A new method for modeling clouds combining procedural and implicit models

Anselmo Montenegro [1*] Icaro Baptista [1 †] Bruno Dembogurski [2 ‡] Esteban Clua [1 §]

Universidade Federal Fluminense, Departamento de Ciência da Computação, Brazil[1]
Univesidade Federal Rural do Rio de Janeiro, Departamento de Ciência da Computação, Brazil[2]

Figure 1: Examples of clouds computed with the proposed method

ABSTRACT

Cloud models are important components in 3D open environment games and virtual reality applications. In the literature, different approaches have been proposed to solve the problem of modeling the complex geometry and texture of clouds including physical simulation, photo-based modeling and a combination of implicit modeling with procedural techniques. The latter is simple and produces satisfactory results and can be also done in realtime. Nevertheless, it is still difficult to produce the desired results by combining implicit modeling with procedural techniques. The reasons for this limitation are twofold. First, the parameters are not intuitive and intricately coupled with each other. The second reason is that it is hard to combine the overall shape defined by the implicit model with the details generated by the procedural models in a smooth and localized way. In this work, we propose a new way of modeling clouds by combining volumetric implicit modeling with procedural noise techniques. Differently from previous works, the details produced by noise are attached to each implicit primitive independently, in different scales, before they are smoothly blended. This provides a greater level of control in the process of designing and modeling the clouds. By implementing our technique in GPUs it is possible to change the parameters and the blending of primitives in realtime and observe immediately the produced results. This interactive behavior enables the modeler to more easily build and experiment different versions of his ideas. Although we applied this technique to cloud modeling we show that the same approach can be useful for modeling many natural objects as rocks, terrain and planet shapes, among others.

Keywords: Modeling of Clouds, Procedural Modeling, Implicit Modeling, GPU processing.

*e-mail:anselmo@ic.uff.br
†e-mail:decarvalho.icaro@gmail.com
‡e-mail:taunos@gmail.com
§e-mail:esteban@ic.uff.br

1 INTRODUCTION

Modeling natural phenomena can be a very difficult task in Computer Graphics because of the richness of details and complexity of the inherent processes associated to their formation. Among different natural objects, clouds are one of the most important for open environment scenes. According to Harris [11], two of the most common ways used to model clouds are planar-texture-based models and particle systems [19].

Voxelized models are also very common. In such representations, density values can be defined using implicit functions that specify the fallout of the density values from certain primitives, usually represented by a set of particles. Although flexible, pure implicit models fail to produce the fine details and fuzzy aspect usually present in clouds. On the other hand, procedural models [8] are able to produce details generating shapes with fractal [18] and fuzzy appearance. Despite being very powerful, procedural models depend on the choice of parameters that may be not so intuitive and difficult to specify [7]. The control of the overall shape is also limited.

One way to mitigate these drawbacks is to combine the best properties of both models. The work by Lipus et al. [15] proposes the combination of procedural modeling and implicit modeling using a blending scheme based on the Set Theory. In their solution, implicit modeling is responsible for defining the overall appearance of the shape and the procedural modeling specifies its micro-scale details. Although it is possible to produce convincing cloud models using a straightforward approach it is difficult to produce patterns with local variation by just modulating the implicit shape with a global procedural noise function. In real clouds it is possible to notice smoother regions in central areas and more turbulent patterns in the peripheral regions. These features are difficult to reproduce using a global modulation approach.

In this work we propose an extension of the ideas proposed in [15] which explores the blending function proposed by them in a more powerful way. Instead of modulating the whole implicit model via a procedural noise function, in our method, each implicit primitive is modulated by an independent instance of the same noise function with its own input arguments, before it is blended to pro-

duce the final model. We show that the blending functions work quite well on implicit primitives modularized by the Perlin Noise presenting smooth blending results. Besides, we explore the power of GPUs in our implementation. Hence, a user can experiment different set-ups of parameters and primitives in realtime facilitating the generation of a desired model.

This paper is organized as follows: in the next section we describe some of the works related to cloud modeling. In section 3, we describe the basic model that combines implicit modeling and procedural Perlin Noise. In section 4, we describe the proposed method. We present and analyze our results in section 5. Finally, in the last section we describe our conclusions and discuss possible future works.

2 Related Works

Many works have investigated the problem of cloud modeling using many different techniques: 2D texture-based models (*impostors*), physically based models, and volumetric procedural models.

As far as we are concerned, noise functions were one of the first techniques used by the Computer Graphics community for modeling clouds. One of the most known and commonly used techniques is the *Tuburbulence Function* proposed by Ken Perlin [17].

Ebert and colleagues [7] have done an extensive research in modeling different phenomena using procedural noise. They have proposed new ways to model not only clouds but also steam, smoke and other phenomena. His technique is the basis to the one used in this work: generate a sketch shape using implicit functions which are perturbed using noise. Differently though, we apply noise per primitive, permitting a greater expressiveness.

Although powerful, using such techniques to model convincing clouds require a great amount of parameter tweaking which can be quite cumbersome. Hence, many researchers looked for different ways to facilitate the shape control of cloud modeling by using different techniques including cellular automata [5, 23], l-systems [12], implicit modeling [8], radial basis functions [16], fluid dynamics [6] and even optimization-based photograph inspired modeling [3].

As early as 2000, Dobashi et al. [5] proposed a method based on cellular automata for animating clouds. In their work, transition rules are responsible for describing the dynamics of the clouds allowing the simulation of complex motion with low computational cost. They used OpenGL to render their model using the graphics hardware available at that time. Their visualization approach incorporates cloud motion and casts shadows onto the ground.

The paper of Harris et al. [11], although focused in the rendering aspects of the problem, is one of the landmarks in cloud modeling and rendering. They proposed a high-quality, high-speed method for rendering static clouds in games. In their approach, clouds are modeled as impostors, a texture-based approach. They justify their choice by arguing that games are too complex themselves so they opted for a very fast method. Their paper is rich in details about how to properly render clouds in games.

In 2003, Schpok et al. [20] developed a complete system for realtime modeling and animation of clouds. Their system is based on an extension of the previous ideas proposed by Ebert exploring all the capabilities of the new graphics hardware.

Lipuš et al. [15] in 2005 proposed a new way to blend implicit primitives based on the Set Theory. They detected that the simple summation of implicit primitives does not work appropriately when dealing with cloud densities because the density in the intersecting regions can grow in abnormal ways producing undesirable artefacts.

Man et al. [16] presented a new method for generating and rendering static volumetric clouds modeled by Perlin Noise. The authors argue that a volumetric representation is not adequate for realtime rendering. Hence, they proposed a different representation of the cloud that approximates the original map of densities as a set of *Metaballs* [22] whose position, radii and density are determined using a radial basis function neural network. They show results where clouds are described by hundreds of metaballs.

In 2008, Dobashi [6] and colleagues proposed a way to control the parameters in atmospheric fluid dynamics models for generating cummuliform clouds. The method works in two steps: first, the modeler defines the overall shape of the cloud and in the next step the model adjusts the parameters to generate clouds that conform to the predefined shape.

Jiangbin Xu et al. [23] proposed the use of probability fields for controlling a cellular automata method. The probability fields are created using a fractional Brownian motion function (fBm) which describes the motion of particles generating the cloud features.

Dobashi et al in 2012 [4] proposed a work that approaches cloud modeling as an inverse problem: Photographs of real clouds are used to estimate the parameters of a non-uniform density model using an optimization method. The objective function is based on the difference of the color histograms between the synthesized image and the photograph. In order to use his model, multiple scattering is considered inside the cloud for searching the ideal parameters. The authors claim that they can precompute a cloud model in twenty minutes. They enhanced their method in a work presented in 2014 [3].

Recently in 2016, Elhaddad et al. [9] proposed an new method for dealing with the complexity of modeling large scale clouds in realtime as a N-body problem. They proposed the use of Leonard-Jones potential to simplify and simulate the cloud generation process. They used a scheme to minimize the interaction among particles by subdividing the environment space into cells and define cutoff distances to perform calculation between neighboring particles.

3 Basic concepts

In this section we review the basic concepts related to cloud modeling using implicit and procedural techniques. It follows closely the work in [15] and is added here for comprehensiveness.

A cloud can be defined as a volumetric shape $V \in R^3$ where a density value is defined by a function $\rho(p)$ for each $p \in V$. The function $\rho(p)$ is typically defined in such a way that the modeler can manipulate the shape to be constructed with a certain level of control. This can be done in two steps. The first one defines the overall shape of the model using implicit modeling and the second adds details using procedural noise functions.

Implicit models are constructed by combining primitive implicit functions which can be of two kinds: *point primitives* and *skeleton primitives*. In this work, as in [15] we define the density field as a combination of field functions $g(x_i)$ where x_i is a normalized distance. Each field function $g(x_i)$ is defined by a implicit primitive $b(x_i)$ and a weight value w_i. In our implementation, $b(x_i)$ is the Wyvill-1 function [21]:

$$b_i(x_i) = -\frac{4}{9}x_i^6 + \frac{17}{9}x_i^4 - \frac{22}{9}x_i^2 + 1, 0 \leq x_i \leq 1 \qquad (1)$$

The normalized distance is computed as $x_i = d_i/l_i$ where d_i is the distance of a given point $p \in R^3$ to the primitive and l_i is the size of the region of influence (see Figure 2). Inside the skeleton, the value of the field function is equal to w_i. In other words, the value of $b(x_i)$ is considered equal to one. Out of the skeleton and inside the region of influence the value of the density decays according to $b_{(x_i)}$ until it reaches zero in the region out of the influence region.

The final density field is defined as a summation of n field functions $g(x_i)$ as defined below:

$$\rho_n(p) = \sum_{i=1}^{n} g_i(x_i) = \sum_{i=1}^{n} w_i b_i(x_i) \qquad (2)$$

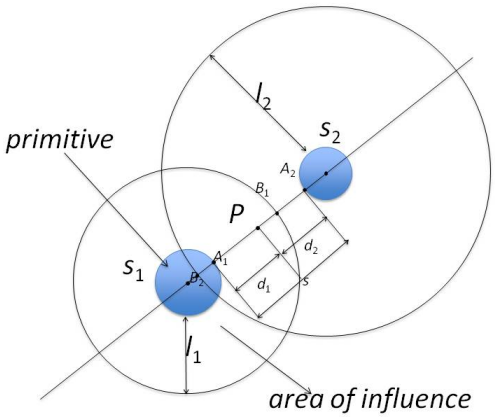


Figure 2: Field function. Picture drawn based on [15].

Lipuš et al. in [15] showed that the summation of field functions does not produce adequate blending results for volumetric data describing density values. Abnormalities appear in the region of intersection of different field functions (see Figure 12 a,b in [15]).

To cope with such problem, they came up with a way to combine implicit density field functions based on the Set Theory. The new blending strategy is defined by a recursive function given by equation 3.

$$\begin{aligned} \rho_1(p) &= g_1(x_i) \\ \rho_i(p) &= \rho_{i-1}(p) + g_i(x_i) - \frac{1}{w}\rho_{i-1}(p)g_i(x_i), \\ i &= 2,\ldots,n, \\ w &= w_1 = w_2 = \cdots = w_n \end{aligned} \tag{3}$$

Intuitively, the formulation given by equation 3 avoids the creation of abnormal density values in the density field by recursively subtracting the spurious duplicated contributions of density in the intersecting regions.

Noise functions are used to introduce fractal behavior into the pure implicit blending model. One of the most used noise functions, and the one also used in this work, is the *Turbulence Function* proposed by Ken Perlin. The turbulence noise function, presented in Equation 4, is a weighted sum of band pass noises. It can be controlled via four parameters: *amplitude* (a), *frequency* (fr), *gain* (ga) and *lacunarity* (lc). Lacunarity and gain define, respectively, the rate of change of the frequency and amplitude of the noise per octave, regulating the fractal behavior of the noise.

$$N(p) = \sum_i a \left| \frac{(lc_i) * fr * p}{ga_i * a} \right| \tag{4}$$

The basic cloud modeling algorithm is a straightforward combination of the implicity density $\rho(p)$ - computed using the Lipuš' blending equation- with the *Turbulence Noise* $N(p)$ modulated by $\rho(p)$. This can be resumed by equation 5 where np is the percentage of turbulence noise to be applied to the intermediary implicit model.

$$\bar{\rho}(p) = (1-np)\cdot\rho(p) + (np)\cdot\rho(p)\cdot N(p) \tag{5}$$

Algorithm 3 shows a pseudocode of the ideas previously discussed. Algorithms 1 and 2 describe, respectively, the implementation details of the implicit blending equation and the noise addition function. Observe that the recursive function in 3 is implemented iteratively in Algorithm 3, as it is a tail recursion.

Besides, in Algorithm 2, the noise function is defined as the absolute value of $N(p) = fBm3D(x,y,z,n,fr,a,lc,g)$, where x,y,z are the coordinates of a given point primitive p, n is the number of octaves and fr, a, la and ga are the turbulence parameters. Taking the modulus of fBm3D is appropriate for generating *cummuliform* clouds [20].

Data: i, j, k, ρ - intial density value, p - primitive position
Result: ρ' - new density
$dist \leftarrow distance(i,j,k,p.x,p.y,p.z)$
if $dist < p.radius$ **then**
  $\rho' \leftarrow 1$
else if $dist < p.influence + p.radius$ **then**
  $dist \leftarrow (dist - p.radius)/p.influence$
  $g \leftarrow Wyvill(dist)$
  $\rho' \leftarrow \rho + g - (1/p.weight)(\rho)g$
end if

Algorithm 1: ComputeImplicitDensity

Data: ρ, p
lc - lacunarity, fr - frequency, a - amplitude, ga - gain, s - scale,
n - numOctaves, np - noise percentage
Result: $\bar{\rho}$
$N(p) \leftarrow max(|fBm3D(p.x,p.y,p.z,n,fr,a,la,ga)|,1)$
$\bar{\rho} \leftarrow (1-np)\rho + \rho(N(p))(np)$

Algorithm 2: AddNoise

Data: P - set of primitives
la - lacunarity, fr - frequency, a - amplitude, g - gain, s - scale,
n - numOctaves, p - noise percentage
width, height, depth - dimensions of the volume array
Result: Density volume V
for $V_{ijk} \in V$ **do**
  $V_{ijk} \leftarrow 0$
  $\rho \leftarrow 0$
  $x,y,z \leftarrow i + rand()*s, j + rand()*s, k + rand()*s$
  for $h = 0$ to $|P|$ **do**
    $\rho \leftarrow \rho + ComputeImplicitDensity(i,j,k,\rho,P_h)$
  end for
  $V_{i,j,k} \leftarrow max(AddNoise(\rho,P_h,n,fr,a,la,g),1)$
end for

Algorithm 3: CloudGenerator

4 Proposed Method

The basic algorithm presented in the previous section produces satisfactory results but has one limitation. The procedural noise modulation is applied onto the implicit shape defined by the blending of the $g(x_i)$ implicit primitives. As it works applying a global modulation to the blended implicit primitives, it is not possible to define different levels of noise for different parts of the model. See for instance a real cloud picture (Figure 3), where noise lacunarity and frequency vary from region to region. Consequently, it demands a lot of parameter tweaking to define a cloud with varying levels of detail along the whole shape.

Figure 3: A picture of a real cloud.

Instead of applying noise onto the initial implicit shape, given by blending the fields defined the Wyvill function, we propose a different scheme where noise with different characteristics is added to each primitive independently. This results in a more effective way to model general procedural shapes because we can control the local noise at each primitive separately and then combine them to obtain the final result. Then, by setting $g'_i(x_i) = N_i(g(x_i))$, where $N_i(...)$ is the i-th instance of the noise function, we extend Equation 3 as follows:

$$\begin{aligned} g'(x_i) &= N(g(x_i)) \\ \rho_1(p) &= g'_1(x_i) \\ \rho_i(p) = \rho_{i-1}(p) + g'_i(x_i) - \frac{1}{w}\rho_{i-1}(p)g'_i(x_i), & \quad (6) \\ i &= 2,\ldots,n, \\ w = w_1 = w_2 = \cdots = w_n & \end{aligned}$$

This allows the creation of details in different scales, amplitudes and frequencies at each local part of the model under the influence of a given primitive.

4.0.1 Differentiability

One important aspect to be considered is the differential properties of the shape produced by blending different noise functions using equation 6. The experiments show that the detail patterns produced by the blending of different compositions of Perlin Noise with the Wyvill implicit primitives, using equation 6, produce details that seem to blend continuously and smoothly through generated shape. To completely explain the behavior of the modified blending function we must analyze the result of the composition of the Perlin Noise which is based on a fifth-degree interpolator - that has continuous second-order derivative - with the Wyvill blending function that has G_1 geometric continuity in the seams of overlapping regions. We intend to analyze this in a future work.

Algorithm 4 presents the proposed method in pseudocode.

4.1 Parallelization

The computation of the combined procedural and implicit modeling is rather computationally intensive. However, it is not difficult to leverage the massive parallel computational power of GPUs to compute such models as the density value $\rho(p)$ can be computed independently for each voxel v with coordinates given by p.

We implemented a straight forward CUDA kernel that implements the model defined by Equation 6. Our kernel function receives as input parameters the positions of the skeleton primitives, their weights, the associated noise function parameters and computes an array of density valeus in global memory. The *Turbulence Function* is computed in a CUDA device function called by the main kernel. Our implementation of the parallel Perlin Noise

Data: Set of primitives P
la - lacunarity, fr - frequency, a - amplitude, g - gain, s - scale,
n - numOctaves, p - noise percentage
width, height, depth - dimensions of the volume array
Result: Density volume V
for $V_{ijk} \in V$ **do**

$V_{ijk} \leftarrow 0$
$\rho \leftarrow 0$
$x,y,z \leftarrow i + rand() * s, j + rand() * s, k + rand() * s$

for $h = 0$ to $|P|$ **do**
$dist \leftarrow distance(i,j,k,P_h.x,P_h.y,P_h.z)$
if $dist < P_h.radius + P_h.influence$ **then**
$\rho' \leftarrow ComputeImplicitDensity(i,j,k,\rho,P_h)$
$g' \leftarrow (AddNoise(\rho', P_h,$
$P_h.n, P_h.fr, P_h.a, P_h.la, P_h.ga)$
$\rho \leftarrow \rho + g' - 1/(P_h.weight)(\rho)g'$
end if
end for

$V_{i,j,k} \leftarrow max(\rho, 1)$
end for

Algorithm 4: CloudGeneratorModified

was based on the implementation of Ron Farber [10] which uses shared memory. A sketch of the implemented kernel is show in pseudocode in Algorithm 5.

The resulting model stays in GPU memory at all times and is bound to a 3D texture used in the volumetric rendering step which is performed using a modified Ray Casting algorithm explained in the next section.

5 Visualization

Cloud visualization is a very important issue as the final appearance of the cloud model directly depends on how light interacts with its particles and how it is transmitted and scattered back to the viewer. There are several works in the literature devoted to cloud rendering which propose sophisticated models. One example of a method that produces impressive results is [1]. Basically, rendering a cloud requires two steps: a first one that computes the total incident light at each particle (*Light Scattering*) and a second step that computes tha light that reaches the observer (*Eye Scattering*). The first step requires solving the multiple scattering equation described in [11].

$$I(p,\omega) = I_0(\omega)\dot{e}^{-\int_0^{D_p}\tau(t)dt} + \int_0^{D_p} g(s,\omega)e^{-\int_s^{D_p}\tau(t)dt}ds \quad (7a)$$

$$g(x,\omega) = \int_{4\pi} r(x,\omega,\omega')I(x,\omega')d\omega' \quad (7b)$$

In equation 7a, $I_0(\omega)$ is the intensity of all direct light from direction ω. $I(p,\omega)$ measures the of intensity of all light from direction ω incident to particle p. $\tau(t)$ is the extinction coefficient of the cloud at depth t and D_p is the depth of a particle p in the path in which the light traverses the cloud.

The first term in 7a describes the intensity of light $I_0(\omega)$ that is not absorbed by intervening particles. The second term computes the intensity of light scattered towards p from other particles. Equation 7b defines the intensity of light from all directions ω' scattered in the direction of ω at point x. The term $r(x,\omega,\omega')$ is the *Bidirectional Scattering Distribution Function* and expands to $r(x,\omega,\omega') = \alpha(x)\tau(x)\phi(\omega,\omega')$. The factor $\alpha(p)$ is the albedo and describes the ratio between the scattering and absorption of light,

Data: Set of primitives P
la - lacunarity, fr - frequency ,a - amplitude, g - gain, s - scale,
n - numOctaves, p - noise percentage
width, height, depth - dimensions of the volume array
Result: Density volume V

```
i, j, k ← getCoordinatesFromThread()
if i < width and j < height and k < depth then
    V_ijk ← 0
    ρ ← 0
    x, y, z ← i + rand() * s, j + rand() * s, k + rand() * s
    for h = 0 to |P| do
        dist ← distance(i, j, k, P_h.x, P_h.y, P_h.z)
        if dist < P_h.radius + P_h.influence then
            ρ' ← ComputeImplicitDensity(i, j, k, ρ, P[h])
            g' ← AddNoise(ρ, P_h,
            P_h.n, P_h.fr, P_h.a, P_h.la, P_h.ga))
            rho ← ρ + g' − 1/(P_h.weight)(ρ)g'
        end if
    end for
    V_i,j,k ← max(ρ, 1)
end if
```

Algorithm 5: CloudGeneratorModifiedKernel

$\tau(p)$ is the *extinction factor* and ϕ is the *phase function* describing how light is scattered.

The *Eye Scattering* is modeled by 8 which describes the intensity of light arriving to the eye from direction $\vec{r}$ where θ is the angle between the viewer direction $\vec{r}$ and the direct light direction ω:

$$I(p,\vec{r}) = \int_0^{D_p} (e^{-\int_t^{D_p} \tau(s)ds})(I(t)\tau(t)\phi(\theta)dt \qquad (8)$$

5.1 Our simplified shading model

Our work does not focus in the rendering of clouds but rather in the modeling process. Thus, we did not implement the multiple or a single scattering algorithm for the *Light Scattering* step, but a simplified illumination algorithm based on a ray casting strategy. We consider that the intensity of direct incident light in each particle is constant which means that incident light is not attenuated by intervening particles. We only modeled the *Eye Scattering* which is the part of the illumination algorithm that computes the amount of light incident at each particle that is scattered and attenuated as it traverses the volume in its path towards the viewer.

For the sake of simplicity, assume that direct light is given by only one source in the direction given by l. Then, in our model, the incident light intensity at each voxel v whose center is at position p is given by a constant $I(p) = c$. The light $I_s(p,\vec{r})$ that is scattered from a voxel v at p towards the viewer in direction $\vec{r}$ is given by: $I_s(p,\vec{r}) = I(p)\alpha(p)\tau(p)\phi(\theta)$ where the extinction factor is approximated by $\tau(p) = 1-\rho(v)$ and the phase function $\phi(\theta)$ is given by the Rayleigh's phase function $\phi(\theta) = 3/4(1+cos^2(\theta))$, where θ is the angle between the viewer direction $\vec{r}$ and the light direction l.

For each pixel in the image we cast a ray $\vec{R}$ in direction $-\vec{r}$ and march from the first intersected voxel v_0 at p_0 towards the interior of the volume in a front to back approach. The marching process is divided in discrete steps k, $1 < k < N$ where N is the number of discrete steps in the path. At each step k we accumulate the total intensity $I_k = I_s(p_k) + I_{k-1}\tau(p_k)$, where p_k is the position of voxel v_k by summing the scattered intensity $I_s(p_k)$ at step k and the incident intensity it does not absorb $I_{k-1}\tau(p_k)$ (see equation 5 in [11]). The former process is a discrete approximation to the *Eye Scattering* equation in 8. All the images presented in the next section were produced using our simplified illumination method except figures 7e and 7f which used Levoy's volumetric shading [14].

6 Results

The experiments were performed in a PC desktop with an Intel i7 CPU with 16GB memory and a GeForce 750 graphics board. The volumetric models were defined on a 256x256x256 grid and the viewport has 512x512 pixels. With such configuration, we achieved a rendering throughput of above 30 frames per second using a basic non-optimized ray casting renderer. During model modification, we achieve a frame rate of about 20 frames per second which enables us to model our clouds in interactive times. In a more powerful GPU, we believe we can work with larger models and, by using more optimized algorithms, it is possible to produce smooth animations in realtime.

Experiments were made to compare the parallel versions of the basic algorithm and the modified one, with different parameter settings; those are described next.

In figure 4a the user starts with a very coarse model which is then turned into a cloud by modifying the noise parameteres (Figure4b). In the experiments we discovered that adjusting the frequency fr and the noise percentage np is usually sufficient to achieve satisfactory results. One must be aware that we also use a scaling parameter that divides the x, y and z coordinates of the voxels. This is related to frequency but has more drastic results. The larger the scale, the more violent the variation. In our experiments we kept the scale fixed at 0.02 and just modified the frequency values and noise percentage.

A comparison between the original model and the one proposed in this work is presented in figure group 5. The figures in the left show models computed using the basic algorithm that adds the same noise to the blending of Wyvill functions using equation 3. In the right, we can see the versions produced by changing noise per primitive and finally blending them, using equation 6.

We can see that the proposed method can produce more local variation that the basic one, permitting a greater level of expressiveness in the cloud design. In Figures 7a, 7b, 7c and 7d we show different examples of clouds produced by our modified method. In figure 6 we present an enlarged image showing one of our results rendered without using a scattering illumination model.

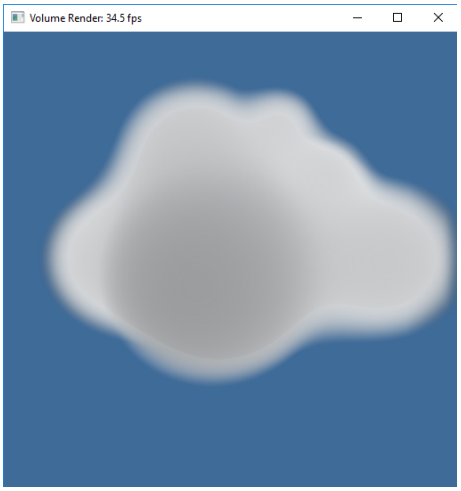
6.1 More than clouds

Finally, it is possible to use our system to produce other natural objects like rocks and planets as it can be seen in pictures 7e and 7f. In this examples, the power of blending primitives with procedural details in different scales come into evidence.

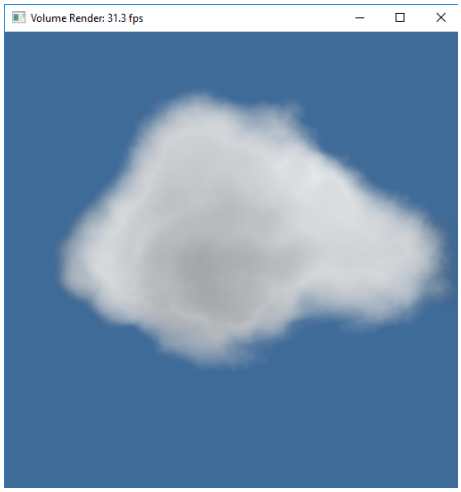
6.2 Final remarks

In our GPU cloud generation method it is possible to produce some convincing clouds using only a few primitives. In fact, all examples were built using less than 10 primitives. We have noticed that Perlin Noise seems to produce patterns in large scales that are not exactly similar to the patterns we find in clouds. But with practice and experimentation it is possible to produce nice effects. It is possible to try many configurations without overhead and choose the one that is most appropriate because the system responds in realtime to any modification in the parameters. This is one of the advantages of having a modeling tool in almost realtime.

Our visualization method is still not adequate as it does not contemplate multi-scattering. For this, it is possible to see in the pictures that the darker regions are in the inner parts of the model. We believe that by implementing multi-scattering these artifacts would disappear and the final appearance would be more consistent with the direction of illumination. All clouds in the examples were illuminated from the top position.



(a) Blending of Wyvill functions



(b) Cloud generated using the modified algorithm

Figure 4: Making a basic density field into a cloud

(a) Cloud modeled by the original algorithm

(b) Cloud generated using the modified algorithm.

Figure 5: Original (up) and the modified algorithm (bottom).

7 Conclusion

We proposed a GPU method for modeling volumetric clouds for games and virtual reality scenarios in realtime. It combines implicit with procedural modeling being able to easily define the overall shape of a cloud and its texture model. The main difference of our method is that it permits the modeler to perturb the implicit primitives independently before the blending step that builds the final shape. As our approach is completely implemented in GPU it is possible for a modeler to experiment many different configurations of clouds by changing the model parameters and observe the resulting appearance in almost realtime in a below entry level GPU. In the future we intend to investigate the combination of spring-mass models attached to the implicit primitives for animation purposes as in the system proposed by [20]. The use of other noise functions as Gabor Noise [13] and Wavelet Noises [2] in the modeling process is also a topic to be investigated. Finally, we would like to investigate ways to accelerate the computation of the density fields as it was done in [9].

8 Acknowledgements

The authors would like to thank CAPES and FAPERJ.

References

[1] A. Bouthors, F. Neyret, N. Max, E. Bruneton, and C. Crassin. Interactive multiple anisotropic scattering in clouds. In *Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games*, I3D '08, pages 173–182, New York, NY, USA, 2008. ACM.

[2] R. L. Cook and T. DeRose. Wavelet noise. *ACM Trans. Graph.*, 24(3):803–811, July 2005.

[3] Y. Dobashi. Inverse approach for visual simulation of clouds. In *Mathematical Progress in Expressive Image Synthesis I*, pages 85–91. Springer, 2014.

[4] Y. Dobashi, W. Iwasaki, A. Ono, T. Yamamoto, Y. Yue, and T. Nishita. An inverse problem approach for automatically adjusting the parameters for rendering clouds using photographs. *ACM Trans. Graph.*, 31(6):145, 2012.

[5] Y. Dobashi, K. Kaneda, H. Yamashita, T. Okita, and T. Nishita. A simple, efficient method for realistic animation of clouds. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 19–28. ACM Press/Addison-Wesley Publishing Co., 2000.

[6] Y. Dobashi, K. Kusumoto, T. Nishita, and T. Yamamoto. Feedback control of cumuliform cloud formation based on computational fluid dynamics. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2008)*, 27(3), 2008.

[7] D. S. Ebert. *Texturing & modeling: a procedural approach*. Morgan

Figure 6: Model created without scattering lighting effect

Kaufmann, 2003.

[8] D. S. Ebert and E. Bedwell. Implicit modeling with procedural techniques. *Proceedings Impliict Surfaces*, 98, 1998.

[9] A. Elhaddad, F. Elhaddad, B. Sheng, S. Zhang, H. Sun, and E. Wu. Real-time cloud simulation using lennard-jones approximation. In *Proceedings of the 29th International Conference on Computer Animation and Social Agents*, CASA '16, pages 131–137, New York, NY, USA, 2016. ACM.

[10] R. Farber. *CUDA application design and development*. Elsevier, 2011.

[11] M. J. Harris and A. Lastra. Real-time cloud rendering for games. In *Proceedings of Game Developers Conference*, pages 21–29, 2002.

[12] S. Kang, K. C. Park, and K.-I. Kim. Real-time cloud modeling and rendering approach based on l-system for flight simulation. *simulation*, 10(6), 2015.

[13] A. Lagae, S. Lefebvre, G. Drettakis, and P. Dutré. Procedural noise using sparse gabor convolution. *ACM Trans. Graph.*, 28(3):54:1–54:10, July 2009.

[14] M. Levoy. Display of surfaces from volume data. *IEEE Comput. Graph. Appl.*, 8(3):29–37, May 1988.

[15] B. Lipuš and N. Guid. A new implicit blending technique for volumetric modelling. *The Visual Computer*, 21(1-2):83–91, 2005.

[16] P. Man. Generating and real-time rendering of clouds. In *Central European seminar on computer graphics*, pages 1–9. Citeseer, 2006.

[17] K. Perlin. An image synthesizer. *ACM Siggraph Computer Graphics*, 19(3):287–296, 1985.

[18] P. Prusinkiewicz and A. Lindenmayer. *The algorithmic beauty of plants*. Springer Science & Business Media, 2012.

[19] W. T. Reeves. Particle systems—a technique for modeling a class of fuzzy objects. *ACM Trans. Graph.*, 2(2):91–108, Apr. 1983.

[20] J. Schpok, J. Simons, D. S. Ebert, and C. Hansen. A real-time cloud modeling, rendering, and animation system. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 160–166. Eurographics Association, 2003.

[21] B. Wyvill and G. Wyvill. Field functions for implicit surfaces. In *New Trends in Computer Graphics*, pages 328–338. Springer, 1988.

[22] G. Wyvill, C. McPheeters, and B. Wyvill. Data structure for soft objects. *The visual computer*, 2(4):227–234, 1986.

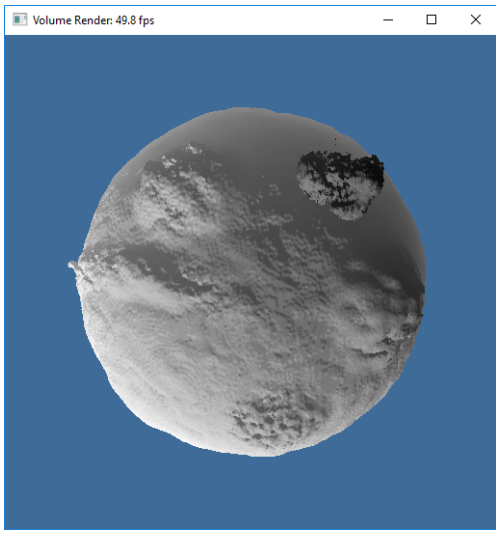
[23] J. Xu, C. Yang, J. Zhao, and L. Wu. Fast modeling of realistic clouds. In *Computer Network and Multimedia Technology, 2009. CNMT 2009. International Symposium on*, pages 1–4. IEEE, 2009.

(a) Another cloud generated using the modified algorithm.

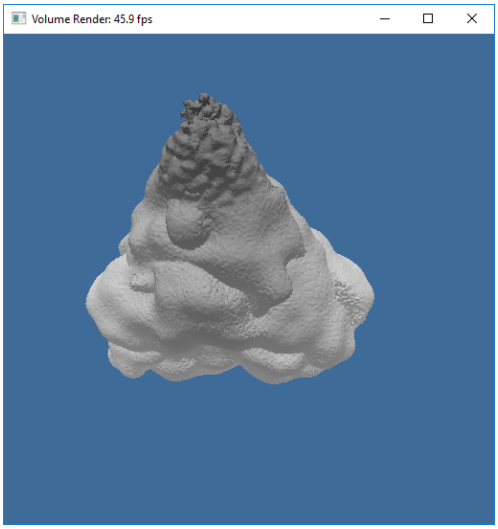
(b) Modification of the model on the left by scaling the density field.

(c) Thin cloud

(d) The same cloud in the left from another point of view.



(e) Mini planet with orbiting asteroid.



(f) Arrow point like object. Details in the top are finer than the rest.

Figure 7: More results