

CloudyDay:

Rendering of clouds, atmosphere and light shafts in HDR for testing Computer Vision (CV) algorithms

Michael Beham¹, Vienna University of Technology and AIT - Austrian Institute of Technology



Figure 1: CloudyDay: Rendering of 3D clouds (stratus, cumulus clouds at center), 2D clouds (stratocumulus clouds at top) atmosphere, terrain, and airplanes to test Computer Vision (CV) algorithms.

Abstract – Rendering of clouds, atmosphere, and other natural phenomenon is an important topic in computer graphics. In this technical report, we present a novel solution, which uses different techniques to generate a realistic representation of the sky. We present a billboard-based approach to create clouds. We use half-angle slicing to generate volumetric shadows. The resulting shadow map is then used for casting shadows on the terrain, the clouds, and other objects. We also use and compare different atmosphere models and providing light shafts. Furthermore, *CloudyDay* provides HDR mapping, a bloom effect, colour grading as well as some natural phenomenon like rain.

We develop *CloudyDay* to test an autonomous flying robot. We present several enhancements, which consider the specific requirements of this specific application area. All objects can be created by an artist. This is great workflow, if a specific test-case should be created. However, creating a lot of different variations of an object is a time-consuming task. A more reasonable way is to create the shapes with procedural modelling. This technique enables to create objects (in this paper clouds, atmosphere,...) and vary the representation by varying the parameters.

1. Introduction

Rendering of the sky, with atmosphere, clouds, and light shafts, is a very important topic in computer graphic. In this technical report, we present our sky solution as well as our archived results.

Our application is created to test an autonomous flying object. Such autonomous object uses

Computer Vision (CV) algorithms to detect other objects, like airplane or mountains. Typically, a dataset with different test-scenarios is created to test CV algorithms. E.g., different lightning scenarios, different objects,... Testing of Computer Vision (CV) application in real-life is a very time-intensive, and extensive task. Instead, computer-generated simulation of test-cases enables to execute a lot of different test-cases fast. In this report, we do not only consider realistic drawing of the sky, but the special requirements of testing CV algorithms.

¹eMail: e0726417@student.tuwien.ac.at
Matriculation number: 0726417

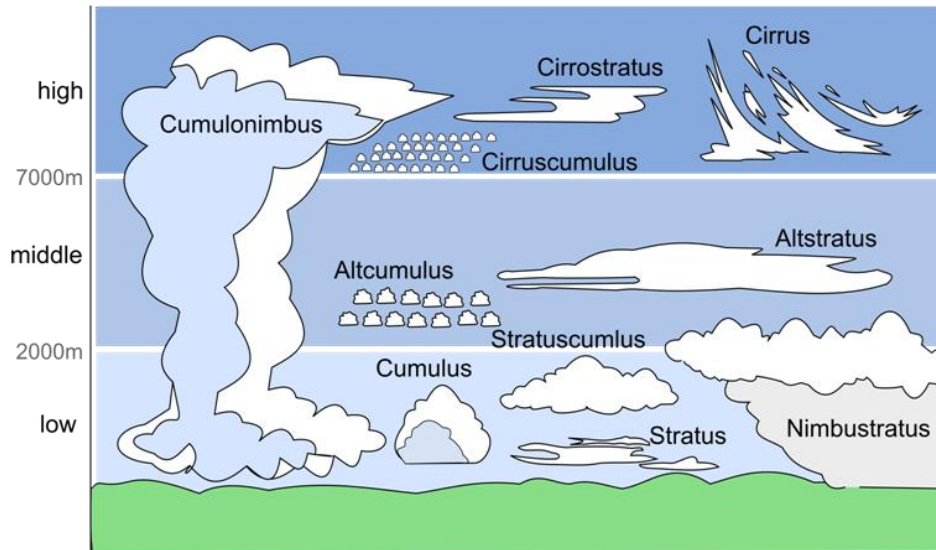


Figure 2: Overview of cloud types [WCloud14]

In the next sections, we give some background information about clouds and testing of CV applications. The last section of this chapter gives an overview of the requirements of *CloudyDay* as well as an overview of the main contributions. In chapter 2, related work is presented. An overview of our *CloudyDay* is given in chapter 3. Then, we describe the used algorithms in detail. First, we present two approaches for rendering the atmosphere. Then, the clouds are in focus of this report and natural phenomenon, like rain. In chapter 6, we describe our post-processing effects. In chapter 7 details of our implementation are given. Our archived results are discussed in chapter 8. The last chapter gives a summary of this technical report.

1.1 Background: Testing of Computer Vision applications

In the recent times, applications using Computer Vision (CV) find the way outwards of science laboratories and take place in fabrics, sky and the street. Cars are equipped with advanced driver assistance systems, drones observe public places to ensure the public safety, and robots are used for quality control in fabrics. Computer Vision applications act with the real world. A save operation mode is very important to prevent accidents and false working of the robots. So, testing of the CV systems is important. If every test case has to perform in the real world, this task is very expensive and time intensive. An obvious idea is to use computer generated simulations to reduce the amount of time to test and the costs.

To test a computer vision (CV) system, a computer generated scene is created. This scene builds up of different geometric shapes, which illustrate the real world [Zendel13]. Optionally, the shapes are

animated. The objects can be created by an artist. However, creating a lot of different variations of an object is a time-consuming task. To simulate a specific test case (e.g., the cloud should look like a specific object), artist generated modelling is needed. However, in a lot of test cases only some constraints have to be

fulfil. Then, a more reasonable way is to create the shapes with procedural modelling. This technique enables to create objects (in this paper cloud, atmosphere,...) and vary the representation by varying its parameters (for creating different objects for the same test-case).

Then, a test case can be performed using the generated scene. Generating each test case by hand is time intensive. To speed up this process, a test case simulator is used. Such generator samples the parameter space of the used objects and generate different test-cases (e.g., the same scene with different clouds). The computer vision algorithms have to classify the objects, e.g., all airplanes. Then, all correct classifications are counted (true/positive = airplane detected as airplane, false/negative = no airplane is detected by another object) as well as all false classifications (false/positive = airplane is not detected, a true / negative = another object is detected as an airplane).

In this report, we focus on the requirements of rendering CV test cases. More information of testing computer vision algorithm are given by Zendel et al [Zendel13].

1.2 Background: Clouds

In meteorology, a cloud is an accumulation of liquid droplets or frozen crystals in the atmosphere [WCloud14]. The literature differs among ten types of clouds, depending of the appearance (see figure 2).

Stratus clouds: These clouds are flat, hazy, and featureless. They occur at low altitude, and their colour varies between dark gray to nearly white [WCloud14]. Stratus clouds can result in rain.

Cumulus clouds: Cumulus clouds are very „puffy“ in appearance. They have a flat base, and occur at low altitude. Cumulus clouds are white [WCloud14].

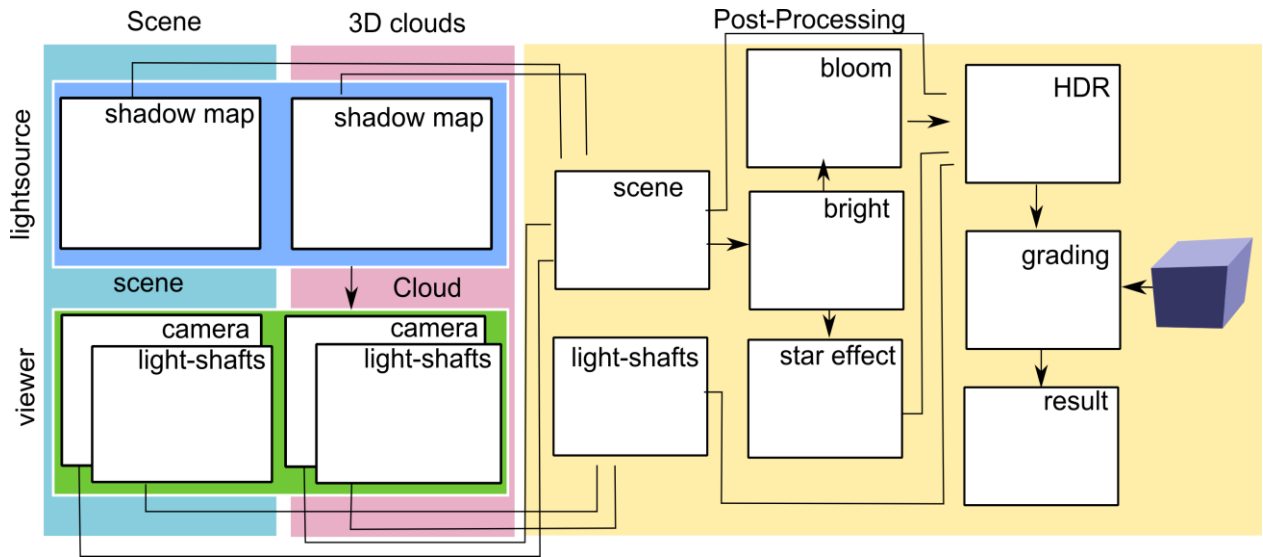


Figure 3: Overview of CloudyDay

Stratocumulus clouds: Similar to cumulus clouds, but large dark, rounded masses. Instead of cumulus, stratocumulus clouds usually occur in groups, lines, or waves [WCloud14]. They occur at low altitude.

Alto cumulus clouds: Alto cumulus clouds are puffy" in appearance. Alto cumulus clouds are white. In difference to cumulus clouds, they are located at low altitude [WCloud14].

Altostratus clouds: Similar to stratus clouds, generally uniform gray to bluish-gray sheet or layer, lighter in colour than nimbostratus and darker than high cirrostratus. They are located at middle altitude [WCloud14].

Nimbostratus clouds: This cloud type has a considerably vertical and horizontal extent. They clouds distribute over a wide area. They are located in low-to-middle altitude [WCloud14]. They are associated with rain.

Cirrus clouds: Cirrus clouds appear in thin, and wispy strands. They are white or light gray in colour, and are located in high altitude clouds [WCloud14]

Cirrocumulus clouds: These clouds are high, thin. Cirrocumulus clouds are white. Looks similar to altocumulus clouds, but they are located at high altitude [WCloud14].

Cirrostratus clouds: Cirrostratus clouds are very thin. They are similar to altostratus clouds, but are located at high altitude [WCloud14].

Cumulonimbus clouds: Cumulonimbus clouds are dense towering vertical clouds. They form alone in clusters, or along cold front squall lines. They are associated with thunderstorms and atmospheric instability [WCloud14].

1.3 Requirements and Contribution

This application is created to test an autonomous flying robot. We want to test different representations of clouds at different times of a day. Furthermore, rain, light shafts and other natural phenomenon should be supported. In summary, our application has to fulfil the following requirements:

1. **Realistic rendering of low and middle located clouds:** The view position is always lower than 4000m. The realistic rendering of the clouds is important as well as the realistic in-cloud experience.
2. **Controlling the algorithms with parameters:** To create many different test cases, the algorithms should be control-able with parameters.
3. **Controlling the algorithms by an artist:** To create a specific test-case an artist should also be able to create a specific result.

According to these requirements, we develop *CloudyDay*. Instead of developing new techniques from the scratch, we use existing approaches and combine them in a novel way. The new combination of well-suited techniques requires to adapt each to ensure a realistic experience. The main contributions of our work are:

- **Clouds:** Realistic rendering of different cloud-types with volumetric shadows.
- **Atmosphere model:** We provide two atmosphere models and use it for shading of the clouds, and other objects.

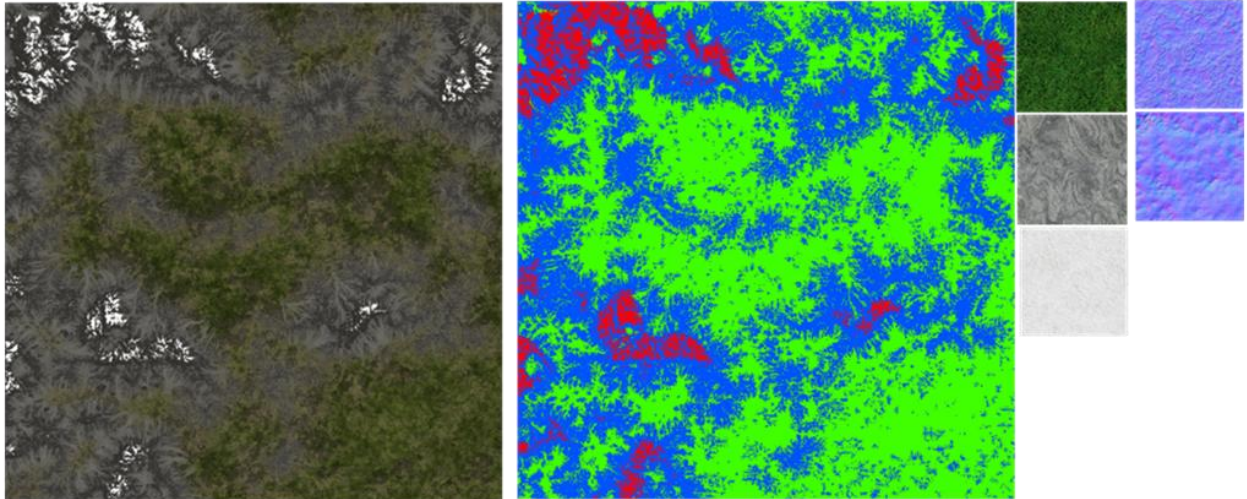


Figure 4: Textures to generate the terrain

- **Testing of Computer Vision:** The objects and effects are created to test computer vision (CV) applications. The scene should be controlled using parameters. However, artist-controlled techniques should also be supported.

2. Related Work

Our work based on a lot of other work. In this section the most sophisticated works of rendering clouds, and atmosphere, are presented.

Clouds: Rendering of clouds is an important topic in realistic rendering of natural objects. In a lot of

computer games, clouds are only drawn using a 2D texture. This approach is adequate, if the view camera is located on the ground of the earth. In flight simulators or other application scenario, where the viewer camera is in the sky, 3D approaches are needed. In literature common approaches are the use of particles respectively billboards, 3D textures or 3D models. In following, we present some of the most sophisticated works to create clouds.

Clouds can be created using 3D models. Bouthors et al. [Bouthors08] presents a technique to represent clouds, and provides a novel-based shadow and light-scattering technique.

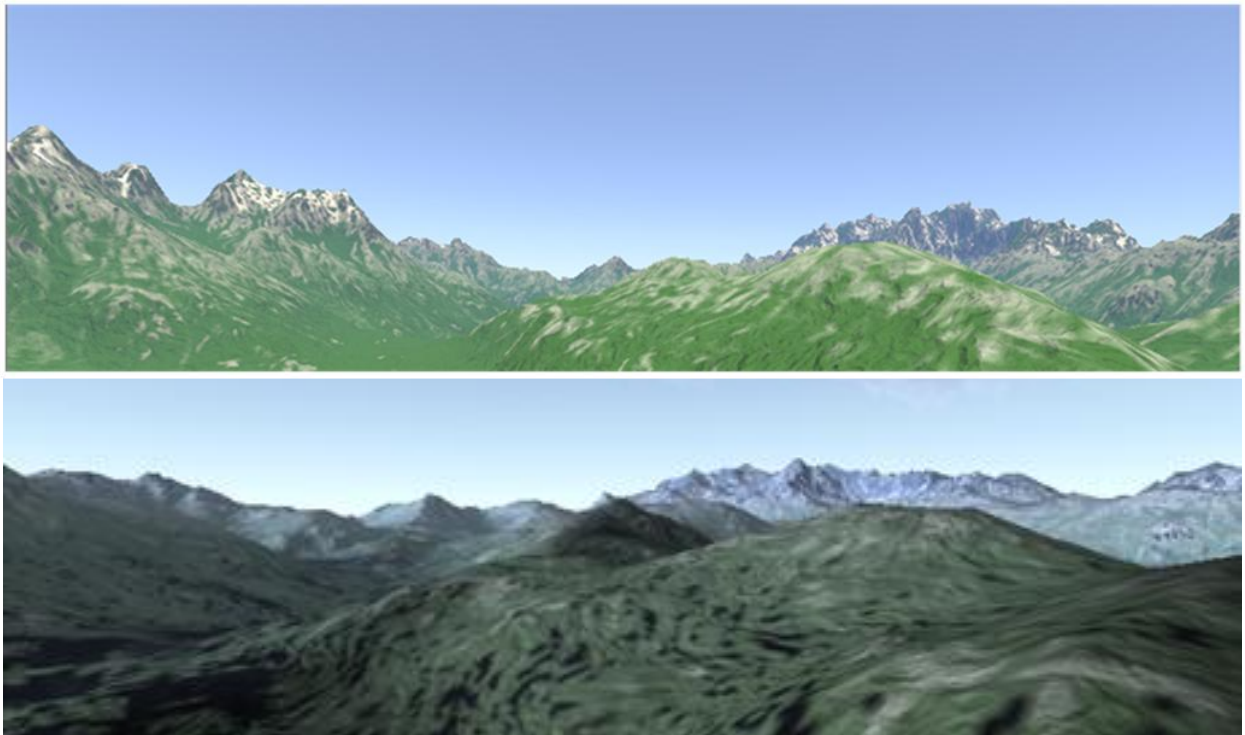


Figure 5: Result of the atmosphere algorithm: Bruneton et al. (top) and O'Neal (bottom)

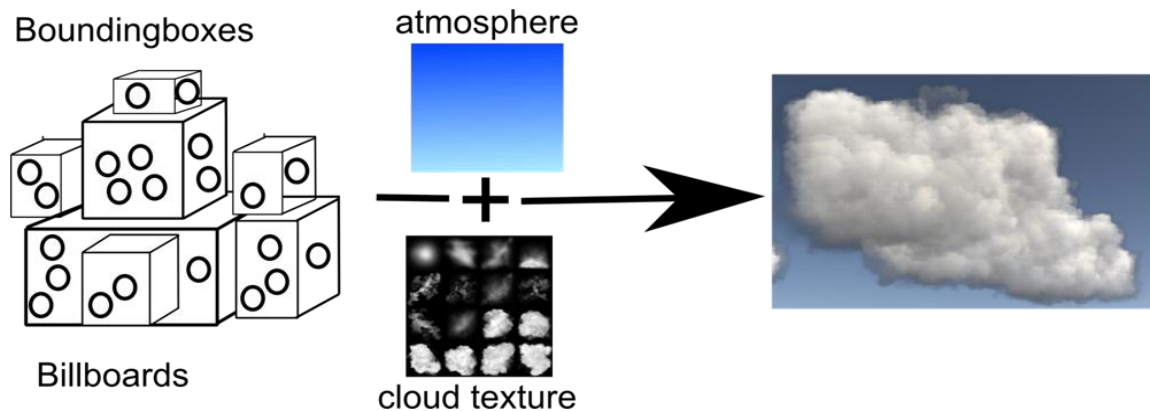


Figure 6: Cloud generation algorithm

Another popular technique to display clouds are the use of 3D textures. [Ikits04] The cloud is created using a simulation model. Then, the 3D texture is drawn using popular techniques, like slicing or raycasting [Ikits04].

Billboard techniques are very well known suited for rendering clouds. Harris et al. [Harris01] present in a framework to simulate the generation of clouds. The clouds are simulated clouds using a Navier-Stokes equation system. The resulting clouds are rendered using billboards.

Wang [Wang02] presents another technique to render clouds with billboards. Instead of simulating the creation of clouds, he presented an artist-controlled technique. The advantage using billboard techniques is a very good in-cloud experience.

Our approach based on the technique from Wang. However, we extend their technique, to integrate an atmospheric model, implement volumetric shadows using half-angle slicing, and provide shafts-of-lights.

Atmosphere: Another challenging problem is the realistic rendering of the atmosphere. To get realistic results, the incoming light, single and multiple scattering, as well as the out coming light has to be considered [Bruneton08].

To simplify the atmosphere calculation, usually some simplifications of the atmosphere model are done. A typical simplification is to ignore the multiple light scattering, using a flat earth hypotheses, constant atmosphere density, and ignoring the Mie scattering [Bruneton08]. The flat Earth hypothesis results in acceptable works, if the viewer is on the ground. Ignoring the multiple light scattering is acceptable, if only daylight should be simulated. The multiple light scattering is important for twilight [Bruneton08].

We are implementing the approach from O'Neal [O'N05] and Bruneton et al. [Bruneton08]. Both approaches use the Earth hypotheses and calculate the single light scattering. The O'Neal simulates the

effects by using only a very low sampling rate. The approach from Bruneton et al [Bruneton08] pre-process the effects. At runtime they use the pre-computed lookup-tables to calculate a realistic atmosphere[Bruneton08]. Bruneton et al. [Bruneton08] also consider multiple light scattering. Besides of integration of both approaches, we show, how we can integrate our cloud approach in this algorithm.

Light Shafts: Light shafts are another important effect. The light shafts results from light, which is scattering off of some particles in the media (e.g., air).

A simple technique is to generate shadow volumes [Mit04a]. Polygons are extruded from the light source. This volumes are coloured using the light colour, and they are attenuated with the distance. Then, the resulting volume is blended in the framebuffer,. However, this simple approach results in some errors between the light shafts and the scene geometry (due the z-buffering).

A method to create light shafts is to use sampling. The atmosphere is sampled to detect regions within shadow. This region is then used to generate shafts of light. An example is given by Dobashi et al. [Dobashi00]. They create a several spheres with different radius. The centre is located at the viewer's position. Then, they map the shadow texture of the clouds to the spheres. This results to light shafts, which results from clouds.

A similar approach is done by Bruneton et al. [Bruneton08] modifies the shadow volume technique to create light shafts. Then, the shadow volume is sampled to calculate light shafts.

Another method to generate light shafts is presented by Mitchell et al. [Mit07a]. They create light shafts as post-processing effect. They render the sun white and the geometry black in an additional framebuffer. Then, the light shafts are calculated using a simple blur filter. The resulting texture is blended to the scene.

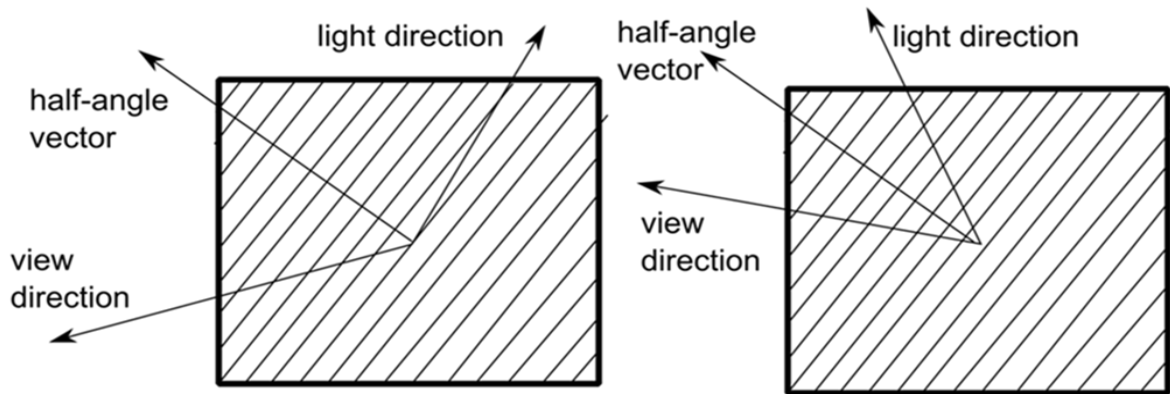


Figure 7: Rendering of 3D clouds

3. Overview of CloudyDay

According to the requirements presented in section 1.3, we develop *CloudyDay*. Our main goal is to generate a realistic representation of the sky. We also consider the integration of other objects, like terrain, and airplanes. Furthermore, we consider the specific requirements for testing Computer Vision algorithms.

We implement ten types of clouds (as presented in section 1.2). According to our requirements (see section 1.3), the lower and middle located clouds are implemented using a 3D billboard approach. This approach gives a realistic representation of the clouds as well as a good in-cloud experience. Clouds, that are located at high altitude, are only represented with a static 2D planes. This approach saves processing time. Furthermore, our solution provides a realistic rendering of the atmosphere, rain, and fog. We also integrate some post-processing effects to get a more realistic representation of the scene and to support more test cases.

An overview of *CloudyDay* is depicted in figure 3. First, the clouds are generated in an initial pre-processing step. Then, the scene, without the clouds, is rendered from the light source, and from the viewer's perspective. The rendering of the light source is needed to calculate shadows. Each object is shaded using Phong shading. For rendering of the atmosphere, we provide two approaches. In another step, the clouds are rendered from the light source and from the viewer position using our half-slicing approach. The last step is to composite the scene, and to process the post-processing effects.

In the next chapter, we present each step in detail. First, the atmosphere is presented in detail. In this chapter, we also present in detail the shading of the terrain, and other objects. In chapter 5, the creation and the rendering of the clouds is presented. Additionally, we implement a rain, and fog, which is also presented in chapter 5. In chapter 6, the post-processing effects of *CloudyDay* are in focus.

4. Atmosphere, terrains, other objects and shadowing

The realistic rendering of the atmosphere and other objects, like airplane, is very important to create realistic computer vision test-cases. In this chapter, we present the two implemented atmosphere model in the first section. Then, the shading of the terrain and other objects is in focus as well as the shadow mapping. For shadow mapping, we need an efficient implementation of the Gaussian blur. Our efficient implementation of the blur filter is in focus of the last section in this chapter.

4.1 Atmosphere Model



Figure 8: Sunlight

A very important part of *CloudyDay* is the realistic rendering of the atmosphere. To simulate the atmosphere, the light equation has to be solved [Bruneton08]:

$$L(x, v, s) = L_0 + R(L) + S(L)(x, v, s)$$

$$L_0(x, v, s) = T(x, x_0)L_{sun} \text{ or } 0$$

$$R(L)(x, v, s) = T(x, x_0)I(L)(x_0, s)$$

$$S(L)(x, v, s) = \int_{4\pi} T(x, y)I(L)(y, v, s)dy$$

where L_0 is the direct sunlight, L_{sun} attenuated before reaching x by $T(x, x_0)$. It is 0, if the sun is not visible (occluded by the terrain). $R(L)$ is the light reflected at x_0 and also attenuated before reaching x

and $S(L)$ is the inscattered light. is the This simulation is very time-intensive.

CloudyDay provides two atmosphere models. The first method implements the technique from O'Neal. Multiple scattering is ignored. The equation is reduced to $L = L_0 + R(L) + S(L)$.

Scattering is solved using low sampling and simplifying of complex functions using simpler 1D functions. More details are presented by O'Neal [O'N05].

A more realistic model is presented by Bruneton et al. [Bruneton08]. Instead of the method from O'Neal, they account the multiple scattering. This enables a more realistic sunset. They pre-compute the light-transport equation and using this technique to display the atmosphere. All details of this algorithm are presented by Bruneton et al [Bruneton08].



Figure 9: Airplane with reflection

4.2 Terrain and objects

Beside the atmosphere and clouds, terrain and other objects (like airplanes) are important. In this section we present our approach to integrate it in the application.

Terrain: The terrain is created using a height-map, bump map, and normal map. The artist can switch between a simple terrain geometry with parallax mapping, and a tessellation shader approach. The advantage of using the tessellation shader approach is, that regions near the viewer are shown in higher detail, than regions that are far away (level-of-details). Regions, which are near to the viewer, are shown with high level of detail. We use a high tessellation factor. Otherwise, the region is represented with less level of detail. We set the tessellation factor low.

Additionally, we provide several textures to shade the terrain (grass, snow, stones, and boulder textures) as well as a texture, which defines the ground. The colour, and opacity defines, if a snow, grass or stone texture should be used for texturing. An example is shown in figure 4.

The stone and boulder textures consist of a bump map and normalmap. We use the bump map to add more detail. We use these textures for displacement and parallax mapping.

The creation of the terrain using textures enables to use procedural terrain generators. The generator result in textures which can be used as heightmap and defines the regions of grass, stones, snow and lots of more. Furthermore, the textures can be created easily by the user with a image processing software (requirement 3). We will add this workflow in future work.

Objects: In the scene also other objects, like airplanes, can be rendered. Such objects are shaded with the Phong lighting model using Phong shading. All objects are textured. Additionally, we calculate for each object the atmosphere colour in the vertex shader (as presented in the previous section). In the fragment shader the atmosphere colour is mixed with the ambient and diffuse colour. For reflection CloudyDay provides creating an environment map using a cube texture.

4.3 Shadow mapping

Shadows are a very important part in the nature, especially for testing CV application. Shadows are a common source of error. E.g., the shadow of a airplane could be detected as airplane instead as shadow. Furthermore, CV application should detect objects, which are casted of a shadow.

For calculation of the shadows we use the well known variance shadow algorithm. This algorithm results in soft shadows using an approximation of the normal distribution. We prefer soft shadow algorithms because soft shadows also occur in nature. If we would implement hard shadows, some false/positive matches could be occur (i.e. they detect an shadow as airplane), because the shadows have hard edges.

However, variance shadow maps based on following inequality (Chebychev's inequality, one-tailed version):

$$P(x \geq t) \leq p_{max}(t) \equiv \frac{\sigma^2}{\sigma^2 + (t - \mu)^2},$$

where μ is the mean, σ^2 is the variance. This inequality gives us an upper bound of the Gaussian distribution. According to Donnelly [Donnelly06] we use the value $p_{max}(t)$ to approximate the true value p (resulting from the Gaussian distribution). The mean μ is calculated by using:

$$\mu = M_1$$

and the variance σ^2 is calculated by using:

$$\sigma^2 = M_2 - M_1^2,$$

where:

$$M_1 = E(x) = \int_{-\infty}^{\infty} xp(x)dx$$

$$M_2 = E(x^2) = \int_{-\infty}^{\infty} x^2p(x)dx$$

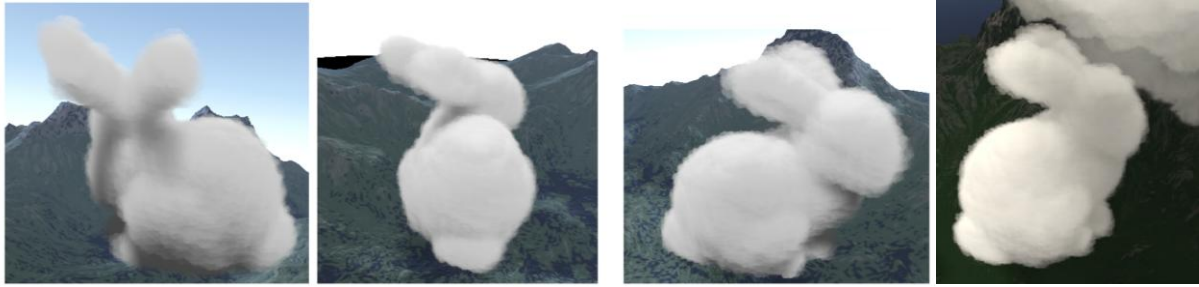


Figure 10: Stanford bunny rendered with our 3D cloud algorithm

At rendering from the light source we save the first and second moment of the distance from the light point to the closest polygon in a shadow map. The sum is then calculated using anisotropic filtering and an additional Gaussian blur filter. At rendering from the viewer camera, we use the resulting shadow mapping. If the actual fragment is in shadow, we calculate p_{max} to calculate the soft-shadow.

For clouds, we only use the standard shadow map approach, which is generated by the cloud rendering algorithm. Before shadow mapping, we also blur the shadow map using a Gaussian blur filter with size of 9.

4.4 Gaussian Blur

We implement a Gaussian blur filter using linear interpolation presented by Rakos et al. [Rakos10]. In order to get an efficient implementation of the Gaussian filter, we use following property: "The two-dimensional Gaussian filter can be implemented by multiplication of two one-dimensional Gaussian functions" [Rakos10]. So, we blur the image in the vertical direction in a first pass. The resulting image is then blurred in horizontal direction in a second pass [Rakos10].

To reduce the number of texture fetches, we take advantage of the linear interpolation. We can get the average of two pixels by using a single texture fetch. We only need to sample between both texels. The

linear interpolation gives us the mean of both texels. For generating a filter of size 2, we only need $N/2$ samples [Rakos10].

The Gaussian blur filter is used to blur the shadow map. We also use this filter to generate a bloom effect. An adapted approach of this blur filter is also used to create a star effect and the light scattering.

5. Clouds

The clouds are a main contribution of this technical report. *CloudyDay* supports ten types of clouds, which are created using a 3D billboard-based approach or with the simpler 2D approach.

In the next sections, we describe our 3D billboard-based approach in detail. First, we present four techniques to create clouds. Then, the rendering of the 3D clouds is presented in detail. Then, our 2D approach is in focus to create high altitude clouds like cirrus. In the last section, we describe our rain, and fog implementation.

5.1 Generation of 3D Clouds

In the first step, the clouds are generated. The artist defines the type of the cloud, the location within the scene, and the generation algorithm. In this section, we present our four techniques to generate clouds.

Creation using Wang approach: Our solution of rendering clouds based on the work of Wang [Wang01]. Wang presents an artist-controlled



Figure 11: Shadow at cumulous clouds and in-cloud experience of a stratus cloud.

technique to generate 3D clouds. *CloudyDay* also support their approach.

The artist defines bounding boxes of the cloud using an arbitrary 3D modelling software. Then, *CloudyDay* loads the bounding boxes, and distributes billboards within the bounding boxes. Each billboard is defined by the size, the position, the opacity, the corresponding bounding box and the texture to display. After sampling the billboards within the bounding boxes, we calculate for each billboard the nearest neighbour. If the distance is smaller than an artist-defined threshold, the billboard is deleted. This operation saves processing time and ensures, that the billboards are distributed over the bounding boxes more uniform. In figure 6, an overview of the cloud algorithm is shown.

Creation using random sequences: Additionally, *Cloudy-Day* supports to create the clouds automatically. We use a similar approach to Wang, but we generate the bounding boxes automatically using a random generator.

First, we create a bounding box. The size is selected randomly. The first box is placed at the bottom of the cloud. Then, the next bounding box is placed aside of the previous generated bounding box. For this task, we calculate two random generated angles, which defines the joint position of the bounding box. We repeat this process until the cloud is created. After generation of the bounding boxes, we use the same approach, presented above, to generate the 3D cloud.

This generation algorithm can be used to create a cloud layer. An artist defines the size of the layer, as well as the number of cloud clusters, the density of the cloud layer, and the cloud type. The cloud clusters are distributed over the layer area randomly. We use for this task a Sobel sequence, because low discrepancy sequences ensure a uniform distribution of the clots. First, we create some clouds to form the cloud clusters. Then, we measure the cloud density in the cloud layer. If the cloud density is lower as an artist-defined threshold, we add some clouds to each cloud cluster. We repeat this process until the threshold is larger than the defined cloud layer density.

Parameter of Layer	Type	Min	Max
Type	Integer	0	6
Middle point	Vector	-Inf	Inf
Size	Vector	-Inf	Inf
Overcast	Float	0	1
Number of clots	Integer	1	Inf
Variance	Float	0	1

Colour	Vector	0	Inf
Opacity	Float	0	Inf

Table 1: Parameters of 3D clouds

Creation using 3d models: Besides the random generation of clouds, the application also supports to generate clouds using a 3D model. This feature is important for testing a specific representation of a cloud (e.g., cloud looks similar to an airplane). The user can easily generate clouds easily by using a watertight 3D model. Then, the clouds are generated automatically to test the classification between airplanes and clouds.

Our approach enables to generate a 3D cloud by using a watertight 3D model. For this task, the application voxelize the watertight 3D model. We use the approach from Crane et al [Crane07]. First we calculate the AABB bounding box of the 3D model. Then, each triangle of the shape is rendered into a slice of a 3D~texture using an orthogonal projection. We set the near plane of the orthogonal camera to match with the current slice. The far plane is set to infinity. Additionally, a stencil buffer is initialized to zero. Then, the shape is rendered. The stencil buffer is incremented, if the triangle passes the back face test. Otherwise, the stencil buffer is decrement (i.e., if the back face test is failed). After rendering the watertight 3D model, a voxel is created for each nonzero entry of the stencil buffer. This process is repeated for each slice of the 3D texture.

We create the clouds as presented earlier. After vocalization each voxel is used as a bounding box, and we sample billboards within each box. The user can define the cloud type (e.g., cirrus, stratus,...). The user can also define the resolution of the 3D texture. We use a resolution of 16^3 .

Simulation of the clouds using cellular automat: *Cloudy-Day* also enables to simulate the clouds using cellular automate. We implement the approach from Dobashi et al. [Dobashi00], which based on the work of Nigel. This approach provides following features:

- Growing of the clouds
- Extinction of the clouds
- Wind effects
- Controlling cloud motion using ellipsoids

According to Dobashi et al. [Dobashi00], we use a cellular automaton to simplify the cloud dynamics. The simulation space is divided into voxels. For each cell three values are assigned, which determines the actual status of the cellular automaton. We use following attributes:

- **humidity value:** The humidity determines, if enough vapour is available to form a cloud.



Figure 12: Stratocumulus clouds with (b) and without (a) light scattering

- **activation value:** The activation factor determines, if a phase transition occurs.
- **clouds value:** The cloud value determines the cloud particles

The growing of the clouds is simulated as follows:

$$\begin{aligned} hum(p, t + 1) &= hum(p, t) \& \& !act(p, t) \\ cld(p, t + 1) &= cld(p, t) || act(p, t) \\ act(p, t + 1) &= !act(p, t) \& \& hum(p, t) \& \& f(p) \end{aligned}$$

where

$$\begin{aligned} f(p) &= act(p + (0,0,1), t) || act(p - (0,0,1), t) \\ &|| act(p + (1,0,0), t) || act(p - (1,0,0), t) \\ &|| act(p + (0,1,0), t) || act(p - (0,1,0), t) \\ &|| act(p + (0,0,2), t) || act(p - (0,0,2), t) \\ &|| act(p + (2,0,0), t) || act(p - (2,0,0), t) \\ &|| act(p + (0,2,0), t) || act(p - (0,2,0), t) \end{aligned}$$

Then, *CloudyDay* simulates the extinction and regeneration of the cloud as follows:

$$\begin{aligned} cld(p, t + 1) &= cld(p, t) \& \& IS(rnd > pext(p, t)) \\ hum(p, t + 1) &= hum(p, t) || IS(rnd < phum(p, t)) \\ act(p, t + 1) &= act(p, t) || IS(rnd < pact(p, t)) \end{aligned}$$

The wind is calculated as follows:

$$\begin{aligned} hum(p, t + 1) &= hum(p - v, t), \text{ if } p.x - v.x > 0, \\ &\text{otherwise } 0 \\ cld(p, t + 1) &= cld(p - v, t), \text{ if } p.x - v.x > 0, \\ &\text{otherwise } 0 \\ act(p, t + 1) &= act(p - v, t), \text{ if } p.x - v.x > 0, \\ &\text{otherwise } 0 \end{aligned}$$

The cloud motion is controlled using ellipsoids. Each ellipsoid is defined by the position, radius, and lifetime. At each iteration of the simulation, we move the ellipsoid according a wind vector. Furthermore, the lifetime is decreased. If the lifetime lower than 0, than the ellipsoid is deleted and a new ellipsoid is

generated. At each iteration of the simulation, we update the probabilities *pact*, *phum*, and *pext*. *pact* and *phum* are set high in the centre of the ellipsoid and low at the border. *phum* is set high at the border of the ellipsoid and low at the centre of the ellipsoid. *pext*, *pact* and *phum* is set to zero, if it is outside of all ellipsoids.

After simulation of the clouds, we use a splatting algorithm to generate the 3D clouds. Each billboard is displayed with eight billboards. Before sampling the volume, we are smooth the it with linear interpolation.



Figure 13: Altocumulus clouds generated with cellular automaton

5.2 Rendering of 3D Clouds

After creation, the clouds are rendered. We use a similar approach to Wang [Wang01]. The clouds consists of a set of billboards, which are grouped within a 3D bounding box. To create different cloud types, we mix 16 cloud textures (as presented by Wang [Wang01]). The textures give a realistic look of the cloud. *CloudyDay* also provides volumetric shadows. Furthermore, we integrate the atmosphere model and improve the shading. In following, all modifications of the Wang [Wang01] approach are presented in detail.



Figure 14: 2D clouds (altocumulus) with parallax mapping (left) and with displacement mapping (right)

Half-angle-slicing: To generated volumetric shadows, we adapt the half-angle slicing algorithm for our proposes. Half-angle slicing is well known algorithm to generate volumetric shadows for 3D volumes and based on the slicing algorithm.

Slicing is a technique to display a 3D texture by using 2D planes, texturing and blending. Each slice shows a layer of the 3D texture. Then, the slices are sorted after their depth. Additionally, the slices are blended using the front-to-back (or back-to-front operator, depending on the rendering direction of the slices).

Half-angle slicing uses the duality between the front-to-back, and back-to-front operator to calculate the volumetric shadows. First, the angle between the viewer and the light position is calculated (the half-angle vector). If the angle is lower than 90 degree, than the mean vector of the light direction and viewing direction is calculated. Otherwise, the mean of the inverted viewing direction and light direction is calculated. Then, all slices are created. Each slice is oriented to the half-angle vector. To display the volumetric shadows each slice is drawn two times. First, the slice is drawn from the viewer position. If the viewing direction is inverted, we use the back-to-front operator, otherwise the front to back operator. Then, the same slice is drawn from the light source position using the front-to-back operator. Then, the same slice is drawn from the light perspective. This approach is depicted in figure 7. While drawing the viewing perspective, the user can read the shadow

texture and we use it to create volumetric shadows. We repeat this process until all slices are processed.

Instead of displaying a 3D texture, we display the clouds using billboards. Then, we sort the billboards after their depth. We group the billboards with similar depth. Each group of billboard is used instead of the slices. Then, we render each group of billboards as presented above. In opposite to the original half-angle approach, the billboards are oriented to the viewer instead of the half-angle direction. This orientation prevents the typical slicing error.

Shading: At rendering the clouds in the viewer framebuffer, each billboard is shaded. At first, we map for each pixel a texture, showing a part of the cloud. Then, Wang uses for shading of clouds following equation [Wang01]:

$$C = C_{amb} + C_{diff}$$

The ambient colour is set by an artist, which depends by the height of the actual fragment. The diffuse colour is calculated by the vector from the vertex position to the middle point of the bounding box.

In our application, we use the same diffuse part. The set of the billboards within a bounding box are grouped to a clump of the cloud. We calculate the directional component of shading for a given vertex in the cloud by first computing the vector to that point from the shading group center. We also find the

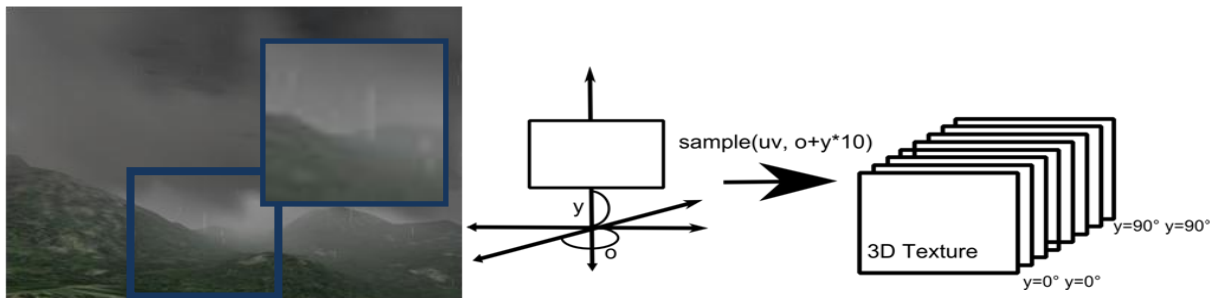


Figure 15: Our rain approach

vector from the group center to the sun, and compute the dot product of the two vectors after normalization.

However, instead of varying the ambient colour to simulate the shadows, we use a constant colour. We calculate the shadows using half-angle slicing as presented earlier.

Furthermore, we add the atmosphere colour to the cloud. Depending on the atmosphere approach, we use different shading equations. For the approach of Bruneton [Bruneton08], we use following formula:

$$C = C_{amb} + C_{diff} + I_{scatter} + C * I_{extinction}$$

,where is C the atmosphere colour, $I_{scatter}$ is the scattering factor, $I_{extinction}$ is the extinction factor, C_{amb} is an ambient term, C_{diff} is the diffuse term of Wang.

For using the atmosphere model from O'Neal, we adapt the shading as follows:

$$C = C_{amb} + C_{diff} + C_{atmsphere}$$

,where $C_{atmsphere}$ is the atmosphere colour.

After shading the cloud, the volumetric shadows are added. We sample the shadowing map and multiply the final colour with the sampled value. The billboard are blended.

Scattering of Light: Our shadow algorithm is also used to simulated light scattering. At each iteration of the cloud shadow, the shadow map is blurred. We use a simple Gaussian filter with size 3. More information of Gaussian blurring are presented in section 4.3.

Shadows at other objects: The volumetric shadows results in a shadow map, which saves the opacity. To cast shadows resulting from the clouds on the terrain and other objects, we adapt the Variance Shadow Map. After drawing the clouds, we create a depth shadow map as well as for the shadow map as presented in section. Later in shadowing, we multiply the opacity of the cloud to the result of shadow mapping.

5.3 Rendering and Creation of 2D clouds

The 3D clouds are used to for rendering stratus, cumulus, stratocumulus, nimbostratus, and cumulonimbus clouds. These types of clouds are located at low and middle altitude. Instead, cirrus, cirrostratus and cirrocumulus are located in high regions of the sky. As presented in section 1.3, the camera is always lower than clouds, which are located at high altitude (requirement 1). To save processing time, we use a simpler 2D approach.

First, we create a 2D plane to display the texture. A fat plane would look unrealistic, especially, if the view direction is parallel to the billboard. Instead, we use

quadratic function with a saddle-point to achieve a more realistic look. The plane is textured with a cloud texture. The texture shows an image of the cloud layer.

Optionally, *CloudyDay* provides an algorithm to generate altocumulus and cirrus-cumulus clouds with an Perlin noise generator. We use the Perlin noise using following function:

$$C = 1 - s^{p-c}$$

where s is the sharpness factor (we set s to 0.95), c determines the coverage (we set c to 0.2), and p is the Perlin noise. An overview of all parameters is given in table 2.

Parameter of 2D Cloud	Type	Min	Max
Type	Integer	0	6
Middlepoint	Vector	-Inf	Inf
Size	Vector	-Inf	Inf
Shaprness (optional)	Float	0	1
Cover (optional)	Float	0	1

Table 2: Parameters of 2D clouds

For shading, we use a parallax mapping algorithm [Kaneko01]. The 2D texture defines the opacity and the height of the cloud. Additionally, we use a normal map for shading. We create it with a Sobel filter. The Sobel filter creates the gradient in the x and y direction. Then, we calculate the normal using the cross product of the two gradients.

Similar to the shading of terrains, *CloudyDay* also supports the shading using displacement shading. The alpha value is used as heightfield and the normal texture defines the normals.

The advantage of this approach is, that the processing time is reduced. Another advantage using the 2D representation is that cirrus, and cirrocumulus are difficult to represent with our 3D algorithm. Our billboard approach works is more well suited for big voluminous clouds. Small, thin clouds result in errors by changing the viewer position [Wang01].

5.4 Rain

CloudyDay also provides rain. Rain occurs usually with stratus, cumulonimbus, and nimbostratus clouds (see section 1.2).

For creating the rain, we use a simple particle system. We use following equation:

$$x_{i+1} = x_i + w * v_i * f$$

where x is the raindrop position, v is the velocity of a raindrop (set to (0,-9.81,0)), w is a weighting factor and f the force. The lifetime of each particle is updated as follows:

$$l_{i+1} = l_i - dl$$

where l is the lifetime, and dl is a decrease factor.

At rendering, each particle is drawn with alpha blending. Additionally, we add a 3D texture to each particle. Each slice of the 3D texture represents a raindrop from a certain viewing direction [Kshitiz06]. This technique ensures a correct representation of each raindrop from different viewing directions. First, we measure the orientation from the viewer to each particle. Then, we display the most similar raindrop representation. In figure 16, this algorithm is

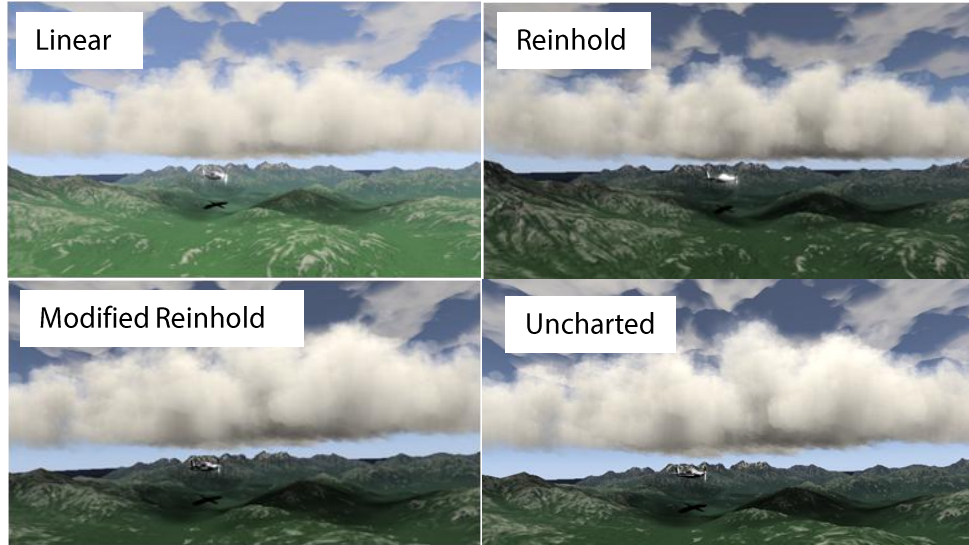


Figure 16: Results of different HDR mapping algorithms

illustrated in detail. In table 3, an overview of the parameters of the rain implementation is given.

Parameter:	Type	Min	Max
Density	Integer	0	1
Size	Float	0	Inf
Middle point of Region	Vector	-Inf	Inf
Size of Region	Vector	-Inf	Inf
Velocity	Vector	-Inf	Inf
Lifetime	Float	0	1

Table 3: Parameters of our rain implementation

5.5 Fog

Rain usually occurs, if water condensates. Beside of rain, also some fog occurs. To display fog, *CloudyDay* provides a simple fog algorithm. We use following equation:

$$F(z) = e^{-z \cdot k_2} k_1$$

where k_1 and k_2 are two user-defined constants. We implement the equation in the fragment shader of

the atmosphere, terrain and other objects. $F(z)$ is used to mix the fog colour with the scene colour.

Using the fog over the entire scene is not always desirable. E.g., the airplane can overfly such regions. In such scenarios, an artist can use the 3D stratus clouds. This technique enables to draw fog at a specific region within the scene.

6. Post-processing

In the post processing step, the scene is composited, and the light shafts are calculated. Furthermore, we

calculate bloom effects and a star effect. *CloudyDay* also provides techniques for HDR mapping and colour grading.

The post-processing effects enable many different test-cases for testing of CV algorithm. We can test the influence of shafts of lights, and test different

mappings of the high dynamic range to the low dynamic device colour-space. Each of the effects are difficult for object recognition algorithms. In the following sections, we describe each step in detail.

6.1 Light shafts

The light shafts are created as a simple post-processing effect presented by *Kenny Mitchell*. We use an additional framebuffer attachment at rendering from the viewer camera. We draw the sun white, and all other objects black. For drawing the rain and clouds, we use blending (black colour with the corresponding alpha value) additionally. After drawing all objects, the light shafts are calculated using following equation:

$$L(s, O, o) = exposure \sum_{i=0}^n decay^i w \frac{L(s_i, O_i)}{n}$$

where the exposure controls the overall intensity, the weight factor w controls the intensity of each sample, and decay dissipates each samples contribution as the ray process away from the light source. n are the number of samples [Mit04a]. This algorithm results in shafts of lights, if the viewer looks to the sun. An example is shown in figure 17.



Figure 17: Sunset and cumulus clouds

6.2 Bloom

In addition to the light shafts, *CloudyDay* provides a simple bloom implementation. The bloom effect is a very important effect for testing of computer vision algorithms. A common test case is to distinct between airplanes and other objects. In bottom, several objects have a very high reflectance. This reflection can be seen from the airplane. Such scenarios can be produce false-positive recognition errors (e.g., a car is detected as airplane).

In the first step, we check for each pixel in the viewing buffer, if the intensity is higher than an artist-defined threshold. If the intensity is higher as the threshold, we subtract the threshold from the intensity and saves the resulting intensity in an own texture. Otherwise, we set in the texture 0. Then, the resulting texture is blurred using a Gaussian filter. The last step is to add the texture to the final image.

6.3 Star effect

Another important effect are stars, which results from reflections. The stars are caused by microscopic bumps and scratches in the camera optic, which produce internal reflections and refractions [Luksch07].

To achieve this effect we use a special blur filter. We blur the image, resulting from the threshold (presented in the previous section), in horizontal and vertical direction. Both resulting textures are saved. Then, we take the maxima from both texture and add it to the final image. This operation results into a star like grow.

To achieve bigger stars, Luksch [Luksch07] performs this filter several times. Using the same filter several times is time-intensive. Instead, we down-sample the bright image and we perform these operations for each size. The resulting textures are than merged into one texture, which is added to the final image. The advantage of this approach is, that it can be implemented fast. We can use the result of the bloom effect after horizontal blurring.

6.4 HDR mapping

After calculation of the star effect and the bloom effect, we map the high dynamic range of the scene to the lower dynamic range of the device colour space. The mapping is called HDR mapping.

A good mapping between the high dynamic range to the lower dynamic device range is very important. In some test cases, the robot is below the clouds. Then,

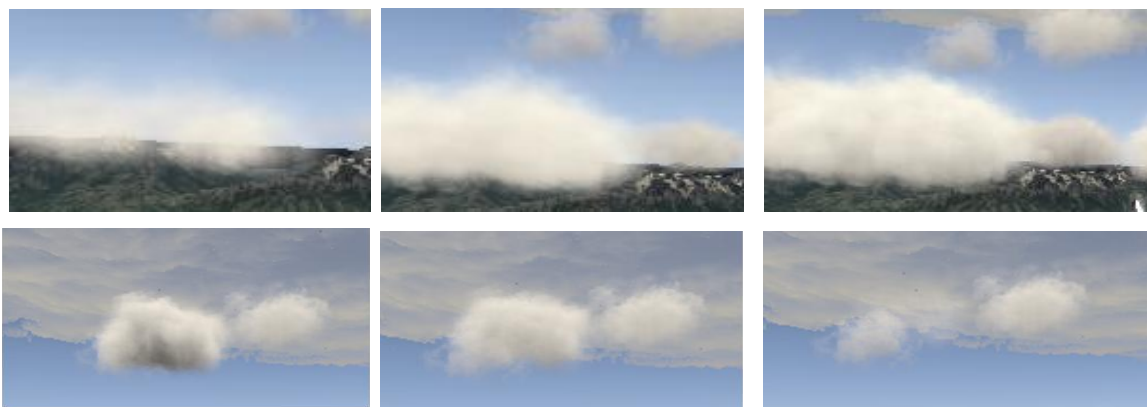


Figure 18: Growing (top) and shrinking (bottom) of clouds

the scene is dark (due the shadows). If the robot is above the clouds, than the scene is very bright. If we do not provide a good mapping, we lost a lot of details of the clouds. The low dynamic range of the device colour space is small.

We implement a similar approach to Luksch [Luksch07] CloudyDay support four HDR mapping algorithms:

- **Linear mapping:** $L_{new} = \frac{aL_w}{\bar{L}_w}$
- **Reinhard's operator:** $L_{new} = \frac{L_{scaled}}{1+L_{scaled}}$
- **Modified Reinhard's operator:**

$$L_{new} = \frac{L_{scaled} \left(1 + \frac{L_{scaled}}{\bar{L}_{white}^2} \right)}{1 + L_{scaled}}$$
- **Uncharted:** $x = C_x + C_y + C_z$

$$L_{new}(x) = \frac{x(Ax + CB) + DE}{x(Ax + B) + DF} - E/F$$

where, L_{white} is set to 2.5 and L_{scaled} is calculated as follows:

$$L_{scaled} = \frac{aL_w}{\bar{L}_w}$$

where \bar{L}_w is the average luminance and a is called key.

The average luminance \bar{L}_w is calculated using an image pyramid. First, we calculate the luminance using following equation for each fragment:

$$L_w = \begin{pmatrix} R \\ G \\ B \end{pmatrix}^T * \begin{pmatrix} 0.2126 \\ 0.7152 \\ 0.0722 \end{pmatrix}$$

The weights are also used for the YUV colour model and it covers the human perception of colour more closely than the mean. The average luminance is calculate as follows:

$$\bar{L}_w = \exp \left(\frac{1}{N} \sum \log(0.001 + L_w) \right)$$

To calculate the sum, we downsize the image. We use four passes to generate a image pyramid. The first pass starts with a size of 256x256. For each pixel, we use the average of a 4x4 pixel group. To reduce the number of samples, we use the linear interpolation of the texture. We always sample the texture at the middle point of four pixels. Linear interpolation gives us the average of the 4 neighbor pixels. Using this approach reduce the number of samples from 16 to 4. We repeat this process until we result with an image of size 1.

The key value a is calculated as follows [Luksch07]:

$$a = \max \left(0, 1.5 - \frac{1.5}{0.1 \bar{L}_w + 1} \right) + 0.1$$

The key value controls the exposure of the HDR

mapping. Optionally, the user can define a constant factor. We recommend a value between 0.1 (result in a dark image) and 0.5 (result in a bright image).

6.5 Colour grading

In the last step of the post-processing pipeline, *CloudyDay* performs colour grading. The artist can modify the colours using standard image processing software.

This technique changes the colour of each pixel using a 3D lookup table. We use a 3D texture with the size of 32 for each dimension. The texture is simulated using linear interpolation. An artist can modify the 3D lookup table to simulate some camera errors, like colour engraving. Optionally, the user can use for different objects a different 3D lookup table.

7. Implementation

Our application is implemented with C++ and OpenGL. As shader-language we use the OpenGL Shading Language (GLSL). The 3D clouds and the rain effect are implemented using geometry shaders. The terrain implementation also uses tessellation shaders to implement a view-dependend level-of-detail.

We implement our application using the open soucre rendering engine: *OpenSceneGraph (OSG)*. *OSG* is an open source 3D rendering engine. A key feature of *OpenSceneGraph* are a lot of different plug-ins. The plug-ins enable to adapt *OpenSceneGraph* to the needs of our requirements. For our requirements, we develop our own plug-in.

8. Results and Discussion

In this section we present the results of *CloudyDay*. First, we discuss our 3D cloud rendering algorithm. Then, we present our results of the 2D cloud algorithm. Besides the clouds, we also compare the atmosphere algorithms, and discuss the post-processing algorithms.

8.1 3D Clouds

The 3D clouds are the main contribution of our clouds. We adapt the approach from Wang [Wang01] to render clouds, which are located at low and middle altitude. In this section, we discuss our archived results.

Generation: We provide four cloud generation algorithms. Our implementation based on the algorithm from Wang [Wang01]. We implement their cloud generation algorithms using standard modelling applications like Blender or Maya. This approach is good for an artist-generated workflow.

Another artist controlled workflow is the use of 3D models to generate clouds. *CloudyDay* provides a voxelization algorithm to generate 3D clouds. This technique is very well suited to test the cloud

recognition of the CV application

Instead of the artist controlled techniques, *CloudyDay* provides random generated clouds. The algorithms create a layer of clouds. We use a simple algorithm to generate clouds using bounding boxes.

CloudyDay also provides a simulation model using cellular automata. The simulation model provides growing of clouds, extinction of the clouds and wind effects. The advantage of this approach is that is easy to implement. Furthermore, the simulation can be controlled by ellipsoids. The resulting cloud is always within some ellipsoids. The controlling of the clouds with ellipsoids makes it easy to perform a specific test-case.



Figure 19: Cumulus clouds from top

Shading and volumetric shadows: The half-angle slicing algorithm results in volumetric shadows. A nice example of our approach is given in figure 10. The ears of the bunny cast a shadow at the body. Instead of normal shadow mapping, the opacity of the cloud is considered. Furthermore, inner parts of the cloud are also in shadow. This is important for a realistic rendering of the in-cloud experience.

The advantage of the half-angle slicing algorithm is that it results in a realistic look with moderate processing costs. According to other billboard approaches, the billboards have to be sorted one time. However, instead of rendering the billboards in one call, multiple layers of billboards are created, which are drawn 2 times. First, we draw the billboards from the viewing perspective. Then we draw the billboards from the light perspective to create the shadow map.

The integration of atmosphere model enables a realistic representation of the clouds. The atmosphere model is used to simulate the impact of the atmosphere scattering. At the sunset, the atmosphere results in a realistic look of the clouds. At the day, they result in a bluish representation.

A big advantage of this approach is the good in-cloud experience. If the viewer flies due the clouds,

the billboards displays small details within the clouds. This representation is especially well suited for fog and Stratus clouds. An example of the in-cloud experience is figured in figure 11.

However, the volumetric shading algorithm results in some errors. An error occurs by changing between front-to-back and back-to-front shadowing. Both operators result in the equal representation. However, changing the slicing algorithm results in a significant change of the orientation of the billboards. A small, but recognizable change of the clouds can be seen. This error depends on the size of the billboards. If we would use smaller billboards, the error gets smaller.

8.2 2D Clouds

The 3D approach is not well suited for drawing cirrus clouds [Wang01]. Cirrus clouds are very big but thin. Billboards, which are oriented to the viewer, would result in an unrealistic behaviour of the clouds, if the viewer position is changed. Instead, we use a simpler 2D approach for high altitude clouds, like cirrus or cirrus cumulus. We only draw a texture of a cloud using a 2D plane. This approach is adequate, because the viewer is below the high altitude clouds. Another advantage of this approach is, that it saves processing time.

The 2D clouds are very well suited for our proposes. The camera is always lower than 4000 m (see requirement 1 in section 1.3). A realistic in-cloud experience is not needed, and only the bottom of the cloud is shown. As presented in section 5.3, we provide a parallax mapping and a displacement shading approach. The advantage of the displacement approach is, that it results in a more realistic look than the parallax mapping. The resulting clouds look more bulky. In figure 19 a comparison between parallax mapping and displacement shading is given.

In addition to the high level clouds, we implement a simple altocumulus generator with Perlin noise. The Perlin noise generator provides altocumulus and cirrocumulus clouds. A disadvantage of our implementation is, that we do not provide any generator to generate the cirrus, cirrocumulus and cirrostratus textures.

8.3 Atmosphere

The atmosphere is a very important part of our sky approach. We use the atmosphere model to create the atmosphere and to shade other objects, like terrain and airplanes, realistic. As presented in chapter 4, we implement two approaches.

O'Neal proposes a very low sampling rate (<5 Samples) for his method. However, using current graphics hardware enables to use higher sampling rate. Another simplification of O'Neals approach is to

use different shaders, depending on the objects position. O'Neal provides a shader, if the object is below the camera, and another shader, if the camera is above the camera. Otherwise, another shader is used. However, in our application scenario, we cannot make this distinction. The terrain can be higher than the camera (mountains), and the clouds can be below the camera (if the airplane overfly clouds). However, both shaders can be merged into one, and we query the object's position.

The main advantage of this algorithm is that it can be implemented fast. The algorithm can be implemented in one shader. However, we need to determine the values of many parameters. Finding a good parameter setting is very time intensive.

Additionally, we implement the approach from Bruneton et al. [Bruneton08]. Their technique simulates single scattering, as well as multiple scattering. In figure 5 the resulting atmosphere with terrain is shown. As seen in this figure, their technique looks very good. Especially, the atmosphere results in more realistic look at sunset (see figure 13). The algorithm is more time-intensive to implement. However, it results in a very realistic model of the atmosphere.

8.4 Terrain and objects

Another important part of *CloudyDay* is the integration of the terrain and other objects (e.g., airplanes). We use the atmosphere model to create a realistic representation of these objects.

Optimally, the terrain can be created using tessellation shader. Our approach depends on a view-dependent level-of-detail. Regions near the viewer are drawn with more detail, than regions that are far away. This approach enables a very detail representation of the terrain.

We use for shading of the terrain and other objects the Phong shading approach. Instead of interpolation of the colours between the vertices, we interpolate the normals, and the lightning is calculated for each fragment. This implementation prevents artefacts, resulting from colour interpolation.

8.5 Rain and fog

Rain and fog are two important phenomenon in nature. The rain is implemented using a simple particle system and each raindrop is shown using a billboard and a texture. The fog is created with a decreasing exponential function or with stratus clouds.

A raindrop is usually very thin but long (similar to a stroke). If we would use a simple 2D texture, the raindrops would be look unrealistic, if the camera positions shows from the top to the bottom (or vice

versa). Instead of a 2D texture, we use a 3D texture of the rain, which consider the viewing direction to the raindrop. This results in a more realistic representation of the rain.

The rain implementation can be adjusted with several parameters. The rain intensity can be adjusted with the density, size, lifetime, and velocity parameters. Furthermore, the region, where the rain occurs, can be determined. The parameters can be adjusted by an artist or the test-case generator.

Both, the rain and the fog are important to test CV algorithms. Both phenomenon affect the representation of objects. Fog influences the texture. If the fog is very thick, than objects can be invisible. The rain also results in some problems of object detection.

A disadvantage of our rain implementation is, that we do not simulate the raindrops on the camera lens. This effect would enable more difficult test-cases. In future work, we will add such effect. Furthermore, the rain implementation can be adapted to create a snow and hail.



Figure 20: Nimbostratus clouds with rain and fog

8.6 Post-Processing

In this section, we discuss the results of the post-processing effects. First, the scene is composited. Then, the light scattering and bloom effect is created. Furthermore, we implement a star effect. In the last step, the HDR mapping is done, as well as the colour grading to adjust the final colours. In this section, we discuss each step in detail.

Light shafts: The light shafts are created as a post-processing effect. The main advantage of this approach is that is very easy to implement and it results in a good representation. An example of the light-shafts is given in figure 12.

The disadvantage of this approach is, that this approach only works, if the sun is within the viewing frustum of the camera. In future work, we will create a viewing-independent solution, which consider the terrain and the clouds.

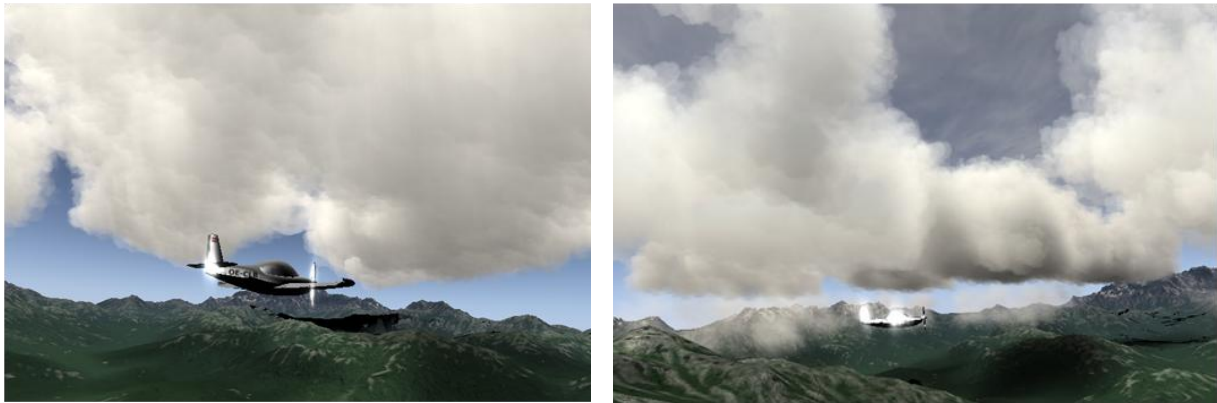


Figure 21: Bloom and star effect at the airplane.

Bloom and star effect: *CloudyDay* also provides a bloom and a star effect. The bloom effect is used to create blooms, which results from specular reflections[Lukks07]. The star effect results from the microscopic bumps and scratches in the camera optic, which produce internal reflections and refraction [Lukks07]. The bloom and star effect are important to test CV algorithms because it is difficult to handle for object recognition algorithms.

Our used techniques are very easy to generate. Furthermore, we can implement it fast with our down-sample approach. As seen in figure 1, the resulting bloom and star effect look very good.

HDR Mapping: After calculation of the light shafts, we map the high dynamic range into the low dynamic device colour space range. As presented in section 6.2, *CloudyDay* provides four HDR mapping algorithms. An example of the different HDR mapping techniques is shown in figure 20.

As presented by Luksch [Luksch07], Reinhard's operator has a good contrast and show all details of the scene. However, the image looks rather grey. With the Modified Reinhard's operator one lose a bit of the contrast, but the colours are more saturated. The sky is dark with both operators.

Colour grading: Additionally, *CloudyDay* supports colour grading. This technique enables to modify the resulting colours for each object by an artist.

The advantage of colour grading is that the colour grading can be created easily. The artist can create different effects by creating of several lookup tables. The lookup tables can be created with any image-editing software or 3D modelling software. Each of the lookup tables can be used to create different test-cases. E.g., to simulate colour engraving. *CloudyDay* supports own lookup tables for different objects.

9. Summary and Future Work

In this technical report, we present an application to generate volumetric clouds, atmosphere, and other natural phenomenon. We adapt the half-angle slicing

approach to generate shadowed clouds. The clouds can be simulated using cellular automate. Furthermore, we generate light shafts, as well as HDR mapping, and colour grading.

We consider the specific requirements for testing computer vision applications. E.g., clouds, which looks like airplanes, can be generated. Furthermore, all effects can be controlled by modifying the parameter space. This enables to generate a lot of different test-cases by modifying each attribute. E.g., the atmosphere can be used to adapt test different lighting conditions, colour grading can be used to simulate sources of errors and more.

In future work, more natural phenomenon should be implemented, like snow and lightning. Such natural phenomenon enables to test difficult test-cases, which are difficult or expensive to simulate in real-world. Furthermore, more errors of a camera should be simulated (e.g., motion blur) to test the effects of some errors, result from the camera.

References:

- [Bouthors08] Antoine Bouthors, Fabrice Neyret, Nelson Max, Eric Bruneton, and Cyril Crassin. 2008. Interactive multiple anisotropic scattering in clouds. In *Proceedings of the 2008 symposium on Interactive 3D graphics and games* (I3D '08). ACM, New York, NY, USA, 173-182.
- [Bruneton 08] Bruneton, E. and Neyret, F. (2008), Precomputed Atmospheric Scattering. *Computer Graphics Forum*, 27: 1079–1086.
- [Crane07] Crane, Keenan and Llamas, Ignacio, and Tariq, Sarah: "Real Time Simulation and Rendering of 3D Fluids", 2007, GPU Gems 3, chapter 30, Addison-Wesley
- [Dobashi00] Y.Dobashi, K.Kaneda, H.Yamashita, T.Okita, T.Nishita, "A Simple, Efficient Method for Realistic Animation of Clouds," *Proc. SIGGRAPH2000*, 2000-7, pp. 19-28.

- [Donnelly06] Donnelly, William, and Andrew Lauritzen. 2006. "Variance Shadow Maps." In *Proceedings of the Symposium on Interactive 3D Graphics and Games 2006*, pp. 161–165.
- [Ikits04] Milan Ikits, Joe Kniss, Aaron Lefohn, Charles Hansen: "Volume Rendering Techniques", GPU Gems, pp. 667-692, Addison Wesley, 2004
- [Kshitiz06] Kshitiz Garg and Shree K. Nayar. 2006. Photorealistic rendering of rain streaks. *ACM Trans. Graph.* 25, 3 (July 2006), 996-1002
- [Green08] Simon Green: "Volumetric Particle Shadows", NVIDIA Whitepaper, 2008
- [Harris01] Mark J. Harris and Anselmo Lastra, Real-Time Cloud Rendering. *Computer Graphics Forum (Eurographics 2001 Proceedings)*, 20(3):76-84, September 2001
- [Kaneko01] Tomomichi Kaneko and Toshiyuki Takahei and Masahiko Inami and Naoki Kawakami and Yasuyuki Yanagida and Taro Maeda and Susumu Tachi: " Detailed shape representation with parallax mapping", In *Proceedings of the ICAT 2001*, 2001
- [Luksch07] Luksch, C.: Realtime HDR rendering. Tech. Rep., Institute of Computer Graphics and Algorithms, TU Vienna (2007)
- [Mit07a] MITCHELL K.: Volumetric Light Scattering as a Post-Process. Addison-Wesley, 2007, pp. 275–285.14
- [Mit04a] Mitchell, J. 2004. "Light Shafts: Rendering Shadows in Participating Media." Presentation at Game Developers Conference 2004
- [Müller12] Daniel Müller, Juri Engel, Jürgen Döllner: Single-Pass Rendering of Day and Night Sky Phenomena. *VMV 2012*: 55-62
- [O'N05] O'NEILS.: Accurate atmospheric scattering. In *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation(2005)*, Addison-Wesley Professional.
- [Nishita99] Nishita, T.; Dobashi, Y., "Modeling and rendering methods of clouds," *Computer Graphics and Applications, 1999. Proceedings. Seventh Pacific Conference on*, vol., no., pp.218,219, 326, 1999
- [Perlin02] K.Perlin. Improving noise. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. ACM Press, 2002.
- [Rakos10] Daniel Rákos: "Efficient Gaussian blur with linear sampling", 2010, Available at: <http://rastergrid.com/blog/2010/09/efficient-gaussian-blur-with-linear-sampling/> (date: 1.10.2014)
- [Tariq07] S Tariq: "Rain", NVIDIA Whitepaper, 2007
- [Wang03] Niniane Wang. 2003. Realistic and fast cloud rendering in computer games. In *ACM SIGGRAPH 2003 Sketches & Applications (SIGGRAPH '03)*. ACM, New York, NY, USA, 1-1.
- [WCloud14] Wikipedia: "Cloud", 2014, Available at: <http://en.wikipedia.org/wiki/Cloud> (date: 1.10.2014)
- [Zendel13] O. Zendel, W. Herzner, and M. Murschitz. Vitro - model based vision testing for robustness. *Proceedings to the 44th International Symposium on Robotics - ISR 2013*, pages October 24–26, 2013

Further Images

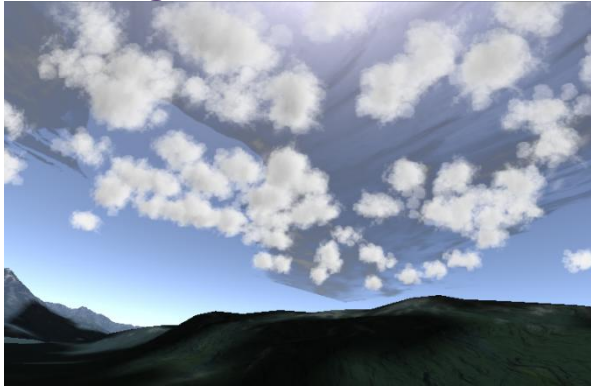


Figure 22: Altocumulus clouds



Figure 23: Sunset

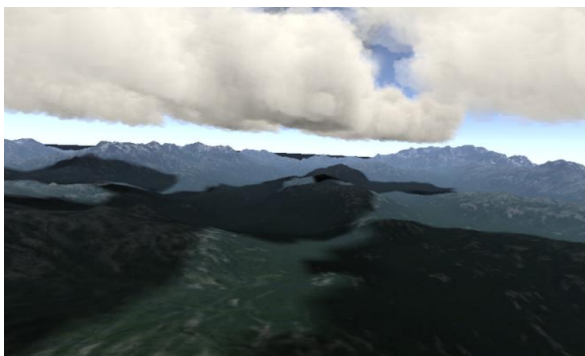


Figure 24: Shadows, casted from clouds, are created using Variance Shadow Map



Figure 25: Cumulus and altostratus clouds (big, thin)



Figure 26: Stratocumulus cloud



Figure 27: Cumulus and altostratus clouds (small, thin)

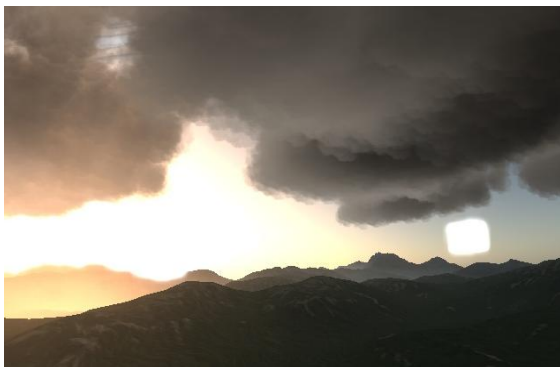


Figure 28: Sunset



Figure 29: Star effect

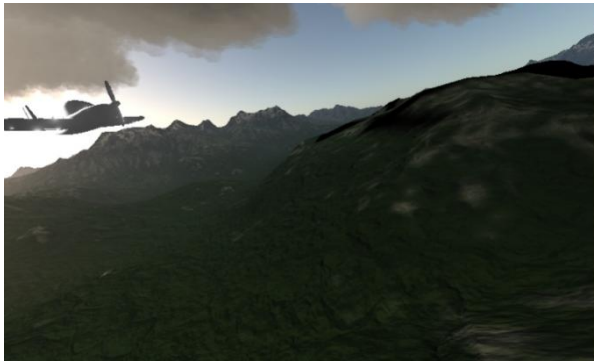


Figure 30: Sunset and glare

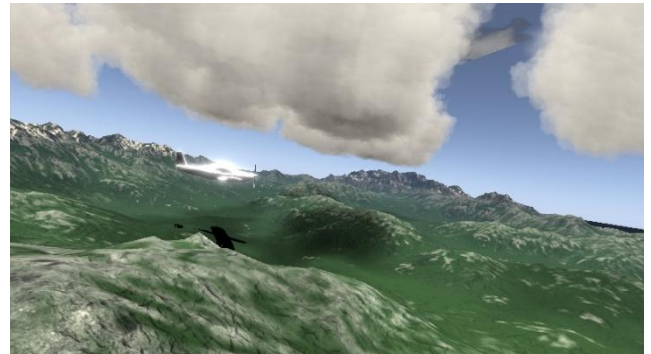


Figure 31: Bloom at day



Figure 32: Clouds at middle altitude (simulated)



Figure 33: Bottom of cumulous clouds



Figure 34: Sunset and cumulus clouds



Figure 35: Cumulous and stratus clouds



Figure 36: Cumulus and cirrus cumulus clouds



Figure 37: Cumulus and cirrocumulus clouds