**.**

On Figure 3 two penetrate particles are shown schematically – only sprites are drawn in top view. Particles are static, but viewer moves. This movement changes order of splats composition with image in frame buffer. In the case of Figure 3a) they are projected and composited in order: **2, 1**. But in case, illustrated on Figure 3c) they are composited in order: **1, 2.** The following equation shows the difference in resulting color in frame buffer after this splats as been blended:

$$\begin{aligned} C_{a)} &= C_1\alpha_1 + C_2\alpha_2(1-\alpha_1) = C_1\alpha_1 + C_2\alpha_2 - C_2\alpha_1\alpha_2 \\ C_{c)} &= C_2\alpha_2 + C_1\alpha_1(1-\alpha_2) = C_2\alpha_2 + C_1\alpha_1 - C_1\alpha_1\alpha_2\,, \\ C_{a)} - C_{c)} &= \alpha_1\alpha_2(C_1 - C_2) \end{aligned} \quad (4)$$

where $C_{a)} - C_{c)}$ is the difference in resulting colors in the image. Figure 3b) shows the case when the distances from two particles to viewer are equal. The resulting order after sorting these particles by distance to viewer may randomly vary from frame to frame. This leads to the effect, which we call *color popping effect*.

5. ELIMINATING COLOR POPPING

To eliminate color popping effect described in previous section we use image-aligned sheet-buffer splatting. This method was developed to avoid popping artifacts, which appears in common sheet-buffer splatting.

In the sheet-buffer method, splats are added within sheets that are aligned parallel to the grid face most parallel to the image plane. After a sheet buffer has been accumulated, it is composited into a cache image that traverses the volume in back-to-front order [3]. Popping artifact occurs when the orientation of the compositing sheets changes suddenly as the image screen becomes more parallel to another volume face.

In image-aligned sheet-buffer splatting sheet images are aligned parallel to the image plane (Figure 4), so popping artifact from the previous example doesn't appear. But from the other side image-aligned slices cut every particle on set of segments (Figure 1a), so instead of storing texture with 2D splat we should store the set of textures with pre-integrated particle segments. Disadvantage of this particle representation is essentially increase of memory usage and speed decrease, but advantage is avoiding of color popping effect.

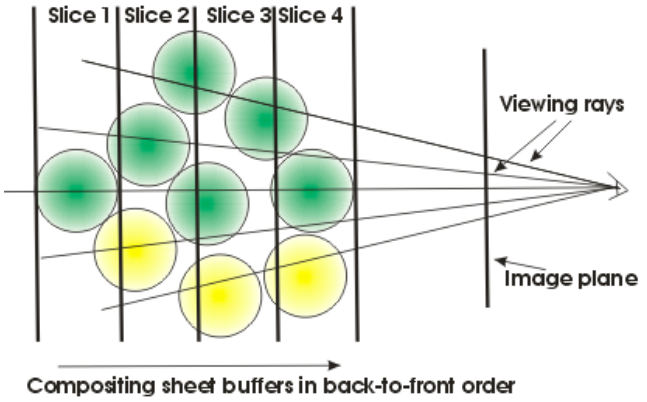

Figure 4: Image-aligned sheet-buffer splatting**.**

Why does it help to avoid our color popping artifact? Solution lies in principles of this method. All particles' segments which lie inside one volume slice are **added (not composited)** into sheet-buffer. Addition is commutative operation and doesn't depend on arguments order.

6. VOLUME TEXTURE FOR IMAGES OF PRE-INTEGRATED SEGMENTS

As it was mentioned in previous section image-aligned sheet-buffer method requires particles representation as a set of segments (Figure 5a), thus every volume slice cuts particles on parts. Mueller and Crawfis [3] store pre-integrated splats of segments in a set of 2D textures (128 segments) and then use the closest texture to the needed one. We propose to use volume texture to store images of pre-integrated segments. Unlike method of Mueler and Crawfis our approach allows to obtain not the closest image but required image of segment via interpolation between existed images. Access to the needed segment's image is done by texture **w**-coordinate. We declare that this method increases resulting image quality.

Additional advantage of using volume texture is solving batching problem, which exists in method of Mueller and Crawfis. Problem consists in high cost of texture setup. So it is more efficient to use one (volume) texture instead of set of 2D textures.

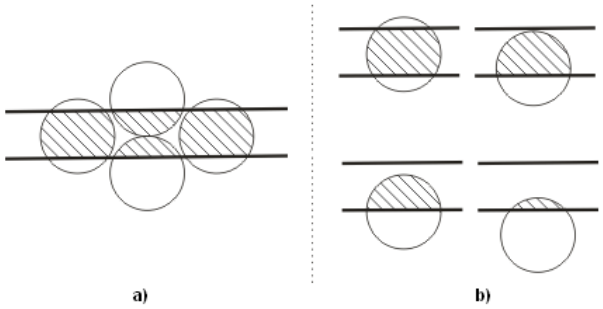

Figure 5: a) every slice cuts particles into set of segments**. b)** we pre-integrate segments and store their images in volume texture**.**

7. LIGHT SCATTERING

We propose to use image-aligned sheet-buffer splatting for modeling of light scattering from light source through the object volume. Note, that Harris [8] used classical splatting for light propagation calculations; Nulkar and Mueller [12] used image-aligned sheet-buffer splatting to render shadow from volume object. We generalize these ideas and declare that for our problem using image-aligned sheet-buffer splatting is more efficient in comparison with using classical splatting. Examine algorithms for light scattering calculations based on: classical splatting (Figure 6-left), image-aligned sheet-buffer splatting (Figure 6-right). Light propagation process looks as light front moving through the object volume. Splatting-based methods model that movement: in classical splatting front moves from particle to particle, in image-aligned sheet-buffer splatting front moves from slice to slice.

The **main idea** of two-pass method is: pre-computing cloud shading in the first pass, and using this shading to render the clouds in the second pass.

Calculation of cloud shading means computing for every particle the portion of light, which reaches it from the light source. Every particle absorbs some portion of incoming light and scatters residual light further. Scattering manner is a material property and is defined by a $PhaseFunction(\omega, \omega')$.

Notation: L is the light direction (from source to particle), ω is the direction from particle to the viewer. Phase function $PhaseFunction(\omega, \omega')$ determines portion of light coming from direction ω' and scattering along direction ω.

We use the same Raleigh phase function as Harris [8]:

$$PhaseFunction(\omega, \omega') = 3/4(1 + x^2), \qquad (5)$$

where $x = \cos(\omega, \omega')$. This phase function gives the most scattering in forward and backward directions, such scattering manner is characteristic for clouds. But we should note, that our algorithm (as well as two-pass methods of Harris and Dobashi et al.) takes into account *only forward multiple scattering*, due the way of shading the cloud in first pass – light front moves away from light source. Backward multiple scattering is very complex process. We include developing of approximation method for modeling backward scattering in our future work directions.

In addition to phase function optical properties of the medium are characterized by albedo. We denote it as $albedo$ in algorithms and in experiments we use $albedo = 0.9$ as it was in [8].

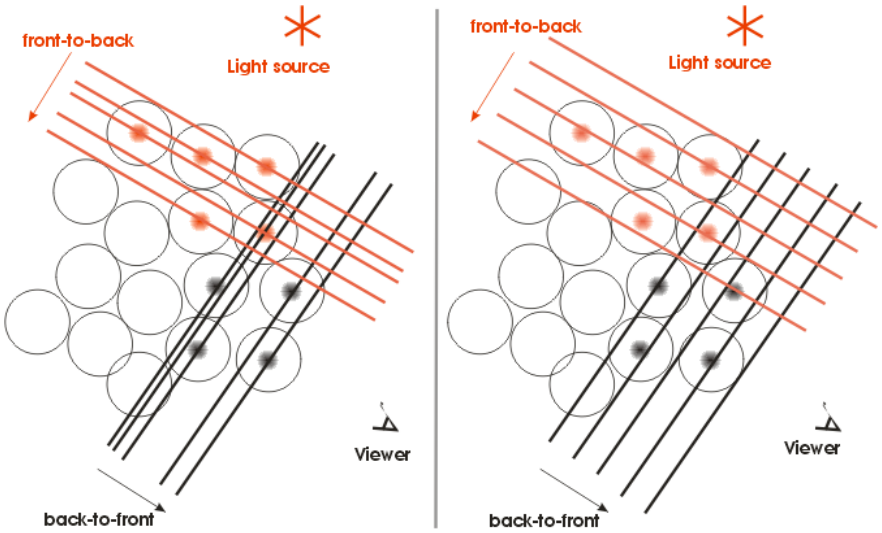

Figure 6: Two-pass rendering of particles set**.** Colored lines illustrate front movement through the volume**. Left:** method based on classical splatting, front moves from particles to particle**. Right:** method based on image-aligned sheet-buffer splatting, front moves from slice to slice**.**

In algorithms presented below extinction coefficient is calculated as $\tau(x) = 1.0 - \alpha$, where α is the particles' density value.

7.1 Classical Splatting

2D footprint of reconstruction kernel in calculated while preprocessing.

Every Frame
I Pass
1. Set camera to light source position
2. Sort particles relative to distance from light source
3. Clear frame buffer with color of light source
4. For every particle $P_k, k = \overline{1,n}$ (**front-to-back** order)
5. Calculate projection center P_k onto the screen
6. Calculate color of the pixel C_{frame_buffer} in frame buffer
7. Store particle color $P_k.C = P_k.C * C_{frame\_buffer} * albedo * P_k.\tau$
8. Light portion, scattered forward $k = PhaseFunction(-L, L)$
9. Render P_k with color $P_k.C * k$ through compositing with frame buffer
10. Endfor

II Pass
1. Set camera to viewer position
2. Sort particles due to distance from viewer
3. For every particle $P_k, k = \overline{1,n}$ (**back-to-front** order)
4. Portion of light, scattered to viewer $k = PhaseFunction(\omega, L)$
5. Render P_k with color $P_k.C * k$ through compositing with frame buffer
6. Endfor

EndFrame

7.2 Image-Aligned Sheet-Buffer Splatting

Preprocessing
1. Calculate 2D images of pre-integrated segments and store them in volume texture
2. For every segment calculate weight Weight[S_i] (portion on particle's material, which is contained in segment volume)

Every Frame
I Pass
1. Set camera to light source position
2. Clear frame buffer with color of light source
3. Build lists of segments for all slices
4. For every slice (**front-to-back** order)
5. For every segment S_i of the slice
6. Build sprite with corresponding segment footprint
7. Calculate projection center S_i onto the screen
8. Calculate color of the pixel C_{frame_buffer} in frame buffer
9. Store particle color $P_k.C = P_k.C * C_{frame\_buffer} * albedo * P_k.\tau$
10. Add segment's color with weight to particle color $P_k.C_{new} += C_{S_i} * Weight[S_i]$
11. Light portion, scattered forward $k = PhaseFunction(-L, L)$
12. Render S_i with color $S_i * k$ through **adding** with image current slice
13. Endfor
14. Composite slice's image with contents of frame buffer
15. Endfor

II Pass
1. Set camera to viewer position

```
2. Build lists of segments for all slices
3. For ever slice (back-to-front order)
4.    For every segment S_i of the slice
5.       Build sprite with corresponding segment
         footprint
6.       Light portion, scattered to the viewer
         k = PhaseFunction(ω, L)
7.       Render S_i with color S_i.P_k.C * k through
         adding to image current slice
8.    Endfor
9.    Composite slice's image with contents of
      frame buffer
10.Endfor
EndFrame
```

Method, which models light scattering process using classical splatting, is slower because we have to read contents of frame buffer for every particle and this operation is very expensive. While method based on image-aligned sheet buffer splatting reads contents of frame buffer for every slice. Number of slices depends on viewer position and particles location (Figure 7).

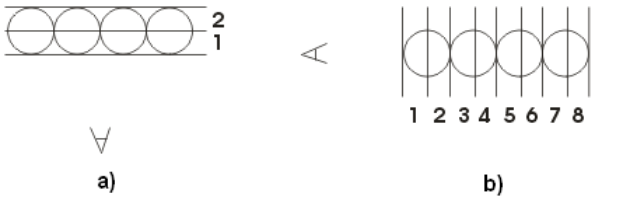

Figure 7: Number of slices in sheet-buffer splatting depends on particles location relative to viewer**. a)** only two slices are needed**. b)** eight slices are required**.**

8. RESULTS

For our experiments we use Intel Pentium 4 platform, 2,4ГГц CPU, 512 M RAM, ATI Radeon 9700 Pro.

All algorithms were implemented using software, because our goal was verification of methods not optimized implementation.

For particles we use Gaussian reconstruction kernel, which was proposed in [3]:

$$G(r) = 0.446 \cdot e^{-2r^2} \qquad (6)$$

where $r = \sqrt{x^2 + y^2 + z^2}$.

Figure 3 demonstrates two penetrating each other particles with different colors (red and green). We use slices of $width = R/2$, where R - radius of particle's sphere (volume), so every particle is divided into the 4 segment. Figure 8 (left, center) shows result of rendering these particles with classical splatting, while Figure 8 (right) demonstrates result of image-aligned sheet-buffer splatting application. We see that application of our method allows us to avoid color popping effect. We should note that result images are not very demonstrative because color popping occurs in dynamic scenes while static images don't give good presentation of motion.

Table 2 contains time, required for rendering these two particles.

Table 2: Result of rendering two particles by different methods**.**

Classical Splatting, ms (fps)	3.11 (321)
Image-Aligned Sheet-Buffer Splatting, ms (fps)	43.10 (23)

Speed is the cost we pay for eliminating visual artifacts, but benefit is the physically-based general approach of our method.

Figure 9 shows results rendering cloud-like objects. It consists of 500 particles and is lit by directional light source. Cloud model is very robust and doesn't' pretend on natural look. It was created to verify shading method (e.g., test halo effect in case when light source and viewer are positioned at the opposite sides from the cloud) and to measure its speed in comparison with shading method proposed by Harris (note that we use our implementation of Harris' method). Table 3 gives time of rendering.

Table 3: Result of rendering cloud with lights scattering modeling.

Lighting by directional light source	**On**	**off**
Classical splatting used for both phases, ms (fps)	3003 (0.3)	2.83 (353)
Image-aligned sheet-buffer splatting for both phases, ms (fps)	286 (3.5)	58.1 (17.2)

9. CONCLUSION AND FUTURE WORK

We have proposed a general approach for rendering volume objects which are modeled with a particle set. Our method includes usage of image-aligned sheet-splatting for visualization of particles with different colors. Also we develop method which models light propagation inside object volume taking into account light scattering, which is very significant lighting effect for visualization of "physical" fuzzy objects. Moreover, our method for scattering is more efficient for light scattering in comparison with the method based on classical splatting because our method is than 10 times faster than the method, presented by Harris. To prevent reader from being confused while comparison our results with Harris' ones, we should make a note. Harris obtained the following results: "shading phase for scenes with only a few thousand particles takes less than a second per light source" – unfortunately these results are given very approximately. We shaded 500 particles for using classical splatting about 3 seconds with implementation that is not optimal. So we think our time results are adequate. We note, that our goal consisted in verification of method based on imaged-aligned sheet-buffer splatting for eliminating color popping artifacts for particles with different colors (e.g., cloud has different-shaded particles that may penetrate leading to color popping) and for shading particles in physically-based manner (image–aligned sheet-buffer method approximates light propagation process better than the classical splatting). We made a justification of using image-aligned sheet-buffer splatting for shading phase instead of using classical splatting. Time results shows that our method is interactive and suitable for interactive visualization.

Algorithms implementation was made without using of GPU possibilities. We plan to optimize these methods using hardware accelerators to relieve CPU for other needs, e. g. simulations. In

fact, most of specified calculations may be processed in vertex shader, e.g., phase function, computation of required segment image. This will allow us to speed up our method in cases when CPU is heavily used for other (not rendering) needs.

Also we plan to compare different reconstruction kernels in respect to image quality and possible visual effects. We want to examine triangles and quads as sprite primitives with a view of rendering speed. Also we are interested in developing method which takes into account not only forward scattering of light inside volume, but also backward scattering.

10. REFERENCES

[1] L. Westover. Footprint evaluation for volume rendering. In *Proceedings of ACM SIGGRAPH 1990*, pages 367–376, August 1990.

[2] Meissner, M., J. Huang, D. Bartz, K. Mueller and R. Crawfis. A Practical Evaluation of Popular Volume Rendering Algorithms. In IEEE/ACM Symposium on Volume Visualization, Salt Lake City, Utah. 2000.

[3] K. Mueller and R. Crawfis. Eliminating popping artifacts in sheet buffer-based splatting. In *Proceedings of the 1998 IEEE Visualization Conference*, pages 239–246, Ottawa, Canada, October 1998.

[4] N. Max. “Optical Models for Direct Volume Rendering”, IEEE Transactions on Visualization and Computer Graphics, vol. 1 no. 2, June 1995.

[5] Y. Dobashi, K. Kaneda, H. Yamashita, T. Okita, and T. Nishita. "A Simple Efficient Method for Realistic Animation of Clouds". SIGGRAPH 2000, pp. 19-28.

[6] Joshua Schpok, Joseph Simons, David S. Ebert, Sharles Hansen. A Real-Time Cloud Modeling, Rendering and Animation System, Eurographics/SIGGRAPH Symposium on Computer Aimation’03. pp.160-166, 2003.

[7] Horng-Shyang Liao, Jung-Hong Chuang, Cheng-Chung Lin, "Efficient rendering of dynamic clouds", Proceedings of the 2004 ACM SIGGRAPH international conference on Virtual Reality continuum and its applications in industry, 2004.

[8] M. J. Harris. “Real-Time Cloud Rendering for Games”. Game Developers Proceedings, 2002.

[9] R. Voss, “Fourier synthesis of gaussian fractals: 1/f noises, landscapes, and flakes”, Tutorial on State of the Art Image Synthesis, ACM Siggraph Course Notes (1983).

[10] T. Nishita, Yasuhiro Miyakawa, and Eihachiro Nakamae, “A Shading Model for Atmospheric Scattering Considering Luminous Intensity Distribution of Light Sources” Computer Graphics Vol. 21 No. 4 (July 1987) pp. 303 – 310.

[11] T. Nisihita, Y. Dobashi, E. Nakamae. “Display of Clouds Taking into Account Multiple Anisotropic Scattering and Sky Light”. SIGGRAPH 1996, pp. 379-386.

[12] M. Nulkar and K. Mueller. Splatting with shadows. International Workshop on Volume Graphics 2001 Stony Brook, June 2001 pp. 35-50.

[13] Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, Markus Gross. EWA Volume Splatting, IEEE Visualization 2001, pp. 29-36, 2001.

[14] M. Weiler, M. Kraus, and T. Ertl. Hardware-Based View-Independent Cell Projection, *Proceedings IEEE Visualization 2002.*

[15] P. Lacroute and M. Levoy, “Fast volume rendering using a shear-warp factorization of the viewing transformation”, *Proc. SIGGRAPH ‘94*, pp. 451- 458, 1994.

[16] B. Cabral, N. Cam, and J. Foran, “Accelerated volume rendering and tomographic reconstruction using texture mapping hardware”, *1994 Symposium on Volume Visualization*, pp. 91-98, 1994.

[17] M. Levoy, “Display of surfaces from volume data”, *IEEE Comp. Graph. & Appl.*, vol. 8, no. 5, pp. 29-37, 1988.

About the authors

Vladimir Beliaev has master’s degree and is an engineer-programmer at St-Petersburg Polytechnic University, Department of Applied Mathematics. His contact email is vladimir@d-inter.ru

Natalia Zaytseva has master’s degree and is an engineer-programmer at St-Petersburg Polytechnical University, Department of Applied Mathematics. Her contact email is nata@d-inter.ru.

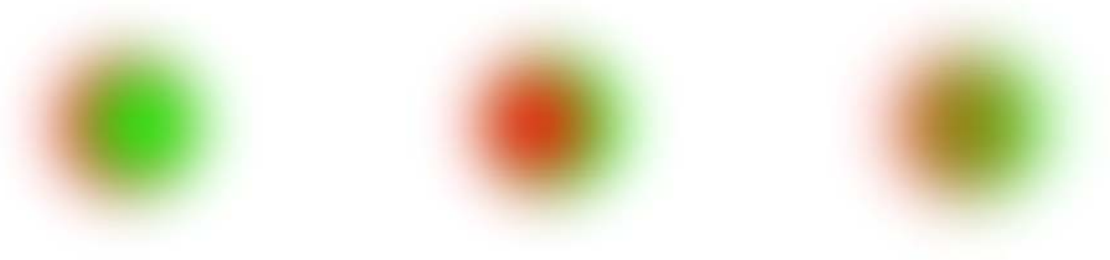

Figure 8: Two particles viewed from two positions, angle between viewing directions differs on $0{,}02^0$. **Left** and **center** images were generated by classical splatting. **Right** image shows result of image-aligned sheet-buffer splatting (images for two viewing directions are accurate within color component value capacity).

Figure 9: Left image shows cloud rendered taking into account light scattering, light source locates behind the cloud – cloud has brighter halo, which is typical for that light source position [8]. **Center** image shows cloud rendered taking into account light scattering, light source is located from the front of the cloud. **Right** image is result of rendering the cloud without light scattering.