Procedural sky rendering

Eduard Zalyaev
DS2-2016
Innopolis University
Innopolis, Russia
e.zalyaev@innopolis.ru

***Abstract*—This document examines a research article that demonstrates an efficient procedural cloud generation algorithm running on graphics hardware with realtime rendering capabilities. Execution time was evaluated and possible extensions were proposed.**

I. Introduction

Generation of clouds is one of the most important tasks in rendering of realistic outdoor environments. There are simple ways to do this by using real photos of clouds. Just put a texture of a cloud on a plane and offset it over time. But there are technical limitations that make this approach impractical when the virtual worlds are getting bigger. Limited number of pictures will at some point form a clearly visible set of patterns and will show lack of dynamics. So that is why people came up with ways to do all this procedurally.

There are two categories of approaches: volumetric and planar generation. The first one tries to model the clouds in 3D space so that they are can be observed from multiple angles. Any game, where players can traverse on or above the sky level, would have to do volumetric generation in order to preserve the sense immersion. Planar type algorithms simply project generated clouds onto a flat surface. When players can't reach the skies there is no reason to waste execution time on computation of the cloud in third dimension.

II. Proposed solution

The paper [1] that we are going to examine is planar rendering algorithm that combines procedural techniques with hand crafted assets to produce impressive results. More importantly it does not take up much memory by today's standards.

We start off with two hemispheres one for the sky and the other for cloud rendering. Figure 1 represents described setup. The cloud dome is filled with plain color so that it is easier to see. Viewers camera will be placed under these domes. Transparency levels will adjust in the lower dome according to our algorithm and we will be able to see some portions of the blue dome.

In order to generate transparency values for clouds we would need four textures:

- **Real cloud photo**- a 512x512 image of the cloud.
- **Noise texture** is used to add some dynamics to the original image by blendig them together.
- **Linear gradient image** for emulation of the light that comes from the sun.
- **High frequency noise** for addition of finer details.

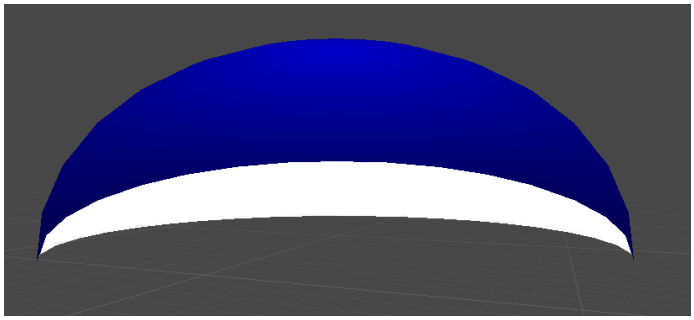
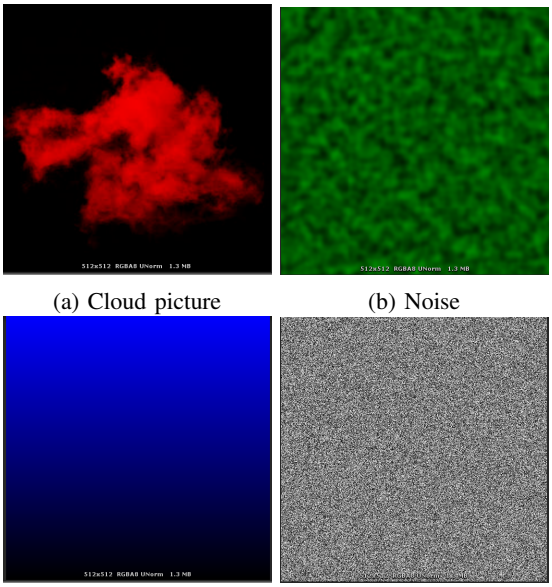


Fig. 1: Two meshes used for sky and cloud rendering.



(a) Cloud picture (b) Noise

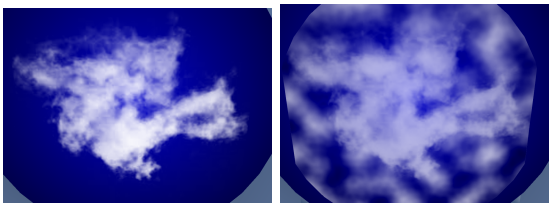
(c) Light gradient (d) Finer noise

Fig. 2: Examples of used textures.

The original image is interpolated to noise texture depending on how transparent the first one is. That way we provide irregularity without overwriting existing structure. Then we increase overall transparency according to density parameter and add finer details with interpolation to second noise texture. Nontransparent colors are then further saturated using value taken from second noise texture. In case if saturation value went out of [0,1] bounds and replaced with first noise texture range. If it sounds complicated figures included further will

provide visuals of pseudocode and result of each operation:

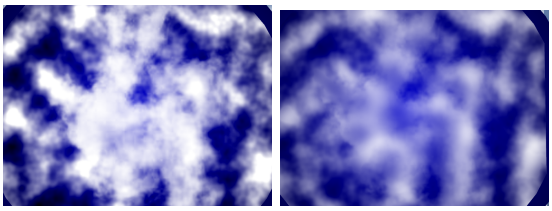
```
(for each pixel in texture space)
float cloudValue = sampleCloudImage(pixel);
float noiseValue = sampleNoiseImage(pixel);
float gradient = sampleGradientImage(pixel);
float fineNoise = sampleFineNoise(pixel);
Color c = Color(1.0,1.0,1.0,1.0);
c.alpha = cloudValue + noiseValue*(1-cloudValue);
...
```



(a) Before (b) After

Fig. 3: Interpolation to noise

```
...
c.alpha = c.alpha + fineNoise * (1 - c.alpha);
c.alpha *= interpolate(fineNoise, 1, density_parameter);
...
```



(a) Before (b) After

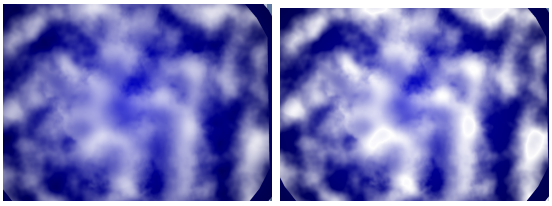
Fig. 4: Lower frequency noise refining

```
...
if (c.alpha  0)
{
        c.alpha += (density_parameter * c.alpha * 2);
        if (c.alpha  1.0)
                c.alpha = 1 - (c.alpha - 1) * 0.5;
}
c.alpha = clamp(c.alpha, 0.0, 1.0);
...
```

After we finished cloud refining we can add red gradient according to the time varying sun parameters.

III. Extension

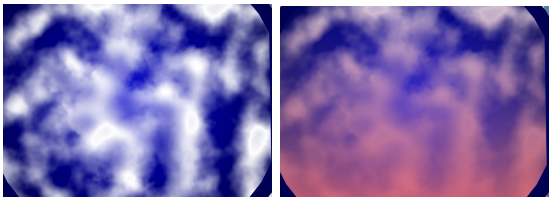
In case if we want to generate a higher resolution image of a cloud we would have to increase the amount of data that we pass to GPU by a factor of 4 since we use 4 textures. This can be avoided by using texture packing. Since the calculation



(a) Before (b) After

Fig. 5: Lower frequency noise saturation

```
...
color light = color(1.0,0.4,0.4,1.0);
light.alpha *= gradient * sunset_parameter;
c = c * (1 - light.alpha) + light * light.alpha;
c *= sun_brightness_parameter;
return c;
```



(a) Before (b) After

Fig. 6: Sunset gradient addition

are mostly transparency related we can use channels of a single texture. That gives us exactly 4 channels that we can easily extract using single sample of packed texture. Original paper [1] didn't propose any animation implementation. We can efficiently do this by offsetting the texture coordinates of our two noise images and the real cloud image over time. Doing so with different velocities provides desirable illusion. However all three textures have to be seamless and of high resolution. Otherwise repeating patterns will be more visible when observed for long enough.

IV. Evaluation

The algorithm works well as the paper stated. 1600x1200 tops at 100 fps which can be explained by difference in hardware. Increasing it to 1920x1080 slows it down a bit with 70fps but the results are still decent. We can also examine texture packing extension. For testing we scaled our textures to 1024x1024 resolution and tested our setup with and without texture packing. Here are average frame rates:

	Packed	4 textures
512x512	70 fps	50 fps
1024x1024	68 fps	45 fps

V. Conclusion

Described method is very efficient, scalable to higher resolution textures and provides impressive visual results. It can be easily applied to environments that have strict hardware constraints. Possible extensions can be easily applied given more sophisticated noise generation algorithms.

REFERENCES

[1] Ian Parberry, Timothy Roden Clouds and stars: efficient real-time procedural sky rendering using 3D hardware In *Proceedings of the International Conference on Advances in Computer Entertainment Technology, 2005*