Cloud Simulation Report

Raphaël Baur, Gianluca Danieletto, Kate Gadola, Mengdi Wang
https://github.com/danielettog/CloudVis

June 14, 2021

1 Introduction

This is the project report of Cloud Simulation group A. Cloud simulation has been an indispensable part both in scientific research and for commercial applications. For instance, it plays an essential role in climate simulations and weather forecasts. However, it is difficult to visualize cloud data sophisticatedly. Firstly, along with the development of models and hardware, cloud data increases significantly in both resolution and complexity, which challenges the visualization of cloud data. Moreover, in contrast to other visualization tasks, cloud visualization is often required to be interactive. Thirdly, cloud data can be multi-modal and be contaminated by missing values. To provide a smooth visualization and high degree of interactivity, we utilized both VTK and Unity in this project. Unlike other groups, we used VTK solely for data pre-processing. For our visualizations we discarded VTK and leveraged Unity instead, which provides a friendly step into user interaction. Since Unity does not support techniques like ray marching or scattering by default, we implemented them from scratch for visualizing scalar fields. For a vector field, we implemented a fourth order Runge-Kutta and ported part of the computation to the GPU in order to achieve real-time results. We also managed to show isosurfaces in an interactive way. Additionally, we integrated an intuitive user interface for a more game-like experience.

In the last section of this report we included detailed instructions on how to run our visualization.

2 Data

The data for this project was originally provided by the German Climate Computing Center (DKRZ) and simulated in the HD(CP)^2 project [2]. The raw data contains both 2D and 3D information describing the weather situation above Germany on April 26, 2013. In the scope of this project, we focused on a subset of the raw data, namely the three scalar fields *CLW* (specific cloud water content, kg/kg), *CLI* (specific cloud ice content, kg/kg), and *QR* (rain mixing ratio, kg/kg), as well as a vector field containing wind speeds. This vector field is in turn split into the three scalar fields *UA*, *VA*, and *WA* (zonal, meridional, and vertical) and measured in m/s. All data used in this project is quantitative.

The dimension of the data is $1429 \times 1556 \times 150$, with a resolution of approximately 525 m along the x-axis, 500 m along the y-axis and about 133 m in the direction of the z-axis. The volume covered by the data grid ranges from 4.5 to 14.496 in longitude, from 47.5 to 54.4975 in latitude and from 183 m to 20'000 m in altitude. The data spreads three time steps for each scalar field. Since the number of time steps are too few for any meaningful time-dependent visualization, we discarded the last two time steps and focused solely on static visualizations concerning only one time step. We decided not to incorporate the scalar field *PRES* into our application, as it does not have enough local variability to visualize it in a slice.

In addition to the provided data, we utilized a 2D scalar field *Z_IFC* (height above sea level, measured in m) and an earth map from NASA Earthdata Research to generate a 3d world terrain in Unity.

3 Goals

Interactivity: To make our visualization appealing to the user, we wanted to make it as interactive as possible. To that end, we have implemented:

- *Fly-through navigation* which allows the user to experience the visualization by flying over Germany. For a better sense of orientation, a more extensive user interface with a minimap was added.
- The *slice plane*, which selects a plane in front of the user view. In the final project the seed-points for the streamtube visualization were sampled on this plane.
- *Visualization on demand*, which allows the user to toggle different visualization modes on and off. Further work would be required to blend different visualization modes more seamlessly.

Realism: As an additional distinction from previ-

ous work we aimed to create an approximately photo-realistic experience by implementing:

- *Ray-casted clouds.* Although it was initially planned to achieve even more realistic cloud rendering with multi-scattering, we have settled for single scattering in favor of a higher frame-rate.
- *More realistic ground.* We achieved this by using a satellite image as ground texture.

The following timeline summarizes the progress for each milestone:
Milestone presentation:

- Data-transformation into formats that were usable in Unity (3D-textures).
- Slice-plane selection.
- Simple rendering of scalar data with direct volume rendering.
- More realistic ground.

Final presentation:

- Ray-traced clouds with single scattering. The frame-rate was not yet acceptable for an interactive visualization.
- Iso-surfaces and iso-lines for clouds. Some ray-tracing artifacts still evident. Does not support real-time interaction yet.
- Complete overhaul of the user-interface. Implementation of interactive map.
- Simple real-time streamlines. No transfer function for coloring implemented yet.

Final report:

- Completion of the visualization-on-demand feature.
- Better frame-rate achieved with single-scattered clouds.
- Removal of ray-tracing artifacts for iso-surfaces and iso-lines. Can now be viewed in real-time.
- Replacement of stream-lines with stream-tubes. Velocity based color-transfer function for stream-tube color added.

4 Visualizations

4.1 Data Preprocessing

As we implemented an interactive visualization application, it was essential to reduce the size of the data such that it fits onto the GPU and can be efficiently processed in real-time. The size of the first time step of the original data for the three scalar fields *CLI*, *CLW*, and *QR* totalled to 2.10 GB, and for the vector field it amounted to 4.42 GB. Using VTK, we first combined the three scalar fields into a single binary file, and then did the same for the three individual components of the vector field. Following this, we created one 3D texture each using C# scripts in Unity: For every scalar value we reduced its precision down to a single byte. In the case of the combined scalar fields, the values were normalized to floating-point numbers in the $[0, 1]$ range. For the vector field (see Figure 1), each of the x-, y-, and z-components was discretized and normalized to fit into the $[0, 255]$ range. The resulting texture sizes are 1.24 GB and 0.93 GB respectively.

Apart from the data provided to us for the project, we asked the lecturer for additional data. We used Python to read the 2D altitude scalar field *Z_IFC* and scaled it. We then stored the height map as an 8-bit PNG image. Moreover, using the longitude and latitude range, we located our area of interest and extracted an earth map for this region from NASA Earthdata Research. These two images were used to generate a 3D world terrain in Unity.

4.2 Ray Marching and Scattering

Clouds exhibit a incredibly large range of visual appearances and phenomena in the real world, making it a challenging task to reproduce them virtually. Especially so if they must be rendered in realtime. To achieve somewhat plausible results several approximations and simplifications to the realistic scattering and bouncing of light within the cloud volume have to be made. The approach used for this project simply traces rays through the cloud volume for each pixel within the camera and samples the densities of water content at a fixed step size. There it approximates the scattering behaviour and light attenuation with an approximating phase function developed by Henyey-Greenstein and marching a single scattered ray towards the sun, sampling the volume again a fixed amount. The fixed step size led to clearly visible slices due to the low resolution of the cloud data, which is remedied by varying the starting position of each marched ray randomly, generating a more noisy, but visually more acceptable effect. Due to the compression of the cloud density data to only 8 bits and the limits of expression in luminance variation due to the single scattering, the visual range of the resulting clouds is very limited, not to mention having the limitations due to a lack of high dynamic range images as output. Another simplification was to treat ice and water (clouds and rain and fog) all equal with the same scattering, further diminishing the possibility of realistic appearance due to computational limitations. Having also no reference on how ice, water, rain and fog affect the visual appearance at the physically accurate density values that were provided, leaves the result to just being an artistic approxima-

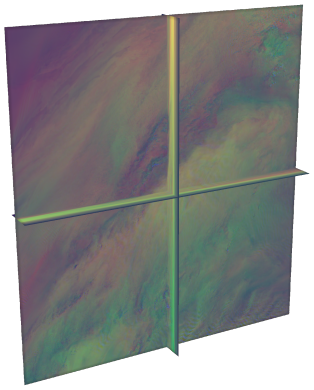

Figure 1: Three slice planes in the 3D-texture which encodes the vectorfield. The red, green and blue channels encode the longitudinal, latitudinal and altitude wind-vector-components respectively.

tion to a potentially plausible visual appearance of the clouds for this dataset.

4.3 Wind-Flow Visualization

To visualize the wind-flow we implemented a real-time system to visualize thousands of streamtubes calculated with a 4th-order Runge-Kutta-Integrator. To that end, we leverage the GPU's computational power to trace the streamlines, calculate the tube-mesh vertex positions and subsequently render the tubes in a single simulation update while maintaining an adequate frame-rate. The streamtubes' base color encodes the particle velocity and thus provides an additional layer of information.

To enable efficient data-access by the GPU, the initially provided scalar fields describing the longitudinal, latitudinal and altitude components of the wind-velocities have been encoded in a 3D-texture. Figure 1 visualizes slice-planes in said texture.

Figure 4 shows a comparison of different streamline visualizations with varying amount of depth cues. Table 1 shows a performance comparison between a purely CPU-based and GPU-based implementation, highlighting the need of employing a GPU if a real-time experience is desired. The scripts used for the streamline visualization can be found here, along with a more detailed description of the pre-processing and computation.

A good example of streamtubes can be seen in figure 2.

4.4 Isosurfaces

Besides the more realistic cloud visualizations described in previous sections, we also want users to be

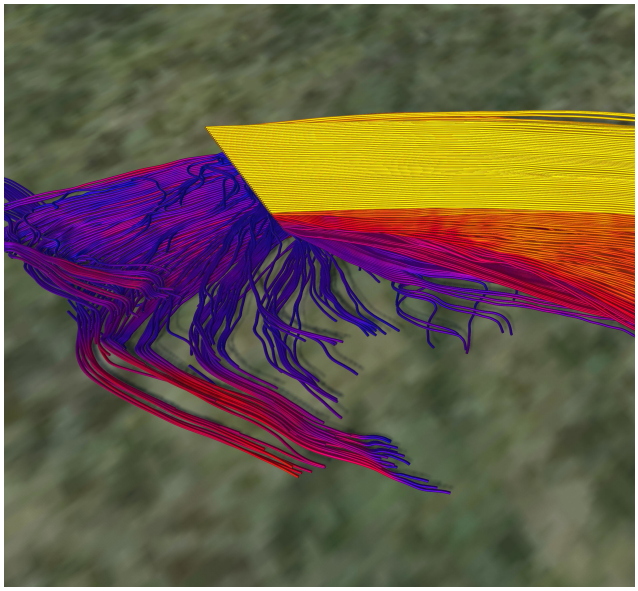

Figure 2: In this example the streamtubes showing a strong difference in wind-flow direction between winds closer to and farther from the ground.

		Number of steps			
		256	512	1024	2048
Device	CPU	4.38	2.17	0.98	0.45
	GPU	23.39	12.59	10.40	2.89

Table 1: Comparison of frame-rates between a purely CPU- and GPU-based approach for the wind-flow visualization for 1024 streamlines across different numbers of RK(4)-steps. Speed-ups are approximately 6-fold with the GPU-based approach. Measured on a Intel i9 CPU and NVIDIA Tesla 2000T GPU.

able to explore the underlying data and its characteristics in a more quantitative manner. To this end, we implemented the functionality to view isosurfaces for the three cloud datasets, namely *CLW*, *CLI*, and *QR*. The user can select any one of these components individually, or view them all together. The isovalue can be interactively adjusted between a value of 0 and 1, which is then linearly interpolated between the minimum and maximum value encountered within the corresponding dataset. This interactivity allows the examination of isocontours and their topology under varying isovalues, as splits and joins of the volumes are clearly discernible.

To find the isosurfaces in the 3D data in real time, we used a ray-marching method with early termination as soon as each ray intersects a voxel with a value equal to or larger than the specified isovalue. To further distinguish between the different data components, we added a transfer function depicting the contribution of cloud water, cloud ice, and rain (blue, grey, and red respec-

tively) to each location on the isosurface. Depth cues were added in the form of a Blinn-Phong illumination model, highlighting the structure of the extracted volumes (see Figure 3).

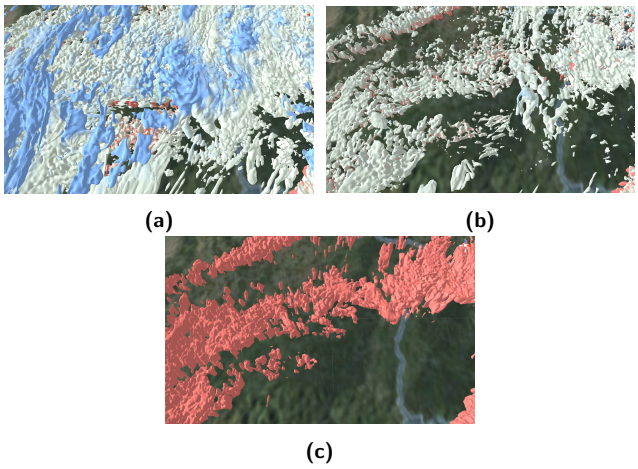

Figure 3: Isosurfaces of a climate phenomenon with vorticity and precipitation south-west of Stuttgart with a low (a) and high (b) isovalue. (c) shows the isosurface of only the *QR* component.

4.5 User Interaction

We benefited much from Unity for a video-game-like visualization. To achieve this, we firstly used Python to read and extract a height map from *Z_IFC* in PNG format. Combining an earth map from NASA Earthdata Search, we built a 3D world terrain in Unity, which provides an overview of the whole of Germany and its surrounding countries with boundaries and cities. We implemented intuitive movement for the camera such that during game play, the user can move freely in the world at will. In order to navigate during visualization, we included a minimap and a corresponding second camera in the game. Longitude, latitude and altitude are also displayed in real-time such that the users can better locate themselves. For a friendly game experience, we also added some play instructions to the game.

A large amount of effort was contributed to view switch during visualization. The game supports switching amongst three views of single scalar fields (*CLI, CLW, QR*), one view for isosurfaces involving all three scalar fields and one view for wind vector field. One can turn on or off each view during game play flexibly. Especially in the view of vector field, the user can play around with camera movement and find out how streamlines proceed, which are seeded on a plane that is orthogonal to camera's view direction. When viewing isosurfaces, the user can adjust the isovalue dynamically to see how isosurfaces vary, split and merge.

5 Contributions

5.1 Raphaël Baur

Implementation of real-time capable stream-tube visualization.

5.2 Gianluca Danieletto

Cloud visualization through raymarching with single scattering. Slice plane selection in front of the camera.

5.3 Gadola Kate

Interactive isosurface visualization for each of the three cloud data components.

5.4 Mengdi Wang

Data preprocessing (ParaView, heightmap, earth map), world terrain, camera movement, UI design and implementation, integration of others' works.

6 Discussion

6.1 Limitations

We tried to do something different than computing some specific property from a single viewpoint that is usual in scientific visualization due to the large amount of computation time. Therefore, we had to cut down on complexity of the data visualized. We achieved the realtime aspect such that it allows the user to fly through the volume and explore it at will, but we did not meet the expectation of a perfectly realistic appearance. Given the time budget and the complexity of building virtual worlds we did well. There is still a lot of room for improvement though. Even at this state, due to a lack of sophisticated algorithms for cloud rendering, the visualization is quite taxing on the hardware. Next to that, we faced quite a bit of technical difficulties due to our inexperience with the software at hand. That time we spent on figuring things out and debugging could have been used to experiment and iterate different visualizations for the dataset to find something that actually conveys more information at once. Our initial idea of having a slice of more complicated data visualized in front of the camera, next to the realistic appearance turned out to be not very helpful. This is why you can visualize both parts, but separately. Short wind-vectors did not provide a lot of useful information, and larger ones clutter the screen or were covered by the clouds.

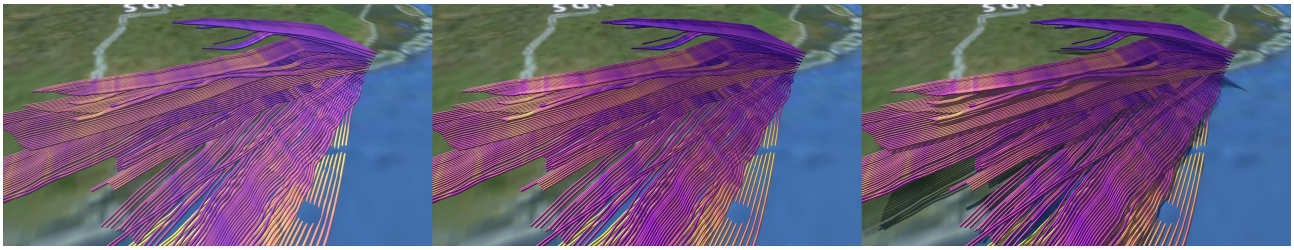

Figure 4: The material choice and shadowing can aid the viewers depth perception. Left: Simple diffuse shader. Middle: Specular highlights. Right: With self-shadowing.

6.2 Future Work

In the following we list some ideas for future work:

- *General - Incorporate multiple time-frames*: As for now, we have only worked with data sampled at a single point in time. A possibility for extending our work would be to use multiple time-frames in the visualization.
- *Wind-flow - More elaborate seeding strategy and depth-cues*: The wind-flow visualization would benefit from a more elaborate seeding strategy, e.g. seeding close to critical points as outlined in [3]. Furthermore, additional depth-cues such as depth-dependent halos [1] would enhance the depth-perception. When multiple time-slices would be used, additional means of visualization such as path-lines could be explored. Additionally, the seeding could be decoupled from the camera location, allowing better exploration of the flow.
- *Dynamically Adaptive Sampling, LODs, Octree*: Currently, the ray-marching and scattering work with fixed sampling distance. This is inefficient, as clouds are not distributed uniformly over the space. There are a lot of possible improvements to the volume rendering to improve the appearance of the clouds. One possible improvement is to sample along view direction with adapative sampling distance. Another approach could be baking the distant clouds to impostors with precomputed Monte Carlo raytracing using multiple scattering, or just change the underlying data structure to something more efficient as an octree. There is also the possibility to only render parts of the clouds each frame, to reproject previous frames and reuse those views. The possibilites are nearly endless and each comes with its own challenges regarding technical implementation.
- *More Viewing Options*: Right now, we only allow for viewing of clouds, wind streams and isosurfaces. But there can be many more properties that could be visualized such as turbulence, updraft regions, ground shadowing etc. to be included as well. All of these could also explore different ways of blending several of those together with color, transparency or even split screen effects.

7 How to run

Here we provide a detailed instruction on how to run our visualization. Required softwares are VTK and Unity version 2021.1.5f1 or higher.

Since our visualizations are created in Unity, we needed to transform the data from VTK-specific formats to Unity-readable formats. Two assets will need to be generated, each 1.2 GB in size. If your connection is sufficient, you can bypass 7.1 (Data Preprocessing) and download the assets located in the following `polybox-folder`. Once downloaded, place them in `CloudVis/Assets` and continue with Section 7.2. If you cannot or do not want to download these assets, please commence with Section 7.1.

7.1 Data Preprocessing

Once the project has been cloned or downloaded from the git repository, run the VTK programs `createVectorField.cpp` and `combineCloudData.cpp` located in the `VTKscripts` folder. If executed successfully, the binary files `clouds.data` and `vectorfield.data` can be found in the working directory or the directory that is specified in the scripts. Then open the project with Unity and move the two generated `.data` files into `Assets/ImportedData`. Focus on the prefab `Assets/Scripts/TextureCreator` in Unity and, in turn, run *Generate cloud texture*, *Generate Vectorfield*, and *Generate Noise Texture*. The three generated `.asset` textures can now be found under `Assets`.

7.2 Attach Textures to Camera

Select the scene `Assets/Scenes/CloudVis` to display it in the hierarchy. Focus on *Main Camera* and look into the inspector. In the *Cloud Manager* component, modify the fields *Cloud Texture* and *Noise Texture* by selecting the corresponding generated or downloaded

textures. Similarly, in the *Vector Field Visualizer* component, set the *Vectorfield Texture* to the correct texture file. Then run the game.

7.3 Control during Visualization

The camera movement works as in standard FPS games. Use WASD or D-pad for movement. Use the keys Q/E for moving up/downward. Hold CTRL/SHIFT/MouseLeft to speed up movement. Keyboard numbers specify different views: 1 for *CLI*, 2 for *CLW*, 3 for *QR*, 4 for wind vector field, 5 to toggle the isosurface on and off, and 0 for showing nothing. The same number is used to turn on/off each view. When visualizing isosurfaces, use the keys J/K to decrease/increase the isovalue, and use the key M to iterate through the isosurfaces for *CLI*, *CLW*, *QR*, or a combination of all three.

The parameter displacement in the component *Vector Field Visualizer* of *Main Camera* describes the number of units the streamtubes are displaced on the slice window. Choosing higher values will result in less streamtubes being rendered and vice versa. For computational reasons, the maximal number of streamlines rendered at any point in the visualization has to be fixed with the parameter *Max Streamline Count*. The streamline color can interactively be changed by modifying the parameter *Gradient*. The number of integration steps can be chosen by double-clicking on *New Tube Template* in the parameter *Template* and subsequently choosing a value for *SegmentCount*.

One can also hit the keyboard button H to view the control instruction during game play.

References

[1] M. Everts, H. Bekker, J. Roerdink, and T. Isenberg. Depth-dependent halos: Illustrative rendering of dense line data. *IEEE transactions on visualization and computer graphics*, 15:1299–306, 01 2010.

[2] N. Rimensberger, M. Gross, and T. Gunther. Visualization of clouds and atmospheric air flows. *IEEE Computer Graphics and Applications*, 39:12–25, 2019.

[3] T. Weinkauf. Extraction of topological structures in 2d and 3d vector fields. pages 311–320, 01 2008.