Rendering Smoke & Clouds

Jürgen Treml

juergen.treml@gmail.com
www.juergentreml.de

Abstract

This paper is to give a short summary on the topics covered by a talk given during the course 'Game Design' at the Technical University of Munich. Among the topics is an overview about clouds in the real world, what they are and why we can see them as well as two different approaches on rendering and displaying clouds on a computer. The first one of these approaches is based on physical modeling and an automated, noise-based generation of clouds whereas the second one is a more artistic approach on creating and displaying clouds. Last but not least I will discuss, very shortly, a few ways how to animate clouds. Much unlike the title suggests, the topic of smoke is somewhat put aside, as rendering it is quite similar to rendering clouds whereas animation, where smoke differs from clouds, has not been part of the talk.

Keywords: Cloud Rendering, Volume Rendering, Splatting, Scattering, Random Noise, Perlin Noise

1. Introduction

In the real world, we're that much accustomed to seeing clouds, that quite often we don't even take much notice of them. When it comes to certain computers games, especially flight simulators and games which heavily rely on outdoor scenes, it's just the opposite. We tend to pay quite some attention to the clouds in the sky, unfortunately not for their beautifulness but for their absence or, if present, usually poor quality. Today's games are doing just fine as long as you're busy enough keeping your aircraft under control and don't find the time to enjoy the environment you see outside the aircraft's windows, though one must not forget that things have improved a lot over the last years with the potential of GPUs steadily increasing. Still, so far, there seems to be no perfect or at least optically totally convincing way to render clouds but there are different and promising approaches that do achieve good quality.

Basically there are two different sides where these approaches come from. One is to try to approximate, as good as possible (or necessary), the physical principles and mathematical equations behind clouds. Doing so seems quite straight forward and by definition, if approximated just well enough, should be the most precise and convincing way to simulate and display clouds, just right for, e.g. meteorological simulations and scientific studies. Nevertheless, for a game this doesn't have to be the preferred way to achieve clouds. For a game, usually you need optically convincing clouds, or in other words, clouds that just look good. They don't necessarily have to be physically correct, and in fact quite often they're not. Instead you're worried about your framerate, i.e. how fast your clouds can be rendered and the game designer at the desk next to you expects you to give him as much freedom as possible in influencing the appearance of the clouds to be seen in the game. A physically exact simulation might not be the best way to achieve those two goals.

For this, in the following, after a short paragraph about clouds in the 'real' world, I'm going to present some of the basics and a few examples behind each of the two approaches. I'm going to get into cloud generation as well as cloud rendering and, in short, also into the animation of clouds. But first of all, we should get an idea of what exactly clouds are.

2. Clouds in the 'real' World

What exactly are clouds? Obviously they're usually more or less white spots in the sky. But that leaves two questions open: Where do they come from and why do we perceive them as 'more or less white', or in general, colored spots? Both of these questions are of great interest for implementing a cloud rendering system with the goal to render clouds as realistic as possible.

Clouds are basically volumes of air containing a significant amount of water in condensed form. They form when rising water vapor cools down with increasing height and finally condensates. This is of course just a very basic view but enough for our purpose of rendering them. In reality, the factors influencing cloud formation, movement and dissipation are quite complex and among the most basic ones are temperature, pressure, humidity and the ratio of condensation to evaporation in a volume of air ((Wolke)). Whereas this is of interest for a physics-based simulation of cloud dynamics, these details are not relevant for just rendering clouds. There, we are more interested in why and how we see clouds.

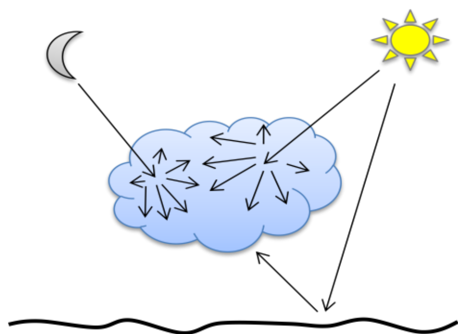


Figure 1: Interaction of light with clouds

Clouds are characterized by various interactions with light as Figure 1 illustrates. Incoming light particles are scattered by the water droplets in clouds. This scattering is a multiple anisotropic and multi-directional scattering of ambient light, as well as light from the sun or other celestial objects and light of those objects reflected from the ground. The fact that light particles may be and in fact are scattered multiple times in an anisotropic way is responsible for the bright appearance that clouds quite often possess. Besides all, clouds of course do also absorb light as it passes through them. Given that one can now try to approximate and simulate this behavior in order to create realistic virtual clouds.

3. Virtual Clouds based on physical Models

3.1. Generating Clouds

A first and straight forward approach on generating clouds would be to implement the physical and meteorological models behind clouds, then model and define your environment and finally get areas with cloud coverage as an ad-hoc result of your model. This is of course a quite complex approach on the topic but in case you're for example planning to implement a complex simulation of clouds and their dynamics later on in order to animate your clouds, then you're going to need all the physics anyways, so why not also use it for generating clouds. Going with a strictly physics based implementation, you'll have to account for things like potential temperature of air parcels, buoyant forces, environmental lapse rate, saturation mixing ratio, thermodynamics, fluid flow, vorticity confinement and water continuity as proposed in (Harris, Baxter III, Scheuermann, & Lastra, 2003).

In case you don't need an exact physical simulation of your clouds and just want to have optically-convincing clouds which may be even static, i.e. don't form, dissolve or flow in any way, you may be better off with a computationally less expensive approach. A good start for that is functional noise. It is memory efficient and quite fast compared to simulating the physics behind clouds. With clouds being a natural phenomenon, 1/f-noise seems an intuitive choice. With its stochastic distribution and self-similarity it matches well the structures we can see in nature, including clouds.

$$N(x,y)=\sum_{i=1}^{n}\frac{1}{2^i}B\left(\frac{1}{2^{n-i}}x,\frac{1}{2^{n-i}}y\right) \qquad (1)$$

Formula (1) shows the basic structure of a 1/f-noise function in two dimensions. With B being some basis function, and n being the number of noise octaves to generate, (1) accounts for scaling the amplitude of each octave of noise reciprocally with its frequency. Put simple, this means that large variations in noise repeat much less frequently than small variations. The base function B can be anything, but for clouds a stochastically distributed texture or pseudo-random generators seem to make sense.

Examples and further information on functional noise can be found in (Perlin Noise), (Perlin) and (Krüger, 2006/07). (Perlin) also shows an example of the use of functional noise for a 2D animated cloud field. But consider that you can create functional noise for an arbitrary number of dimensions, which results in the fact that you can easily use it to create a 3D density distribution for clouds or even animate this in time using 4-dimensional noise.

Last but not least a noise-based approach on generating clouds can be combined with post-editing methods to allow users, e.g. artists, to control the distribution of clouds. A way to do so

might be similar to what *Terragen*[1] does with landscape modeling, i.e. providing the user with some kind of a brush to flatten or heighten the generated noise in certain regions. This way, an artist could specify regions where more clouds are present and regions where there are less or (almost) no clouds.

3.2. Rendering Clouds using Volume Rendering Techniques

3.2.1. The Volume Rendering Integral

Now that we have a 3-dimensional density distribution of clouds in the air, we can start rendering clouds. Given this form of underlying data, volume rendering appears to be a native way to display it.

Starting from our model of light travelling through clouds and interacting with the water droplets in a cloud, we can try to compute the amount of light which reaches the viewer looking at that cloud calculating the absorption of light as it travels through the cloud.

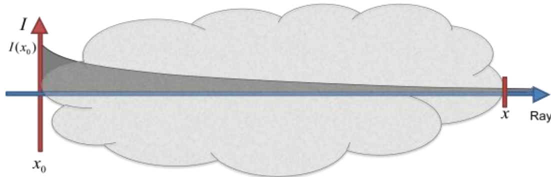


Figure 2: Light travelling through a cloud and being absorbed.

In mathematical terms we can describe the intensity *I(x)* of light exiting the cloud in respect to the initial intensity $I(x_0)$ entering into the cloud as follows:

$$I(x) = I(x_0)e^{-\tau(x_0,x)} \qquad (2)$$

Where τ is the optical depth of the medium and defined as:

$$\tau(x_1, x_2) = \int_{x_1}^{x_2} \kappa(x)dx \qquad (3)$$

Here, κ describes the absorption of light at a certain point in the cloud.

As clouds do not just absorb light but light is also scattered within clouds, we're going to account for this by assuming that particles in a cloud can also emit light.

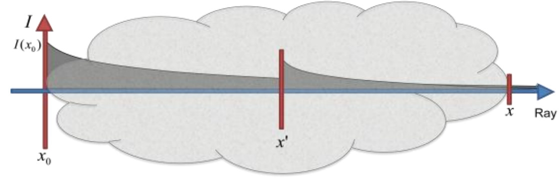


Figure 3: Cloud particles absorbing as well as emitting light.

This very simple model of course just accounts for single scattering of light in forward direction but will serve as a first start. Coming to the mathematical notation, our volume rendering integral changes as follows:

$$I(x) = I(x_0)e^{-\tau(x_0,x)} + \int_{x_0}^{x} q(x')e^{-\tau(x',x)}dx' \qquad (4)$$

In this case, q defines the initial intensity of light emitted by a particle at a certain point in the cloud and the rest of the term accounts for the attenuation of the light emitted at a certain point while it 'travels' through the rest of the cloud.

So now that we have the equations to describe light travelling through clouds (we'll leave multiple scattering for the next chapters) we could try to implement a cloud rendering engine. Unfortunately there is no closed form for these equations. Nevertheless we can try to approximate those integrals by discrete sums, which lead to two quite basic methods of volume rendering.

3.2.2. Direct Volume Rendering: Raycasting and Splatting

Let me start with the most intuitive way of direct volume rendering: Raycasting. This is a so called backward volume rendering technique which means we start from the image plane and go pixel by pixel shooting an imaginative ray through our volume. Along each ray, we're going to accumulate the density at discrete intervals. As our ray is unlikely to hit the voxel center at the chosen intervals, we'll have to decide what to do in this case. We could just take the value of the nearest voxel or we could interpolate the value from the values of the neighboring voxels, e.g. using trilinear interpolation. Reaching the backside of our volume dataset, we finally know how much light from

[1] http://www.planetside.co.uk/terragen/

behind the volume dataset reaches the image plane at this pixel. We do this for all pixels in the image plane we finally get a rendered image of our volume. There is of course a lot of room for improvements in this but I just tried to point out the basic idea behind this technique. (Westermann, 2006/07) describes this in more detail.

Instead of going pixel by pixel on the image plane, shooting rays through the volume, we can also try to do this the other way round, which leads us to what is called splatting. If raycasting is backward volume rendering, this would be forward volume rendering now. This time, instead of going pixel by pixel on the image plane, we'll start from our volume dataset and go voxel by voxel, trying to 'splat' each voxel on the image plane.

That said, a voxel will of course not influence just one pixel but several pixels. Therefore, a quite common way of projecting voxel onto the image plain is using a Gaussian kernel which is illustrated by the following figure.

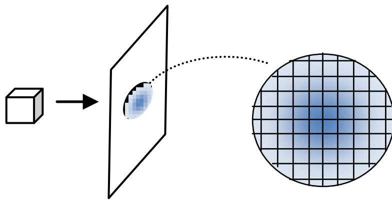


Figure 4: Projecting a voxel on the image plane using a Gaussian kernel

An example for this method used in cloud rendering is (Dobashi, Kaneda, Yamashita, Okita, & Nishita, 2000). Given a grid of voxels representing a discrete density distribution (of clouds), for each cloud, they render a 2D billboard of the cloud by looping through all voxels and projecting them on this 2D plane using a meatball function (instead of a Gauss kernel). The basic reason for metaballs lies in the fact that their effective radius can be controlled more precisely than for a Gauss kernel. (For further details about metaballs, please refer to the original document). Given those billboards they now render an image from the view of the sun, orienting the billboards perpendicular towards the sun and starting with the closes to the sun, rendering them to the framebuffer (initialized to 1) by multiplying the framebuffer pixel values with the billboard pixels. Thus after each billboard, the framebuffer represents a map of the amount of light reaching from the sun through all billboards (i.e. clouds) so far. For each billboard in the line one can just retrieve the corresponding pixels from the framebuffer and multiply them by the sunlight color to get the color of this billboard / cloud, store it, then render this billboard to the framebuffer and repeat this for all remaining billboards. Besides the the most important thing, i.e. the clouds' shading, this method also returns a shadowmap that can be used for shadows cast on the ground by these clouds later on. The rest of the rendering process is pretty much straight forward. Orient the billboards towards the viewer and render them to the framebuffer starting with the closest by multiplying the framebuffer color with the attenuation ration from the billboard plus the colors calculated during the last step.

As this method obviously accounts for single scattering only, I'd like to conclude this chapter about rendering clouds with a very short overview on how to extend this method (and the volume rendering integral) to account for multiple forward scattering.

3.2.3. Extending the Volume Rendering Integral: Multiple Forward Scattering

This chapter is intended to be just a short overview of a method presented in (Harris & Lastra, Real-Time Cloud Rendering, 2001). It extends the volume rendering integral presented before to read as follows:

$$I(P,\omega) = I_0(\omega)e^{-\int_0^{Dp}\tau(t)dt} + \int_0^{Dp} g(s,\omega)e^{-\int_s^{Dp}\tau(t)dt}ds \qquad (5)$$

The function in the second part of the sum is defined as

$$g(x,\omega) = \int_{4\pi} r(x,\omega,\omega')I(x,\omega')d\omega' \qquad (6)$$

This basically accounts for the facts that

- each particle does not only receive light from outside the cloud / the sun but also light scattered by other particles in the cloud
- the amout of light received that way is a function of the (spatial) angle (between current particle, other particles and the sun or light source)
- The exact characteristics are determined by the bidirectional scattering distribution

function (BSDF) and the phase function (e.g. Raleigh scattering)

As with each of these points one could fill pages of text, I'd like to refer you to the original document again for further mathematical details, but given the above integral, we're again in need for a discrete approximization as there is no closed form.

(Harris & Lastra, Real-Time Cloud Rendering, 2001) basically solved it by making a series of assumptions like

- Forward scattering accounts most for the optical perception of clouds thus restricting calculations to small angle around the forward direction.
- Assuming the BSDF and various other parts to be constant due to the small angle.
- Representing the light flow a small number of light paths in discrete directions.

With these and some more assumptions they managed to get a discrete solution of the integral which they then rendered using a enhanced / extended version of the method presented in (Dobashi, Kaneda, Yamashita, Okita, & Nishita, 2000).

3.3. Animating Clouds

As a conclusion of this chapter about clouds based on realistic models, I'm going to give a short overview on how to animate clouds.

Basically there are three different ways to achieve cloud animation, the first two of which are pretty obvious after what has been described in the chapter before.

First of course, if you have a cloud rendering model based on exact physics, i.e. implemented the physical equations behind clouds, cloud creation, movement and dissipation will be an ad-hoc result. This should be quite obvious.

In case you used (Perlin) noise for cloud generation, it might be a good idea to use that for cloud animation, too. As mentioned in the chapter about pseudo-random noise functions, these functions support an arbitrary number of dimensions. Thus, using 4-dimensional noise, one gets a 3D density distribution of clouds in the air plus a fourth dimension that could be used to animate this density field in time.

Now we have an exact physical model and a pseudo-random model, but as one might expect, there's a third way right in the middle between those two, i.e. a way that's not random but neither physically exact, thus allowing a possibly faster implementation and more control on the details. This method would be using / building cellular automata as described in (Dobashi, Kaneda, Yamashita, Okita, & Nishita, 2000).

The automata consist of cells corresponding to the voxels of our animation and carrying (binary) state variables. All you need then is a set of transition function describing the real world cloud behavior as realistic as you'd like it to be. An exemplary automaton can be found in the publication mentioned above but as you can imagine, this method is extremely scalable regarding its level of detail and accuracy.

As a last note, I'd like to point out that using this method you get a binary density distribution. So in the end you'll need some form of smoothening to make the result optically pleasing. Nevertheless it's a not too complex way of realistically and controllably animating clouds.

4. Virtual Clouds based on artistic Models

Having discussed some more or less physics-oriented approaches on virtual clouds, I'd like to present you with a last chapter now describing a completely different method of achieving clouds. It's a method that is completely oriented towards performance and artistic control and it's the system used in Microsoft Flight Simulator 2004 as described in (Wang, 2003).

4.1. Generating Clouds

As mentioned above, artistic control is one of the main factors of this method and therefore makes up most of the part of creating clouds. This basically means a human is going to manually design the clouds supported by a computer at a few of the steps.

First of all an artist will design the general shape of the clouds using a 3D editor and a simple box model. After that, a script will automatically fill these boxes with randomly placed, textured point sprites and do some filtering such as removing sprites to close to other sprites and thus not conveying to the optical perception of the cloud,

etc. This is done within the boundaries given by the artist such as the density of the cloud or other parameters.

Beside this the artist will specify the clouds color in a more or less complex way. He will specify the percentage of ambient color at certain times of the day contributing to the cloud color. Additionally he will specify vertical color levels and according colors for each cloud plus group parts of the cloud to be shaded similarly into so called shading groups. Finally the artist will set directional colors again for various times of the day.

As a last step, these clouds are exported into a file, storing only the sprite centers and sizes plus the coloring information specified by the artist.

4.2. Rendering Clouds

Rendering the clouds created before is now pretty much straight forward.

The first step would obviously be loading the cloud data from hard disk into memory. Now in each rendering pass one would render quads around the sprite centers according to the specified sizes and map textures to those quads according to the type of cloud. To improve naturalism and variety these textures are randomly rotated in the quad plane. Next, the quads are rotated perpendicular towards the camera.

The next step is to calculate the quads' shadings which is a function of the angle between their center point, the according shading group's center and the sun and furthermore takes into account the color levels specified by the artist described in the chapter before, interpolating between the discrete levels given. Last but not least, the quads will be rendered to the frame buffer.

For any details on the calculation of the quads' shadings, I'd like to refer you to the original paper as this would lead to far now. Nevertheless, the short summary gives you an impression on how few calculations need to be done during a rendering past. It clearly shows that most of the work has been done in the step before.

4.3. Animating Clouds

In (Wang, 2003)'s system of rendering clouds there actually isn't very much animation being done as there is no simulation of cloud movement. Nevertheless she presents a quite simple approach on how cloud formation and dissipation can be achieved.

Cloud dissipation, as she describes, can be achieved by slowly increasing transparency on the clouds. To achieve optically convincing results, one would start increasing transparency on the edge particles of a cloud first, the doing the same for the cloud's core, so that the clouds less dense outer parts dissolve before the remaining core finally dissipates, too.

As one might already guess, cloud formation is exactly the same done just in opposite direction.

5. Conclusion

In this document I've basically presented two different types of cloud rendering systems. One being based on physical models trying to provide maximum accuracy and the other one trying to give you as much artistic control about the clouds' appearance as possible together with requiring only a minimum amount of CPU / GPU power. I've also presented ways to create or generate and animate clouds for both of the two systems.

There is clearly no such thing as 'the best' among those systems. Which one you use depends on the purpose you need it for, the time one wants to spend in development as well as accuracy and performance requirements. All of the systems described have different qualities, all have their deficits such as not accounting for single or multiple scattering of light, demanding high performance or showing optically irritating appearances in their results. All of those methods also leave room for improvements such as using impostors, making better approximizations or making use of new developments in the hardware sector.

Last but not least, all those systems are not even exclusive. Besides a few combination that really don't make sense, there's the possibility of mixing some of the approaches described in this document to some hybrid solution on rendering clouds combining various features of both sides. And there's also aspects not covered by this document such as shafts of light and ground shadows that may also influence the choice for one system or another.

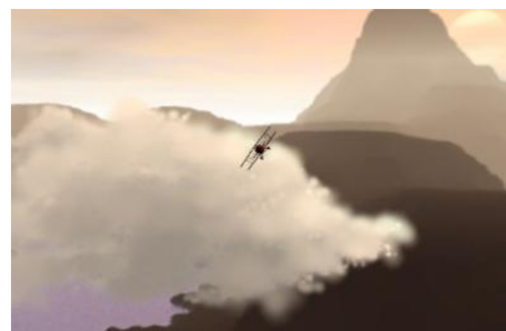
So finally it's up to everyone himself to determine which are the factors that count and which are negligible for his needs.

Screenshots

Microsoft Flight Simulator 2004

Microsoft Flight Simulator X

SkyWorks



References

Dobashi, Y., Kaneda, K., Yamashita, H., Okita, T., & Nishita, T. (2000). A Simple. Efficient Method for Realistic Animation of Clouds. *SIGGRAPH '00* (pp. 19-28). ACM Press.

Harris, M. J., & Lastra, A. (2001). Real-Time Cloud Rendering. *Eurographics 2001*, (pp. 76-84).

Harris, M. J., Baxter III, W. V., Scheuermann, T., & Lastra, A. (2003). Simulation of Cloud Dynamics on Graphics Hardware. *Graphics Hardware 2003.*

Krüger, J. (2006/07). Interactive Terrain Synthesis. *Image Synthesis.* Technische Universität München.

Nishita, T., Dobashi, Y., & Nakamae, E. (1996). Display of Clouds Taking into Account Multiple Anisotropic Scattering and Sky Light. *SIGGRAPH '96*, (pp. 379-386).

Perlin Noise. (n.d.). Retrieved 11 13, 2007, from The good-looking textured light-sourced bouncy fun smart and stretchy page: http://freespace.virgin.net/hugo.elias/models/m_perlin.htm

Perlin, K. (s.f.). *Making Noise*. Recuperado el 26 de 11 de 2007, de Noise Machine: http://www.noisemachine.com/talk1/

Westermann, P. D. (2006/07). Direct Volume Rendering. *Scientific Visualization.* München: Technische Universität München.

Wolke. (s.f.). Recuperado el 26 de 11 de 2007, de Wikipedia: http://de.wikipedia.org/wiki/Wolke