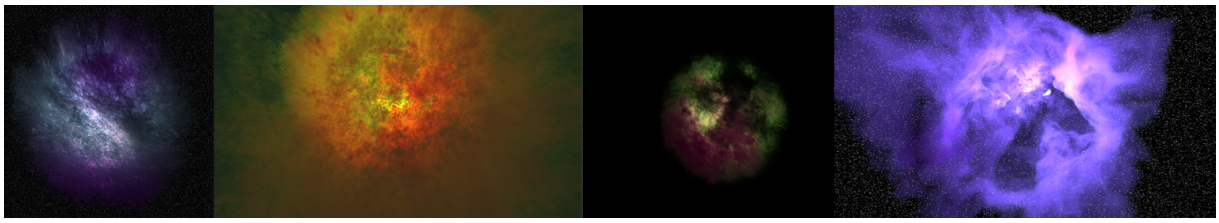
September, 30, 2020

Procedural generation of 3D realistic galactic dust and nebulas

Erwan Leria [1 2], Fabrice Neyret [2 3]

[1] Aix-Marseille Université,
[2] INRIA Grenoble Rhône-Alpes,
[3] CNRS



Real-time rendered nebulas generated with our model. They are ordered by progression of our model

Abstract

In this paper, we propose a model to represent and render full-volumetric 3D nebulas in real-time. This include the large-scale shape and the details of dust, the distribution of color and opacity, and the evaluation of the illumination. We rely on different types of procedural noises to shape the nebula and its details, and we constrain the noises so as to produce physically valid distributions. This allows as to analytically estimates the emissive areas as well as the shadowing. Our ray-marching implementation on the GPU run in real-time at 800×450 *resolution, and is compatible with integration in full galactic scene by limiting the marching to the space region inside a bounding sphere.*

1. Introduction

1.1. Context

As part of my Computer Graphics studies, I had the occasion to do my Master thesis with Fabrice Neyret. He works as Director of Research for the CNRS but belongs to the MAVERICK project-team at INRIA Grenoble Rhône-Alpes. The main topics of the team are expressive rendering, photorealist rendering, geometric modeling and real-time rendering. This paper reports the work I did during the six months of my Master thesis within the MAVERICK project-team.

A few years years ago, Fabrice Neyret did a joint project with RSA Cosmos and Observatoire de Paris-Meudon which was entitled ANR veRTIGE/Galaxy. The goal of the project was to find new visual model for to enrich a real-time 3D galaxy exploration. They managed to produce high quality large-scale scenes with a lot of details close to Hubble pictures. The rendering was done in real time without explicit data in input. The idea is how can we generate nebulas on the fly in a galaxy explorer. Through this document, we present a model to render nebula for real-time volume rendering on at least casual reasonable hardwares.

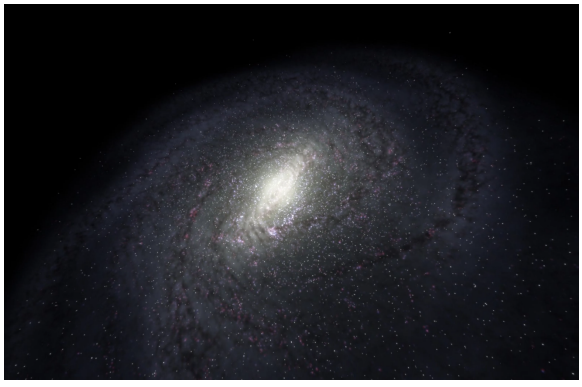


Figure 1: Picture of a galaxy from SkyExplorer, the RSA Cosmos software

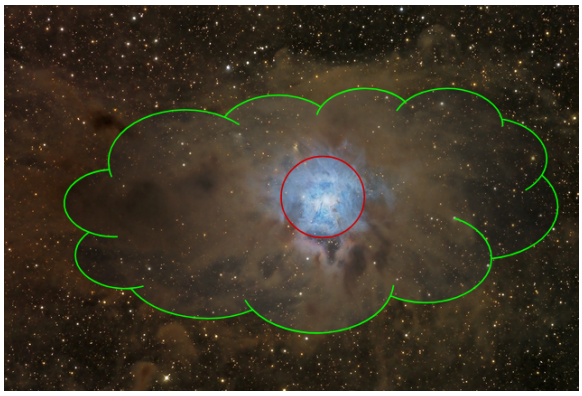


Figure 2: Iris Nebula - NGC 7023, *NASA*, the contours of the dust cloud in green, the contours of the bubble in red

1.2. Characterization of a nebula

First, on the nebula picture, we can see a galactic dust cloud. This cloud extends beyond the edges of the picture of course. Often, at the center of a nebula picture, there is a star that repels the gas around it. Here in the picture, the star is in the blue area. If the bubble drill out of the dust cloud, this open "windows" through which we can see the enlighten regions, and have Hubble getting nice pictures of it. The gas is naturally grainy and filamenty even in calm regions, but the bubble push and compress it as a spherical layer, with only some remaining dense chunks inside. This process creates a bubble that dig an empty space, and the few remaining gas near to the star is in the form of filamentary dust. This is the look of the cloudy filaments and the empty space in the center that interests us.

2. Previous Works

Realistic and large detailed scenes are very expensive for rendering, especially if we consider their global illumination. Most of these scenes are often made offline, as they do, for example, in the visual effects industry or for real scientific simulations based only on physical models. But it can takes hours, days or even weeks to end up with only a few seconds of animation. Another problem with large detailed scenes is the content editing + management (from disk to CPU memory to GPU memory), in addition to the cost of its rendering. Proceduralism consists in generating data on the fly, from a few control parameters. On the other way, textures allow to replace meshes that are costly to manage and to render image with details. Procedural textures do both: produce on the fly some kind of optical illusion replacing the complex explicit data. However, in order to achieve this, we need to be ten million times faster to reach 60 FPS. To produce fast detailed scenes, in Computer Graphics, one can use a quite common option, the procedural noise. This technique is very useful to generate natural looking patterns which could help us for the dusty aspect of the cloud. In this section we will present an overview of some previous works (general techniques) that will help us to do our nebula model.

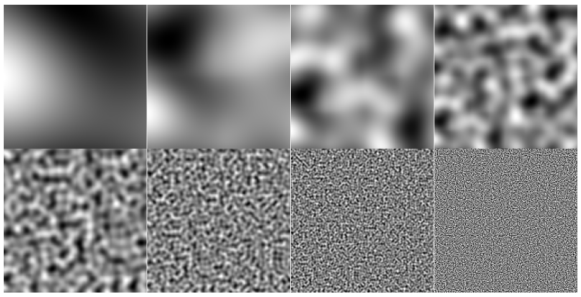


Figure 3: 2D Perlin noise on different frequencies

2.1. Procedural noise

For the computer graphics community, the benefits of procedural noise, such as the Perlin Noise [Per85] is that we can generate 2D or 3D textures that are both random and continuous, and we can also control them. Such textures can be generated on the fly and fairly fast. Noise functions are functions that take a vector of dimension n as input and that return a scalar value. Generating a random and continuous signal is good to give a coherant physical aspect to our nebula. To produce a such signal, first we define a regular grid of dimension n. Then for each point of the grid, we generate random gradients. Finally, we interpolate these gradients between them. The interpolations give a smooth and continuous signal.
In the veRTIGE/GALAXY ANR Project, the multiplicative noise was introduced and experienced

2.2. Volume rendering

To obtain the light intensity and color that should reach a pixel from the 3D density fields of the scene in order to avoid as more as possible cache and memory transfers. Therefore we don't use mesh nor any other set of data. In our case, to generate 3D textures, we need to render volumes. We call each element of our volume a voxel. In the interstellar environment all objects are not fully opaque, and light comes from different sources. Now, one of the main problems in rendering is how to render efficiently the returned intensity and color of a pixel on a screen depending on the 3D scene. We have to take in account the transparency of the volume, the light intensity and the color. For each ray that start from its associated pixel on a screen and that crosses the scene where the volume is, we can solve a light transport equation to get the final color and intensity of the pixel. This equation is also known as the rendering equation.
$I_\lambda(x) = \int_0^\infty e^{-\sigma_t dl} \cdot (\sigma_s) \cdot \psi \cdot \sum_0^{s_i} L_{s_i} \cdot e^{-\int_0^{s_i} \sigma_t dl} dl$, the accumulated transparency in blue, the local color in green in the current evaluated voxel, finally in red, we have the light intensity. Nevertheless, a such integral is not solvable as is on a computer. Consequently, we discretize this integral in order that it becomes usable. $I_{rgb}(x) = \sum_0^\infty \prod_0^l e^{-\sigma_t \Delta_l} \cdot \frac{\sigma_s}{\sigma_t} \cdot Illumination$

Finally from this we can deduce the following algorithm for the associated color of the pixel, depending on the voxels crossed by the ray starting from this pixel.

$$C_{acc} = C_{acc} + T_{acc} \cdot (1 - T_{loc}) \cdot C_{loc} \cdot Illumination$$

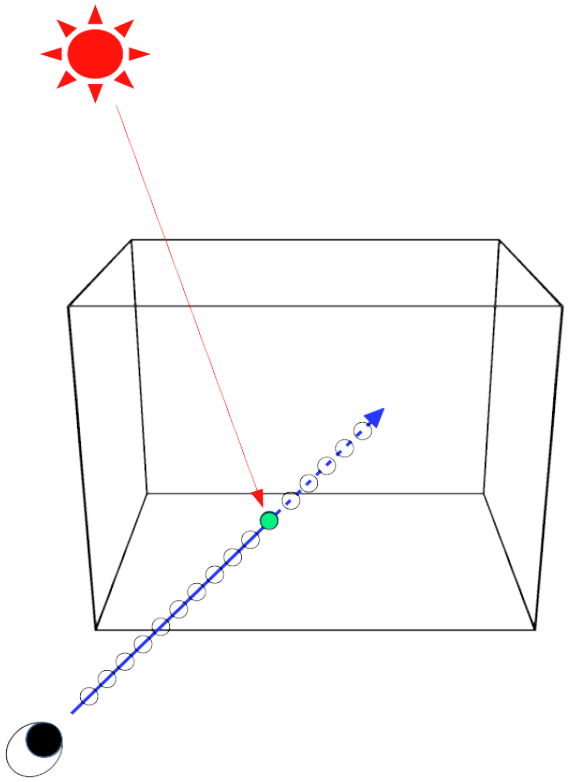


Figure 4: Volume rendering schema with (red) the illumination, (green) the local color and (blue) the transparency. The small circles represent the sampling of the casted ray

C_{acc} is the accumulated color. This is the final color returned on the screen one the ray is fully processsed. T_{acc} is the transparency accumulated at each voxels, T_{loc} is the local transparency of the current voxel, C_{loc} is the the local color of the current voxel, and $Illumination$ is the light intensity. An important point in real-time rendering is the global illumination. In fact, computing all the interactions between light and matter at each voxel would be very expensive in computing time, and so is not affordable in a real-time approach. That is the reason why we need to reduce the cost of such calculations. To generate textures on-the-fly, the uses of offline techniques to solve global illumination equations is not adapted.

2.3. Volume pre-integration

When rendering volumes, some artifacts due to the step size of rays can appear. We can call them : slices, which in fact correspond to a sort of aliasing on volumes. Slices occur when the sampling of the volume is higher than the length of the step of the rays. Thus some parts of the volume we want to render have discontinuous shapes or colors which visually creates slices. However, the smaller the step size is, the lower the performance will be because we would need more iteration for one pixel. Indeed, a smaller step size means to do more steps to cross the volume, and so, much more evaluation for one pixel, which is expensive for real-time.

To correct this problem, K. Engel et al. [EKE01] suggests the pre-integrated volume rendering. In addition, this techniques is suited for high-quality volume graphics and video games. As we want our virtual nebula to be high-quality, close to Hubble pictures, pre-integration is the ideal solution to do it in real-time. It will be used for to correct slices due to the function that modulate the transparency.
This technique consists in computing the integral of a non-linear density function, one of the parameters for the local transparency. So it take in account the value of the previous and the next slice. Thus we get a slab that interpolate smoothly between the two slices attenuating the artifact visibility.

3. Our nebula model

At this stage, we have the tools that will help us to render efficiently and rapidly a nebula. Keep in mind that we want to do real-time rendering. With the noise we can produce details on-the-fly, with volume rendering we can generate 3D shapes and with pre-integration we can do anti-aliasing on volumes. Now the other part of our work is to define parameters in order to control the noise and to give it a good look through a 3D scene. We are looking for a good visual nebula aspect.

3.1. Specifications

In our pragmatic approach, we need to define guidelines create our nebula model and to do protoyping with it. To summarize we want to :

- generate a plausible shape of nebula
- have a lot of details
- do it in real-time
- implement and complete the different existing tools that we noted before

3.2. Dust cloud

The dust cloud of a nebula, is so vast that sometimes even a picture can't show it completely. The gas of the star, affects the dust grains of the cloud and thus creates different effects on it. So we can see this cloud as a big volume. For the dust opacity, we define first a transfer function $\tau(x)$. This transfer function is used as blackbox to modulate a the grain densities which influence their opacities. This transfer function in the best case should follow a physic equation to have good and coherent results. As we do not want a flat distribution of density between our dust grains, we use the procedural noise with the intention of picking random values. The picked values from the noise make the density fluctuate at each voxel. We can represent this function as $\tau(n(x))$, $n(x)$ is the noise, v the current voxel. It is in this function where we insert the multiplicative noise, to see its behavior when used as a density parameters. With this, we are able to create different natural looking and detailed dust distribution in the cloud, tuning the density. We do not use any input data for the shape of the dust cloud, that is why, to give it a global shape we use a mask $m(x)$. This mask defines a shape complex or not. In our case, most of the masks we use have a spherical shape which is more simple. The mask enable to decide which region of the cloud is more or less covered of dust. We use it in our transfer function to influence the

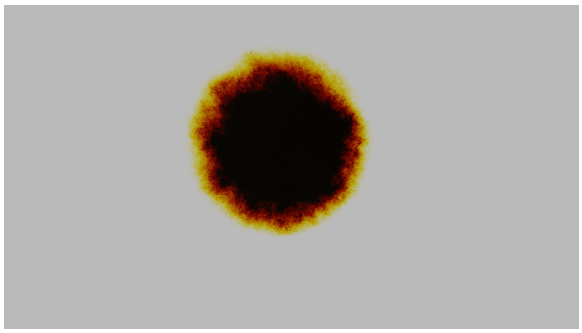


Figure 5: 3D dust cloud rendered in a grey background, the edges are lighter because there is less dust concentration

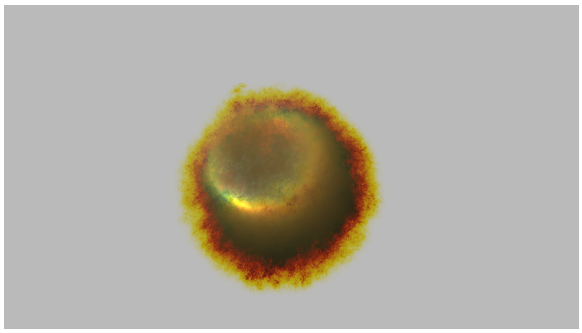


Figure 6: 3D dust cloud rendered in a grey background with a bubble at its center

area where the dust is. With the noise, the mask creates areas of density disturbances.

For the cloud, we can finally get the following formula for our density function with the mask $\tau(n(x)) \cdot m(x)$. Then, according to the discretized rendering equation (section 2.2), the local transparency of a voxel x can be written as :

$$e^{-\tau(n(x))\cdot m(x)\Delta l}$$

3.3. Bubble

Once we have our dust cloud, we can focus on the part of the center, the empty area. We call this area the bubble. The bubble repels the dust cloud all around it. This phenomenon is due to the gas compression. The bubble is then pushes the dust cloud in all direction from the inside. This bubble, as its names says, has a spherical shape. This spherical shape is determined by a mask $compress(x)$. If S is the star at the center of the bubble then, the mask is $compress(m(x)) = Gauss_{r,t}(|v - S|)$, with r and t the mean radius and the thickness respectively.

The bubble can be deformed by playing with its radius and its thickness in order . For example, to deform the surface randomly, we use again procedural noise. However the value of the noise doesn't have to be the same as the previous one. Thus we get for the mask the following formula :

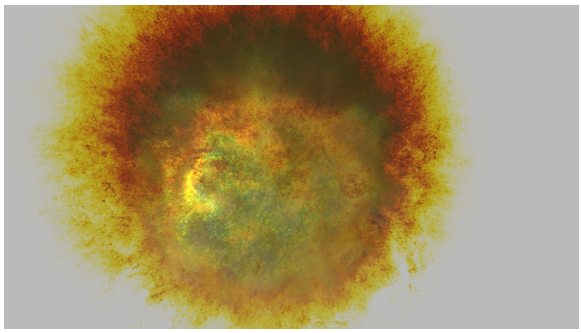


Figure 7: 3D dust cloud rendered in a grey background with a deformed bubble at its center

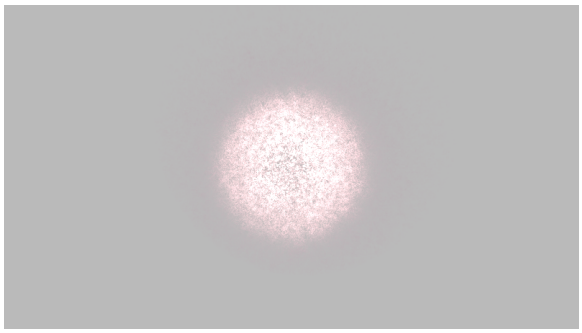


Figure 8: 3D bubble without stretching

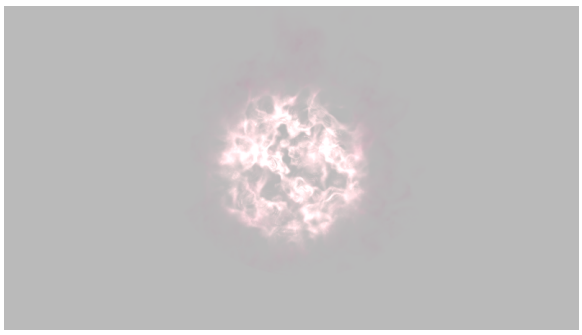


Figure 9: 3D bubble with stretching

$compress(x) = Gauss_{r+n_r(x),t+n_t(x)}(|v - S|)$, n_r and n_t are respectively the noise associated to the radius and to the thickness. We can then push the bubble in the cloud with the following formula $compress(m(x))$, this mask acts on the previous one. It indicates the region where the bubble is pushed. Then, to get the a filamentous look we propose to stretch the noise in all directions, according to the axis of the scene (x, y, z). Our stretching is sufficient to extend the anisotropic look of our nebula. It is very difficult to control the stretching in only one direction. Now we can write the stretching as $n(stretch(x))$, so we have then :

$$e^{-\tau(n(stretch(x)))\cdot compress(m(x))\Delta l}$$

For both the bubble and the cloud we define absorption coefficients α_{rgb} for each color component (red, green and blue). These coefficients affects the local transparency equation. These coefficients are used like an absorption spectrum of wavelength. If we assume that Δl is the step size of the ray that goes through the volume, finally for the local transparency at each voxel we have this :

$$e^{-\alpha_{rgb}\cdot\tau(n(stretch(x)))\cdot compress(m(x))\Delta l}$$

which is finally,

$$e^{-\sigma_t \Delta l},$$

where σ_t is the attenuation coefficient.

3.4. Color and Illumination

The transparency of our dust cloud is defined using noise (see section 3.2), through equation 3.2. We need to specify function $n()$ and $\tau()$ and tune their parameters so that they conform to requirements: some dust filaments are (almost) totally opaque, but some areas in between must sometime let the the light go through. In fact, when the density function $\tau() = 0$ then $e^{-\tau()} = 1$ which means the transparency will be more present in the nebula when the result of the function $\tau()$ is close to 0. On the other hand α_{rgb} is assumed to be an absorption coefficient between 0 and ∞ while basic Perlin noise, used in the density function $\tau()$, returns a signal in [-1,1]. Note that negative values are not physical. Relying on multiplicative noise do provide $[0, \infty[$ range, but if the histogram peaks too high, near-zero values will be very rare, so that no transparent areas will show up. So, we have to.

Each voxel has its own color. This color, annotated in green in the formula is called the local color. We calculate the local color :

$$C_{loc} = \frac{\sigma_s}{\sigma_t}$$

σ_s is a colormap. The color varies depending on the distance from the mean shell to the current voxel.
We want to add light to our scene without computing all the expensive part of light interactions, because we cannot afford . The local material color 3.4, with α_{rgb} and $\tau()$ considered as parameters of σ_t. A reminder that each of these parameters is wavelength dependent. In our implementation we consider (R, G, B) vectors instead of wavelength. The previous paragraph already deals with the density function $\tau()$. We need to tune σ_s and α so that they conform to requirements, either from astrophysical descriptions or target images. α_{rgb} depends on the dust cloud composition, but it can also be tuned from images, looking at the chrominance

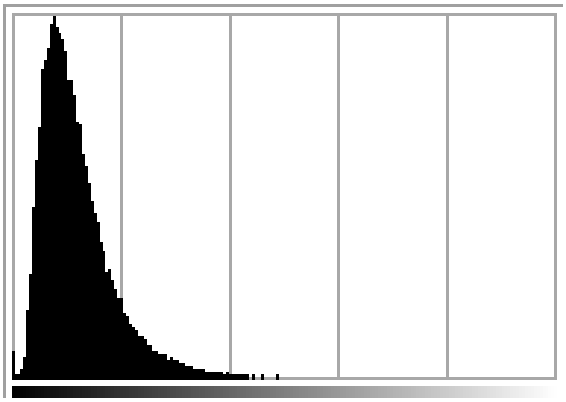


Figure 10: Histogram of the fractalized noise we use for density function

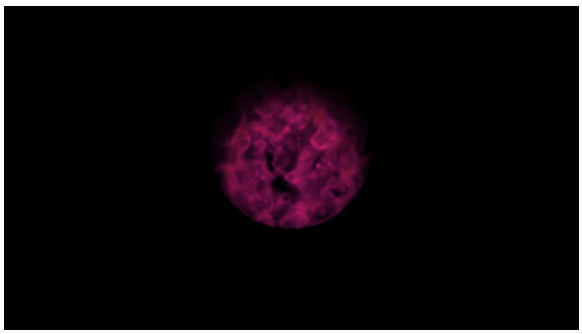


Figure 11: Nebula in a spherical bounding box

shift on quasi-transparent parts. σ_s can have several origins. For usual material as for reflection nebulas, it is the intrinsic color of the material. For the H|| nebula we are primarily interested in, it is the strong emission due to recombination after absorption of the central star strong UVs. In this case, theory as well as observation tells that the first dust shell illuminated by the star emit "blue" light (in fact, O||| emission peak that is usually mapped as blue in images) , then the slightly deeper one emits "green" (H_α emission peak usually mapped as green in images), then a last thin shell emits "red" (figuring the S|| peak emission.) As explained in section Bubble, our bubble model consists in an analytically deformed spherical shell. So at any given location x in space, considering the radial segment to the star we can easily obtain it's depth withing the dust star-wise, from its distance to the star and the shell deformation along this segment. We rely on an analytical LookUpTable to encode the color α corresponding to the offset from the minimum dust radius in this direction.

4. Implementation details

All the prototypes are done in fragment shaders. To do this, we use Shadertoy, a webGL/GLSL website that allows users to directly render fragment shaders.

4.1. Balancing the computation time

First, for the volume rendering we decided to use the ray marching technique. We consider an uniform sampling for the steps of the rays, the steps have the same through the process. However processing a ray for each pixel of the viewport is really expensive if some rays never cross any shape. We want to avoid rays which never cross the nebula or the dust cloud because they would cross nothing. Let $t_i, 0 \leq i < \eta$ be the i^{th} step of the ray, k its length, and η the the number of steps needed to reach the far plan of the scene (or to cross completely the further object of the camera), from η and k we get the maximal distance traveled by a ray of a pixel x :

$$dist(ray(x)) = \eta \cdot k$$

We assume all the steps have the same size. When a ray is cast from a pixel, the ray must to do at least η steps. However, to have a quite good quality for our volume, if we take a step size which is too big, the rendering will have a bad quality, the ray would skip details and some important geometric parts of the volume and create slices, however the ray will cross faster the scene. If the step size is too small, the rendering will be more accurate thanks to its sampling, but the ray will cross the scene very slowly. If we only divide the step size by 2, then :

$$dist(ray(x)) = 2.\eta \cdot \frac{k}{2}$$

then it is obvious to see that η would have to be two times greater. So the ray would have to do more evaluations. Which lead to more computing time, that decreases the performances. To have a good quality for our nebulas, we render them into a spherical canvas, which is a bounding sphere. With this technique, we end earlier the rays that cross nothing. The choice of the step has to be balanced between according to the hardware capabilities.
In our case here is no need to take a very small step because the pre-integration smoothes the rendered shape.

4.2. Implementation of procedural noise

If we consider the computing time of the Perlin Noise, then it would be judicious to use the Simplex Noise, a faster version of his own noise proposed by Ken Perlin, in order to do only *n+1* evaluations at each element composing our texture, n the dimension of the rendered image. A simplex is the smaller shape of $n+1$ vertices that can be make in n-dimensional space. Less evaluations allows to make a faster rendering for real-time rendering. We want to control our noise, giving it variations more natural. So we use the procedural noise as input into a Fractional Brownian Motion function, that is to say, in a simply way, a function that fractalizes the noise [Mv68]. We add up the noise on different scale. In this way it enriches the result because it adds details on the small scales. In addition of our nebula model, we also try to experiment the insertion of multiplicative noise for the generation of 3D textures. Multiplicative noise can be considered as a multi-fractal noise where we multiply different scales of noise instead of adding them up. The advantage of multiplicative noise, is that it keeps a constant mean no matter the scale. The human eye perceives fractalized patterns as natural since most of the natural objects and perturbations are non-linear. By using a procedural noise, we intend to reproduce the realistic aspect of nebulas.

4.3. Illumination calculations

The $Illumination()$ term accounts from the incoming light reaching the current location in space.
It results from the multiple paths through which the light transport source light to this point, potentially via multiple scattering. Multiple scattering simulation is out of reach of real-time rendering, where only the direct illumination - with some potential shadowing - from one of several light sources is accounted. Still, in the ray marching integral computing the current pixel value, this would yield a second integral to be evaluated at each sample voxel, i.e. a complexity n^2 for a volume n voxels wide, which is not affordable. So we rely on an analytical estimation of the shadowing, using the same principle as above: we know the shape of the distorted bubble along the segment from star to current location to be lighted, and the average density along given my mask (since the multiplicative noise just shuffle details in the local distribution). From that, we can analytically estimate $Illumination = I_{source} * exp((l - l0)\rho)$

5. Results and Evaluation

At this point, our model is ready. We want to compare our nebulas with real pictures of nebulas.

5.1. Aspect of the shape

First, we are interested in the shape.

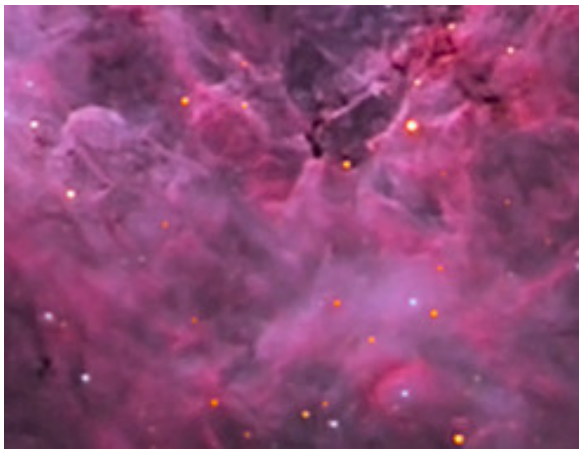


Figure 12: Zoom on Eta Carinae nebula

We can see that our model can reproduce the wavy surfaces of real nebula (Figures 12 and 13). This cloudy look is done with the mask of the bubble (see Figure 13). We can see it on a larger scale (see figures 14 and 15)

Figure 13: Reproducing the wavy pattern of a real nebula

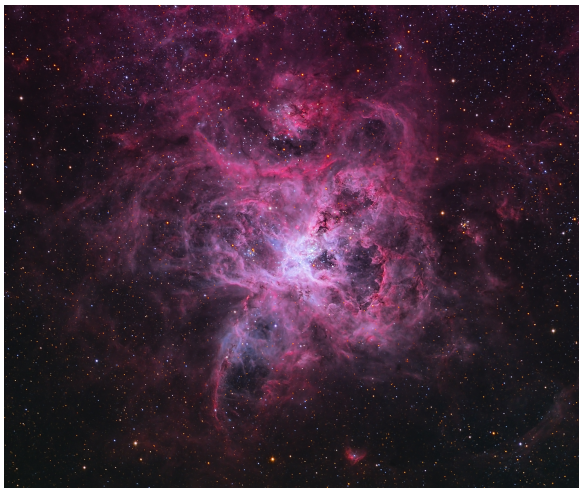


Figure 14: Eta Carinae nebula

5.2. Color aspect

An other interesting point is the colorimetric aspect. The colors and the lights are important because they give the more or less realistic look. Here we propose different tests to see how behaves the nebulas.
On figures 16 and 17 we play with the frequency of the noise in input of the density function. We see that the frequency of the noise affects the transparency.

Now, on figures 18 and 19 we play with the absorption coefficients that are used to calculate σ_t. Here, we make varying the red component of the absorption coefficients. We see that the frequency of the noise affects the transparency. The red is absorbed in the first test, it is not in the second. We see that the blue and green seems to remain when the red disappear. However, remark that the shape doesn't disappear, the colors with a bigger red component are still strong enough to be visible.

On figures 20 and 21 we can se how vary the nebula according

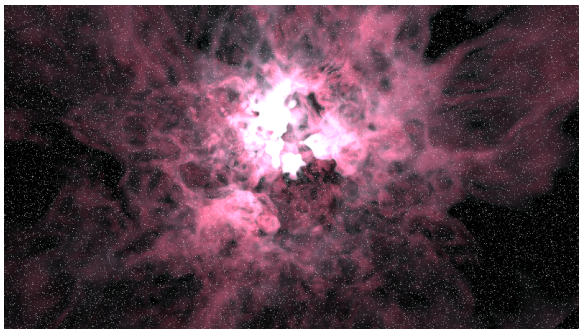


Figure 15: One of our nebulas made from our model

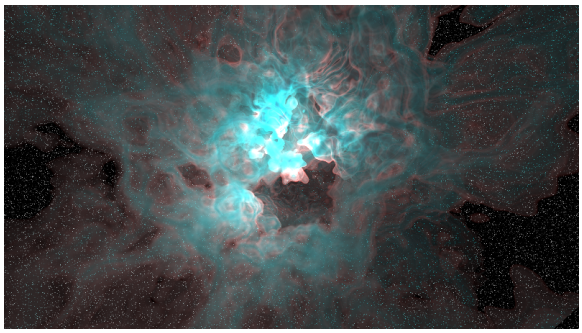


Figure 16: One of our nebulas made from our model

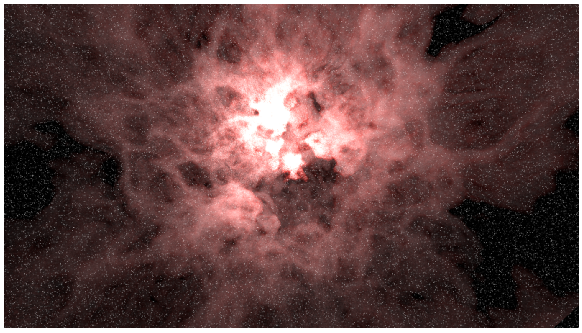


Figure 17: One of our nebulas made from our model

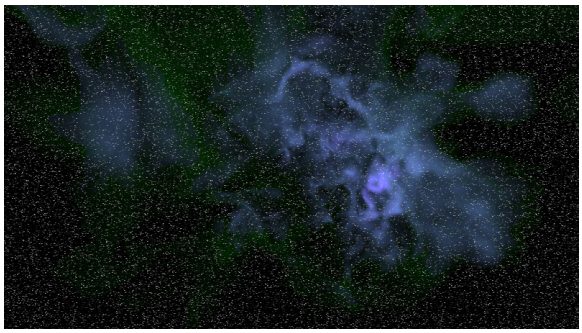


Figure 18: One of our nebulas made from our model with low absorption coefficient

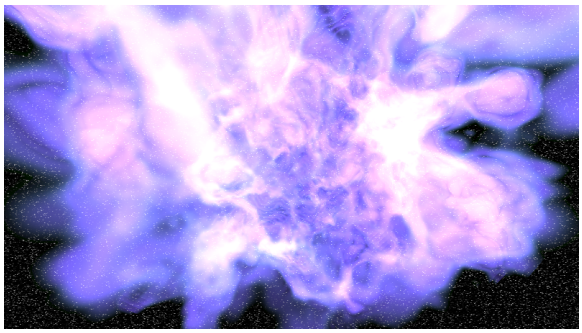


Figure 19: One of our nebulas made from our model with high absoprtion coefficient

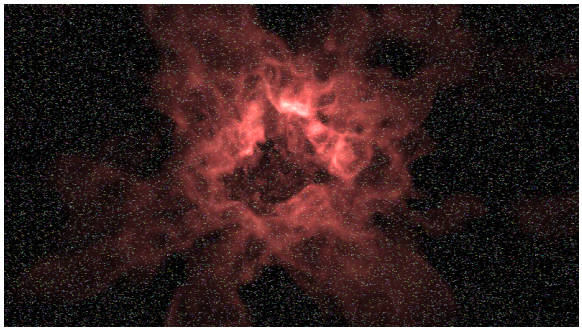


Figure 20: One of our nebulas made from our model with more light

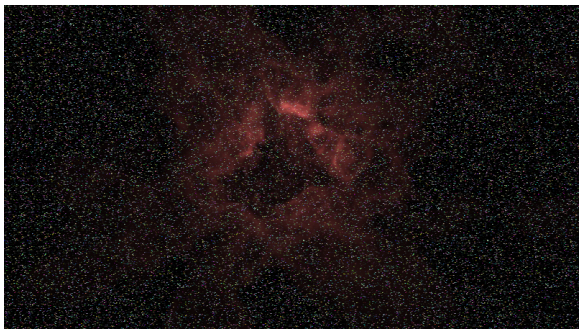


Figure 21: One of our nebulas made from our model with less light

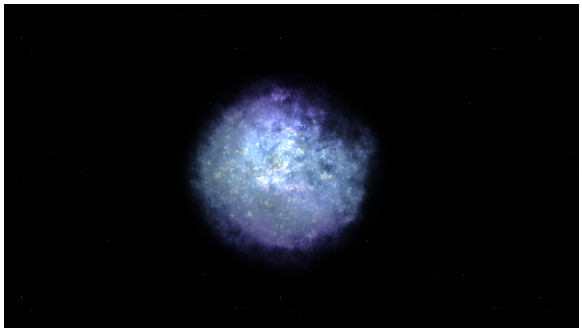


Figure 22: One of our nebulas made from our model with black background

to the illumination.

Let's take a look at our nebulas when the background has not the same color. Indeed, we can obviously see that on figure 23 the background is white, in the center of the figure, there is a nebula. This nebula is the same than the one on 22, there is no modification of colors.

We can compare our two first prototypes with out last one.

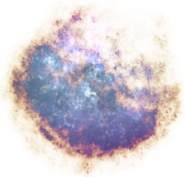


Figure 23: One of our nebulas made from our model with white background

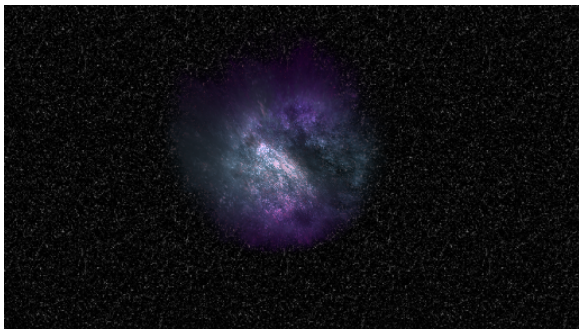


Figure 24: First model prototype for nebula

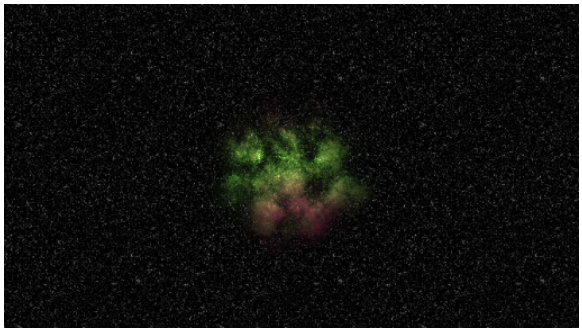


Figure 25: Second model prototype for nebula

In our first prototypes, we had a lot of artefacts. It was mostly due to the deformation we were applying, without using noise for it (see 25. However, in figure 24 the density of the cloud was heavier, which was visually better, but there wasn't this effect of filaments as we can see on 26

Results are very encouraging. For a 800x400 window resolution, our shader Nebula 16 has the following performances :

- 60 fps ⟶ Nvidia GeForce GTX 1080 Ti
- 27 fps ⟶ Nvidia GeForce GTX 770
- 27 fps ⟶ Nvidia GeForce GTX 1050

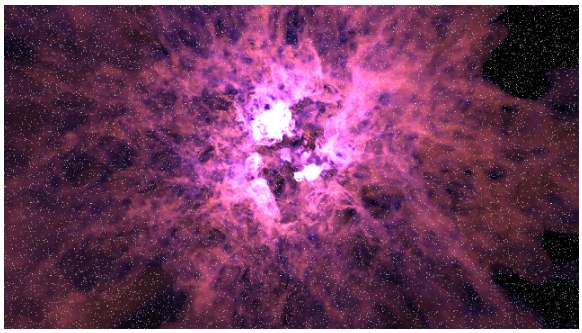


Figure 26: Last model prototype for nebula

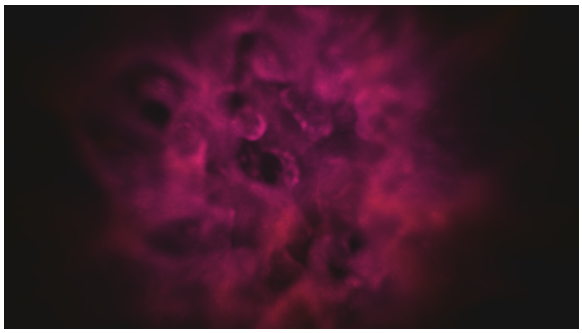


Figure 27: Performances test shader

- 06 fps ⟶ Nvidia GeForce 920M

The performances test were done with the shader, rendered in figure 27.

6. Conclusion

After several prototypes of nebula models for real-time volume rendering, we managed to get a better control and a better look on our nebula volumes. So we have presented a nebula model for real-time rendering. The noise proved to be very useful for these kind of scenes to generate details. Volume rendering is very efficient. These powerful tools are really adapted for real-time applications. We render non-existing nebula with our model. As far as we know, such models for real-time volume rendering are not common. Therefore there is no benchmarks to compare our model with another one of this kind. Our nebula model is sufficient and efficient for real-time rendering.

7. Future works

The nebula model can be enhanced in order to achieve very high performance graphics. A smart method for the volume rendering, would be to predict when we will have an empty voxel, in fact if we can predict this, then we could skip the empty areas instead of computing them. It would cost less evaluations and improve the performances.

Moreover, for the look of the dust cloud, we could have a more filamentous grains with multiplicative noise. We use the noise as a parameter in the density function but it could be also used in different ways. The difficulty, that we still have, is to stretch the noise towards a main direction as we can see with the filaments on most of the nebula pictures. In figure 28, the filaments seems heading towards a principal direction. In addition, we still have some problems on the color and the light transport, as we want to do real-time, we do not have the guarantee to have a good physic for both lights and in particular the shadows. It can be due to the choice of approximating the illumination. The light model and the color interactions can be improved. To have physical colors, it would be interesting to add a ionization system. With this, the emission and absorption spectrums (colors) would be more physical and so we could detect the areas that has to be ionized.
For the motion of the nebulas, we could insert in our model a physicaly based animation. The diffculty would be to animate the noise with coherence. Finally, for future works, our model could be integrated in the existing software of the previous collaborator that co-work with INRIA on the ANR project veRTIGE/Galaxy RSA Cosmos.

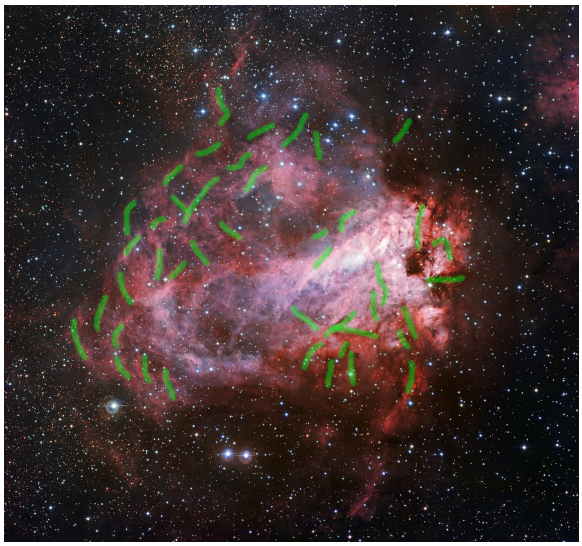


Figure 28: Messier 17, credits : ESO ; in green : the annoted orientations of the filaments

Références

[EKE01] Engel K., Kraus M., Ertl T.: High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware* (New York, NY, USA, 2001), HWWS '01, Association for Computing Machinery, p. 9–16.

[Mv68] Mandelbrot B. B., van Ness J. W.: Fractional Brownian Motions, Fractional Noises and Applications. *SIAM Review. Vol. 10*, Num. 4 (octobre 1968), 422–437.

[Per85] Perlin K.: An image synthesizer. *SIGGRAPH Comput. Graph.. Vol. 19*, Num. 3 (juillet 1985), 287–296.