Generating and Real-Time Rendering of Clouds

Petr Man*

Czech Technical University in Prague
Czech Republic

Abstract

This paper presents a method for generation and real-time rendering of static clouds. Perlin noise function generates three dimensional map of a cloud. We also present a two-pass rendering algorithm that performs physically based approximation. In the first preprocessed phase it computes multiple forward scattering. In the second phase first order anisotropic scattering at runtime is evaluated.

The generated map is stored as voxels and is unsuitable for the real-time rendering. We introduce a more suitable inner representation of cloud that approximates the original map and contains much less information. The cloud is then represented by a set of metaballs (spheres) with parameters such as center positions, radii and density values.

The main contribution of this paper is to propose a method, that transforms the original cloud map to the inner representation. This method uses the Radial Basis Function (RBF) neural network.

Keywords: Atmospheric Effects Rendering, Clouds, Light Scattering, Metaball Approximation, Perlin Noise, Real-Time Rendering, Volume Rendering, RBF Neural Network

1 Introduction

If we generate images of scenes such as mountain, trees, house or the Earth, clouds play a very important role. An outdoor scene without clouds seems to be unnatural. It is a notorious fact that the shape of clouds is mathematically indefinable. But there are many various approximations. We can divide methods of clouds modeling to physically based and orthogenetic methods.

Physically based methods simulate physical processes [16] that cause creation and disappearing of clouds. Dobashi [20] proposed a computationally inexpensive method, which is based on a simulation using cellular automaton. Miyazaki [13] extended this method to simulate atmospheric fluid dynamics.

Orthogenetic methods do not use physical process. They try to catch a visual meaning of clouds. Voss [17] used fractal noise [3] as a density function. There are many fractal noise methods, the overview is given in [11]. Gardner [4, 5] used a fractal noise approximation to modify translucence of quadric surface.

This paper presents the Perlin noise [12] based on random number generator for generation of 3D cloud maps [7]. The Perlin noise is mapped to an ellipsoid-shaped volume. Afterwards, the clouds in scene are created by unifying these ellipsoid volumes, that was presented by Max [9].

Realistic rendering of clouds is difficult. The light passing through a cloud is physically scattered. Each particle of a cloud absorbs a part of this light and the rest of the light is scattered in non-linear way to other particles. This paper summarizes the method for the real-time rendering presented by Dobashi et al. [20] and extended by Harris [6]. It uses a two-pass rendering real-time algorithm. The first pass is a preprocess and it computes illumination of cloud by multiple forward scattering from light. The second real-time pass computes first order anisotropic scattering of illuminated cloud towards to viewer.

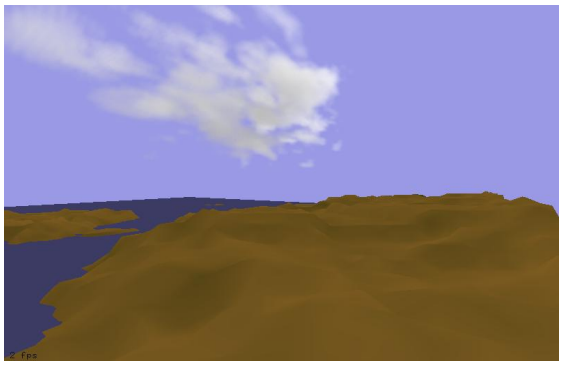

Figure 1: An example of a result of our algorithm for clouds generation.

The direct visualization of the raw 3D map generated by the Perlin noise would be very hard due to huge data count. The problem can be solved by using the metaballs (spheres) with parameters, such as center positions, radii, and density values [1, 19, 20]. We transform original 3D map of a cloud to better representation. It means we are looking for an approximation method, which creates this representation so that a synthesized image of cloud will be similar to the original image. The number of metaballs depends on a level of detail (LOD). This approximation problem is NP-complete.

This paper proposed an approximation method by Radial Basis Function (RBF) [14]. Artificial neural networks

*manp1@fel.cvut.cz

are computational systems inspired by biological neural networks. They consist of highly interconnected processing units called neurons. The artificial neural networks were designed to solve real complex problems. Most of them are NP-complete and may require an approximate solution like multidimensional function approximation, classification and clustering, prediction etc. The alternative method of the cloud approximation was presented by Dobashi et al. [21]. They have used approximation 2D satellite images by precipitation metabals and its subtraction from the map. This method can be extended easily to 3D space.

The paper is organized as follows. Section 2 describes the Perlin noise as a cloud generator. Section 3 propose the RBF neural network as a metaball approximator. Section 4 presents the physically based real-time rendering algorithm. In Section 5 we discuss experimental results, Section 6 concludes the paper and presents future work.

2 Clouds generating by the Perlin noise

The Perlin noise function $\wp$ generates clouds by simply adding up noisy functions h at a range of different scales (frequencies), Figure 2. Each frequence ω can be accentuated or suppressed of its amplitudes φ. There are many parameters for setting, therefore we define a new parameter s (persistence) for replacing of these parameters. The Perlin noise function will be:

$$\wp(x,y,z) = |\sum_{i=0}^{Z-1} \varphi h(\omega x, \omega y, \omega z)|,\ \varphi = s^i,\ \omega = 2^i, \quad (1)$$

where Z is the number of frequencies contained in the noise. It is important to choose it carefully, because a very high frequency can not be displayed due to a pixel size. An absolute value is used according to mapping output values from $<-1,1>$ to $<0,1>$. The function h is continuous and based on random numbers. For creating it we need to make interpolation for a noise function f taking integer numbers.

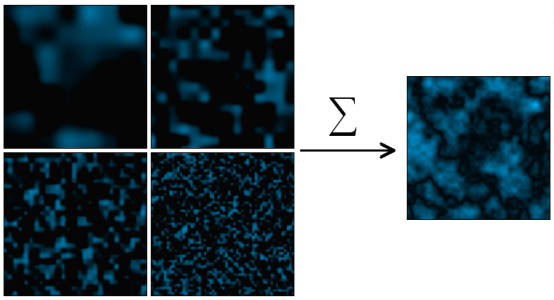

Figure 2: Perlin noise - adding noisy functions at a range of different scales ($s = 0.5$)

The function f is based on a deterministic random number generator g. It takes three integers as parameters in 3D space, and returns a random number based on that parameters. If we pass the same parameters twice, it produces the same random number. We can use the following equation [7]:

$$g(x,y,z) = 1 - \quad (2)$$
$$\frac{((15731n^2 + 789221)n + 1376312589) \wedge 0x7fffffff}{1073741823},$$
$$n = (m \ll 13)^m,$$
$$m = x + 57y + 57^2 z,$$

where $\ll$ means bitwise operator "round left shifting" and $\wedge$ means bitwise operator "and". f takes integer parameters and produces random numbers between -1 and 1.

The function g could be used as noise function, but it contains big changes that are not in real clouds. Therefore it needs smoothing (figure 3). The following discrete noise function produces a weighted average of a neighbourhood:

$$f(x,y,z) = \alpha g(x,y,z) + \beta corners + \gamma sides + \delta dgSides, \quad (3)$$
$$corners = g(x^-,y^-,z^-) + g(x^+,y^-,z^-) + g(x^-,y^+,z^-) +$$
$$g(x^+,y^+,z^-) + g(x^-,y^-,z^+) + g(x^+,y^-,z^+) +$$
$$g(x^-,y^+,z^+) + g(x^+,y^+,z^+),$$
$$sides = g(x^-,y,z) + g(x^+,y,z) + g(x,y^-,z) + g(x,y^+,z) +$$
$$g(x,y,z^-) + g(x,y,z^+),$$
$$dgSides = g(x^-,y,z^-) + g(x^+,y,z^-) + g(x,y^+,z^-) +$$
$$g(x,y^-,z^-) + g(x^-,y,z^+) + g(x^+,y,z^+) +$$
$$g(x,y^+,z^+) + g(x,y^+,z^+) + g(x^-,y^-,z) +$$
$$g(x^+,y^-,z) + g(x^-,y^+,z) + g(x^+,y^+,z),$$

where the upper index x^+ means x is incremented by one and the upper index x^- means x is decremented by one. Parameters α, β, γ and δ are constants driving process of smoothing. They must be choosen with regard to the constraint that the maximum function f equals 1. We experimentally choosed values with the best visual appearance:

$$\alpha = \frac{9}{18},\ \beta = \frac{2}{8 \times 18},\ \gamma = \frac{4}{6 \times 18},\ \delta = \frac{3}{12 \times 18}.$$

The Perlin noise function f needs to take a non-integer values as a parameter, therefore it is needed to make a smooth interpolation between the values. We can use one of these interpolations (figure 3):

$$linear(a,b,x) = a(1-x) + bx \quad (4)$$

$$cosine(a,b,x) = a(1 - 0.5F) + 0.5bF, \quad (5)$$
$$F = (1 - \cos(\pi x))$$

$$cubic(v_0,v_1,v_2,v_3,x) = Px^3 + Qx^2 + Rx + S, \quad (6)$$
$$P = (v_3 - v_2) - (v_0 - v_1),$$
$$Q = v_0 - v_1 - P,$$

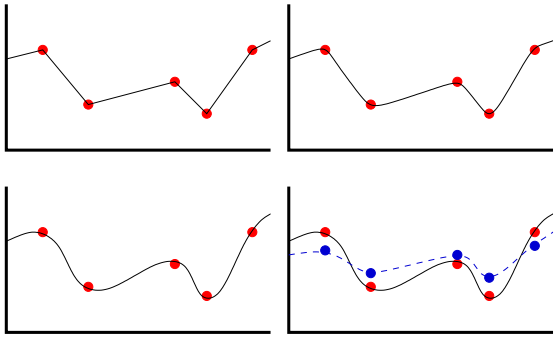

Figure 3: Interpolating functions - linear (up left), cosine (up right), cubic (down left) and comparison of a function and its smoothing (dashed curve) by the function $f(x,y,z)$ (down right)

$$R = v_2 - v_0,\ S = v_1$$

Linear (4) and cosine (5) interpolation functions return a value between a and b based on the value x, which takes value between 0 and 1. Cubic (6) interpolation uses four parameters: v_1 = the point before a, v_2 = the point a, v_3 = the point b, v_4 = the point after b. They are ordered from the fastest one to the best one.

We have choosen the cosine interpolation to achieve compromise between speed and quality. Final noise function h uses a three dimensional interpolation

$$h(x,y,z) = cosine(iz, iZ, z_f), \tag{7}$$

$$iz = cosine(iyz, iYz, y_f), iZ = cosine(iyZ, iYZ, y_f),$$

$$iyz = cosine(xyz, Xyz, x_f), iYz = cosine(xYz, XYz, x_f),$$

$$iyZ = cosine(xyZ, XyZ, x_f), iYZ = cosine(xYZ, XYZ, x_f),$$

$$xyz = f(x_i, y_i, z_i), Xyz = f(x_i^+, y_i, z_i), xYz = f(x_i, y_i^+, z_i),$$

$$XYz = f(x_i^+, y_i^+, z_i), xyZ = f(x_i, y_i, z_i^+), XyZ = f(x_i^+, y_i, z_i^+),$$

$$xYZ = f(x_i, y_i^+, z_i^+), XYZ = f(x_i^+, y_i^+, z_i^+),$$

where the upper index x^+ means x increment by one, the lower index x_i means an integer part of x and the lower index x_f means a decimal fraction of x.

Function $\wp$ can generate clouds density everywhere in 3D space. For displaying one elementary cloud, it is more suitable to use a simply mathematically described volume representation as an ellipsoid. We will map the Perlin noise to the ellipsoid. But then the mapped cloud looks unrealistic because of existing discontinuous changes at the borders. A better solution is to add transparency modulation with respect to position in the ellipsoid to the mapping function. This modulation does not change density at the center, sets it to zero value at the border and makes a nonlinear interpolation between the center and the border. Both, the mapping and the modulation functions are included in the ellipsoid equation.

$$Q(x,y,z) = q_1x^2 + q_2y^2 + q_3z^2 + q_4xy + q_5yz + \tag{8}$$

$$+ q_6xz + q_7x + q_8y + q_9z + q_0$$

where $q_0 - q_9$ are constants determining size, position and rotation of the ellipsoid. The cloud volume is delimited according to the following condition (mapping function):

$$0 \le Q(x,y,z) \le 1 \tag{9}$$

and the cloud density is computed by (transparency modulation):

$$D(x,y,z) = \begin{cases} dQ(x,y,z), & Q(x,y,z) \ge t \\ 0, & Q(x,y,z) < t \end{cases} \tag{10}$$

$$t \in <0,1), d \in (0,1>$$

where the limit parameter t is used for highlighting empty spaces in the cloud and by parameter d we can set the dynamics of the cloud (the difference between maximal and minimal density of cloud). In the figure 4 there are $t = 0.1$ and $d = 1$.

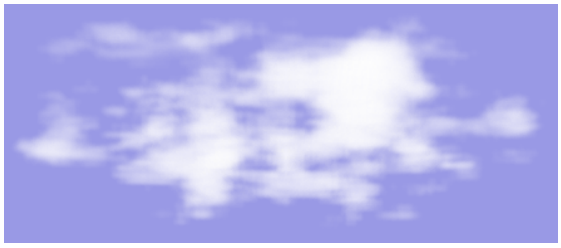

Figure 4: The cloud generated by the Perlin noise function, with $Z = 4$ and $s = 0.5$.

3 Metaball approximation of a cloud by the RBF neural network

We can generate a nice 3D map of cloud by using method described in section 2. But their direct displaying is difficult. More realistic rendering requires complex cloud map. A cloud in figure 4 is made of 11000 voxels. Therefore it is important to choose more suitable inner representation that will be able to reduce complexity and that will approximate the original cloud map. Such an inner representation can be a set of metaballs M [1, 19, 20] with the parameters, such as the center position $\vec{c}$, the radius R and density value ρ.

$$M = \{\vec{m}_i\}, i = 0,...,N,\ \vec{m}_i = (\vec{c}_i, R_i, \rho_i),\ \vec{c}_i = (x_i, y_i, z_i), \tag{11}$$

where N is number of metaballs. This representation is able to simplify voxel burst by one metaball. A density inside a metaball is not constant. It is defined by a basis function f. It defines the density at the center equal ρ and falling nonlinear with increasing distance from the center to zero at the border of the metaball. There are several possibilities to choose the basis function, for example, a Gaussian density function [18] (formula 15). We have choosen

another function proposed by Wyvill et al. [19] to achieve slower growth:

$$f(R_i, r_i) = \begin{cases} \frac{1}{k}(-\frac{4}{9}a^6 + \frac{17}{9}a^4 + \frac{22}{9}a^2 + 1), & r_i \leq R_i \\ 0, & r_i > R_i \end{cases} \tag{12}$$

$$a = r_i/R_i,\ k = \frac{748}{405}\pi R_i$$

where r_i is the distance from the center of the metaball to calculation point P and k is used for the normalization. These functions are depicted in figure 5. An intensity at point P defined by this representation can be computed by:

$$\rho(P) \approx \sum_{i=1}^{N} \rho_i f(R_i, r_i). \tag{13}$$

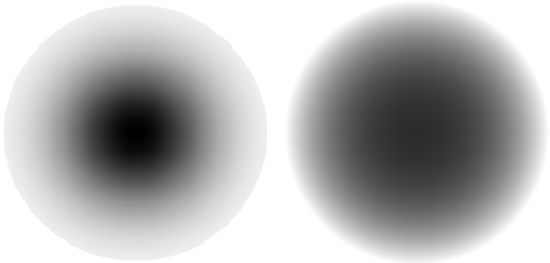

Figure 5: Basis functions - Gaussian, $\sigma = 0.6$ (left) and $f(R_i, r_i)$ (right).

The main problem is to get this representation from the cloud map. This section proposes the RBF neural network as approximator of the 3D cloud map. The set of metaballs M is obtained using the neural network.

A basic artificial neuron model used in artificial neural networks has several inputs and one output. It computes an inner potential φ of inputs that are transformed by transfer function ψ and the result is distributed to other neurons by the output junction. Many artificial neural networks have a layered structure. It means that there exist groups of neurons of a similar function. The artificial neural network benefits of learning process when it is trained to input data. The training process uses an arbitrary training algorithm to modify weight values stored in connections between neurons in the network - so called synapses. Our training set is a set of voxels and the network helps us to approximate the data set and creates a desired representation of the clouds.

The RBF neural network [14] in figure 6 consist of two neuron layers. A hidden layer connected to the network input and an output layer connected to the hidden layer.

The network input (sometimes called input layer) holds and distributes input data to hidden layer. The hidden layer is fully connected with the network input using weighted (w) synapses.

The hidden layer contains N RBF neurons. The RBF neurons are the key to the solution of the problem of reducing amount of voxels in the final result. Each of neuron

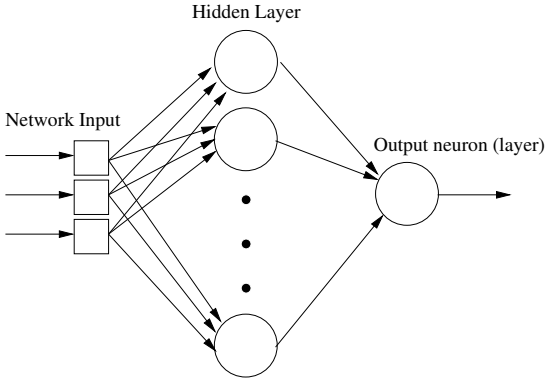

Figure 6: RBF neural network architecture. The RBF neural network has three dimensional input passing to a hidden layer consisting of RBF neurons. The RBF neurons are connected to an output layer consisting of one neuron, which output is the cloud intensity.

represents a required metaball $\vec{m}$. The number of RBF neurons (metaballs) can be chosen manually and it depends on level of detail (LOD) approximation. The k-th RBF neuron has three inputs, that contains weights w_x^k, w_y^k and w_z^k. Let $\vec{w}_k = (w_x^k, w_y^k, w_z^k)$. These weights are the center coordinates of the k-th metaball

$$\vec{c}_k = \vec{w}_k. \tag{14}$$

The RBF neural network uses Euclidean metric for computing the inner potential $\varphi_1 = \| \vec{w} - \vec{x} \|$, where $\vec{x}$ is an input vector sent from the network input (voxel position). The transfer function is Gaussian:

$$\psi_1(\varphi_1, \sigma) : e^{-\frac{\varphi^2}{\sigma^2}} = y^*, \tag{15}$$

where y^* is the output of a neuron and σ is a parameter set at the learning process. It is used for calculation the radius of the k-th metaball

$$R_k = u\sigma_k,\ u > 0, \tag{16}$$

where u is a constant, that drives overlapping of nearby metaballs. In Figure 8 there is the approximated cloud with $u = 2$.

The output layer contains only one output neuron and it is fully connected with the hidden layer. This neuron integrates output of all RBF neurons to a value, which is used to represent intensity of the cloud in our task. Output of the neuron is the output of the whole network.

The neuron has N inputs and one output. Junctions between hidden and output layer contain output weights v_k. The inner potential is

$$\varphi_2 : \sum_{k=1}^{N} v_k y^* = y, \tag{17}$$

where y is output of the neuron. The weights v_k are used for computing intensity of the metaballs similar to the radius:

$$\rho_k = tv_k,\ t > 0, \tag{18}$$

where t is a constant, that drives level of the cloud intensity.

The principle of the radial basis function derives from the theory of functional approximation. We are looking for a function g of the form:

$$g(\vec{x}) = \sum_{k=1}^{N} v_k \psi_1(\varphi_1, \sigma_k), \tag{19}$$

It is an approximation of the K given pairs $\vec{x}_j, y_j$ from the training set and therefore it minimize the following error function:

$$H[g] = \sum_{j=1}^{K} (y_j - g(\vec{x}_j))^2, \tag{20}$$

where K is the number of voxels in the training set.

Now, we know how a RBF network looks like and how we can get the required representation of the cloud. Now, we describe training process, that minimize function H by setting input weights $\vec{w}_k$, output v_k weights and parameters hidden neurons σ.

The learning process consists of three phases. The input weights $\vec{w}_k$ are set in the first phase by the k-means algorithm [14]. The second phase computes the parameters σ of hidden neurons. Algorithms in both phases use only input information like the positions of voxels, therefore the algorithm performs unsupervised clustering. The third phase sets the output weights v_k by the gradient method. It uses all of parameters from training set. It is a supervised method, which means that the training algorithm knows correct output of the trained system - the RBF network. It is the cloud intensity in our case.

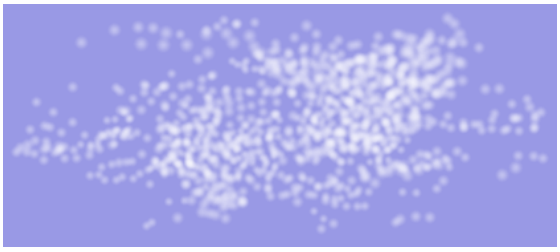

Figure 7: The result of the K-means algorithm, where the number of metaballs K is 1000.

Figure 8: The cloud map approximated by 1000 metaballs.

Figure 9: The cloud map approximated by 100 metaballs.

Figure 10: The cloud map approximated by 30 metaballs.

In the first phase we can choose from various methods for setting weights $\vec{w}_k$ - metaball centers. The simplest method is the direct setting to uniform or random selected patterns from the training set. Chen et al. [15] use method of orthogonal least squares learning for minimizing the network error. Our approach uses self-organizing adaptive K-means clustering [8, 10] (figure 7), which is widely used algorithm for learning RBF neural networks. In the first step of this method we set direct weights $\vec{w}_k$ to random selected patterns from the training set. In the second step we find the nearest center $\vec{w}_k$ for every input data vector $\vec{x}_j$ and we move it according to:

$$\vec{w}_k = \vec{w}_k + \eta(\vec{x}_j - \vec{w}_k),\ j = 1, ..., K, \tag{21}$$

where η is an adaptation speed. We call it *epoch* when one training cycle for all data vectors from the data set $\vec{x}_j$ is performed. The adaptation speed is decreased by every other epoch. The number of epoches is limited by the minimal adaptation speed. We selected a square root method introduced by Darken and Moody [2]:

$$\eta(t) = \frac{1}{\sqrt{n(t)}}, \tag{22}$$

where t is the number of epoch and n is the number of the patterns that have been assigned to the center up to the epoch t.

In the second phase we compute parameters σ of hidden neurons by the root mean square of distances (figures 8, 9 and 10):

$$\sigma_k = \sqrt{\frac{1}{Q} \sum_{q=1}^{Q} \|\vec{w}_k - \vec{x}_q\|^2}, \tag{23}$$

where $\vec{x_q}$ is q-th pattern belonging to a cluster with the center $\vec{w_k}$.

In the third phase we compute the output weights v_k by gradient algorithm. We will repeatedly change the weights v_k for minimizing the error function H by:

$$\Delta v_k = -\lambda \nabla H, \tag{24}$$

where λ is adaptation speed similar to η. End of training the third phase is decreasing a value of the error function H below a specified limit.

4 Real-Time rendering

The algorithm is based on the physical light scattering. The cloud is created using a huge amount of water particles. The light passed though the cloud is scattered by the particles. The part of the light impacted on a particle is absorbed (changed to a heat energy) and the the rest of light is reradiated (scattered) to neighbouring particles. The physical scattering of these particles can be approximated using following Rayleigh phase function:

$$p(\Theta) = \frac{3}{4}(1+\cos^2\Theta), \tag{25}$$

where Θ is an angle between the incident and the scattered light flow. A light passing through the cloud is subsequently partially absorbed by individual particles. But due to multiple scattering the light is scattered many times and therefore the cloud appears to be much more bright then the sky. The amount of the light impacted to the particle is computed by:

$$g(\vec{x},\vec{l}) = \frac{a\tau p(\Theta) I(\vec{x},-\vec{l})}{4\pi}, \tag{26}$$

where $\vec{x}$ is a position of the particle, $\vec{l}$ is a light direction, I is the amount of the light scattered from the direction $-\vec{l}$ to position $\vec{x}$, τ is optical depth (a measure of how much light is absorbed in traveling through a medium) and a is a constant (Albedo [6]), that determines the percentage of attenuation by extinction. The values a=0.9 and $\tau = 8$ are recommended in [6]. The metaballs represent these particles and the rendering algorithm displays them in two passes.

4.1 The first pass

In the first pass the algorithm computes the amount of light sources incident to the every metaball as depicted in figure 11. It uses physical multiple scattering approximation [6] that reflects the light impacted from more directions. The light passing trough the cloud is subsequently subdued by the particles. To compute the light intensity of every metaball it is needed to sort these according to the distance from the light. Sorting is done to scatter the light

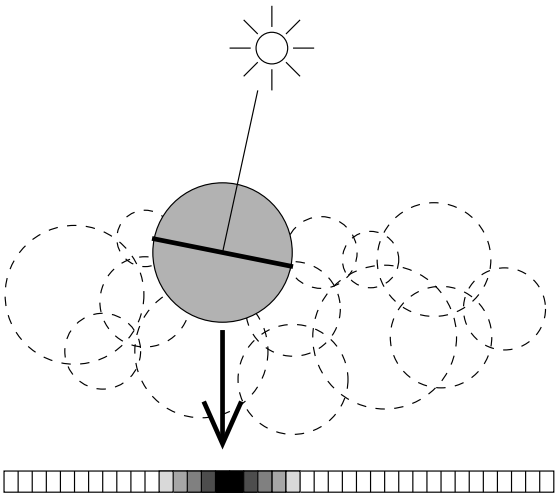

Figure 11: First pass of rendering algorithm

onto next visible metaball only. The algorithm uses hardware acceleration by a blending function and back reading pixels from a framebuffer (a memory stores the rendered picture).

In the first step the framebuffer is filled with white color that represents intensity of the non-absorbed light. The algorithm uses the rendering metaballs to the framebuffer and their back reading for checking the amount of light impacted to the next particle. Therefore, it needs to move the camera to the light position.

In the second step the algorithm computes light intensity for each metaball. It reads amount of the light I from previous metaballs by the back reading of a framebuffer area. Size of the area is defined by a projection of the processed metaball to the framebuffer. The amount of light multiple scattered to the processed metaball equals to average of back read values. The impacted light I is partially absorbed and the rest of the light can be computed by the function $g(\vec{x},\vec{l})$, formula (26). The phase function $p(\Theta)$ can be approximated by value 1.5 that corresponds the Rayleigh function for the angle 180 degrees. The presented can be described by following recurrent formula:

$$I_k = \begin{cases} g_{k-1} + T_{k-1} I_{k-1}, & 2 \le k \le K \\ I_0, & k = 1 \end{cases} \tag{27}$$

$$T_{k-1} = e^{\tau_{k-1}},$$

where I_{k-1} is the amount of the light scattered to the processed metaball that scatters the light I_k, I_0 is the original non-absorbed light represented by white color in the framebuffer, K is the number of metaballs along the light direction $\vec{l}$ and T_{k-1} is the transparency of the metaball.

In the third step the algorithm renders the metaball with the light intensity computed by formula (27) to the framebuffer. The rendering process uses the splatting of the map defined by formula (12) directly to the framebuffer. The metaball defines proportions of the map. It has round

shape, see figure 5. The formula (27) can exploit a hardware implementation of the blending function:

$$c_{result} = f_{src}c_{src} + f_{dest}c_{dest}, \tag{28}$$

where the f_{src} and f_{dest} are blending factors, c_{dest} is an original value defined in the framebuffer, c_{src} is a new value used for modifying the framebuffer and c_{result} is result value written to the framebuffer. We define $c_{result} = I_k$, $f_{src} = 1$, $c_{src} = g_{k-1}$, $f_{dest} = T_{k-1}$ and $c_{dest} = I_{k-1}$.

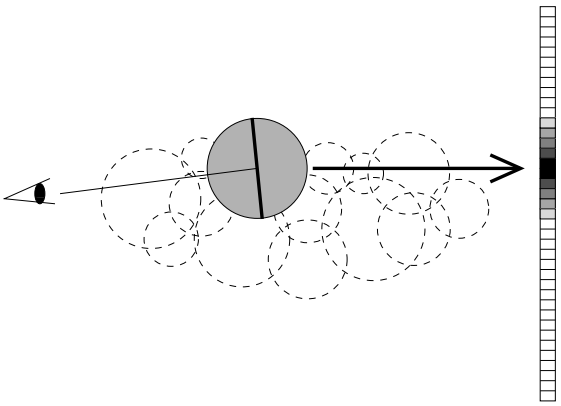

Figure 12: Second pass of rendering algorithm

4.2 The second pass

The second phase of the algorithm is similar (figure 12). The camera is in the original position. The metaballs are sorted according to distance between the camera and the processing metaball positions in descent order, since the metaballs are sources of the light not the camera. The amount of the light excited by the metaball is:

$$I^*_k = g^*_k + T_k I^*_{k-1}, 1 \leq k \leq K^*, \tag{29}$$

where K^* is the number of metaballs along the view direction. Function g^* is computed for the angle between the camera view and the light direction $\vec{l}$. I^*_{k-1} is the light intensity scattered by the previous metaball and computed in the first phase of the algorithm without the phase function. The phase function would be computed twice - first in the light direction and second in the current view direction. This phase is computed in real-time, therefore it uses only first order anisotropic scattering. The amount of the light impacting to the following metaball is not computed by an average value of back read values like in the first phase but it is computed by mixing own light to the previous light impacting to the metaball by the blending function presented in formula (28).

The anisotropic scattering means that a non-constant scatter in various directions defined by function g is used. It gives the clouds their characteristic "silver lining" when viewed looking into the sun. The clouds looks more realistic. Usage of the function g can be compared in figures - anisotropic (figure 14) and constant 1 used - isotropic (figure 13).

Computation of a color light must be solved for every color component separately. If we use more light sources then the result intensity is defined by a sum of their contributions. Faster solutions by means of impostors is presented in [6].

Figure 13: Shading using isotropic scattering.

Figure 14: Shading using anisotropic scattering.

5 Implementation and results

We have implemented three independent programs - a generator, an approximator, and a visualizator. The generator creates a cloud map file using the Perlin noise function. It takes several parameters, such as cloud diameters, the persistence and the number of frequencies. The approximator is an implementation of the RBF neural network. It takes the cloud map file created by generator, approximates by the metaballs and saves the output to a cloud file. The visualizator is 3D engine programmed using the OpenGL and the Glut libraries. It takes the cloud file generated by the approximator and it visualizes the cloud by the presented two-pass rendering algorithm. We have overtake

just the first and the second phases of the approximation algorithm. Therefore the approximated cloud on the figure 8 does not have defined the density values. We tested real-time rendering of a cloud scene. We used the notebook Dell Latitude C840 (CPU 1.6GHz P4, NVidia GeForce 4 video card 64MB, resolution 1024x768). The cloud scene with 1200 metaballs ran at 43fps without using optimizing methods. Other scene with 10000 metaballs ran at 7fps.

6 Conclusion and Future Work

The Perlin noise is an interesting method for generating clouds map. Concering performed experiments we say that it produces visually plausible cloud images. It is suitable for generating clouds - such as cumulus. The RBF neural network reduces complexity of the cloud map preserving its the almost origin shape. The presented methods are suitable for static clouds only. The real-time rendering algorithm provides quick and realistic method of clouds visualization. It can be used even for clouds animation, but there is one basic problem. If the state of metaball position or light source will change, slow first pass have to be recomputed. Final result is shown on figure 1.

Future work will deal with implementation of third phase of approximating method, finding improving methods for the cloud visualization and developing methods for generating the metaball representation directly.

7 Acknowledgements

The author would like to thank to Jan Koutnik for his help with the neural network, Radek Marik for his help with the problem solving, Jiri Zara for his advice, and Bedrich Benes for supervising.

References

[1] G. Wyvill B. Wyvill, C. McPheeters. *Data Structure for Soft Objects*. The Visual Computer, 1986.

[2] Moody J. Darken, C. *Fast adaptive k-means clustering: Some empirical results*. Int. Joint Conf. on Neural Networks, 1990.

[3] Musgrave F. K. Peachey D. Perlin K. Ebert, D. S. and S. Worley. *Texturing and Modeling*. A procedural Approach. Second edition. AP Professional, 1998.

[4] G. Y. Gardner. *Simulation of Natural Scenes Using Textured Quadratic Surfaces*. Computer Graphics, 1984.

[5] G. Y. Gardner. *Visual Simulation of Clouds*. Computer Graphics, 1985.

[6] Mark J. Harris. *Real-Time Cloud Rendering for Games*. Game Developers Conference, 2002.

[7] Jerzy Karczmarczuk. *Functional Approach to Texture Generation.* 2002.

[8] J. B. MacQueen. *Some Methods for classification and Analysis of Multivariate Observations*. Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability, Berkeley, University of California Press, 1967.

[9] N. Max. *Efficient Light Propagation for Multiple Anisotropic Volume Scattering*. Photorealistic Rendering Techniques. Springer-Verlag, 1995.

[10] Mashor Mohd, Yusoff. *Improving the Performance of K-Means Clustering Algorithm to Position the Centres of RBF Network*. 1998.

[11] H-O. Peitgen and D. Saupe. *The Science of Fractal Images*. Springer-Verlag, 1988.

[12] K. Perlin. *An Image Synthesizer*. Computer Graphics, 1985.

[13] Y. Dobashi T. Nishita R. Miyazaki, S. Yoshida. *A Method for Modeling Clouds based on Atmospheric Fluid Dynamics*. Pacific Graphics, 2001.

[14] Haykin S. *Neural Networks, A Comprehensive Foundation*. IEEE Press, 1994.

[15] C. F. N. Cowan S. Chen and P. M. Grant. *Orthogonal least squares learning algorithm for radial basis function networks*. IEEE Transactions on Neural networks, 1991.

[16] Gustav Taxen. *Cloud Modeling for Computer Graphics*. Masters thesis, Royal Institute of Technology, Stockholm, Sweden, 1999.

[17] R. F. Voss. *Fractals in nature: From characterization to simulation*. The Science of Fractal Images. Springer-Verlag, 1988.

[18] L. Westover. *Footprint evaluation for volume rendering*. SIGGRAPH, Black City, 1990.

[19] G. Wyvill and A. Trotman. *Ray-tracing soft objects*. Springer-Verlag New York, Inc., New York, NY, USA, 1990.

[20] H. Yamashita T. Okita Y. Dobashi, K. Kaneda and T. Nishita. *A Simple, Efficient Method for Realistic Animation of Clouds*. SIGGRAPH, 2000.

[21] H. Yamashita T. Okita Y. Dobashi, T. Nishita. *Modeling of Clouds from Satellite Images Using Metaballs*. Proceedings of the 6th Pacific Conference on Computer Graphics and Applications, 1998.