Cloud Visualization
Scientific Visualization Project Report

Group MeteoCAB: Davide Corigliano, Peter Haas, Frawa Vetterli and Annika Öhri

July 9, 2022

1 Introduction

Weather conditions have many implications on everyday life. Based on the weather, we decide whether to spend our free time outside, what (outdoor job) tasks are possible to tackle, etc. Furthermore, it is very important to recognize and warn when a potentially dangerous weather situation such as a storm is building up. Visualizing data properties linked to weather conditions can help analyze a situation and extract valuable information from it.

2 Data

Our dataset contains meteorological properties simulated over central Europe for April 26, 2013 in three timesteps. It is a subset of the HD(CP)² climate simulation data [1]. The given data is located on a rectilinear grid. While the sampling on the x,y plane is regular, the sampling height along the z axis is more dense in lower heights. These sampling heights are given as meters above the ground in two files, where only the vertical wind uses the second sampling heights, the rest is sampled at the same points. The x,y sampling space in contrast is along the lat/lon grid instead of meters. We were given the following properties:

- Z_ifc - Terrain height in m
- Cli - Specific cloud ice content, quantitative measurement in kg/kg
- Clw - Specific cloud water content, quantitative measurement in kg/kg
- Qr - Rain mixing ratio, quantitative measurement in kg/kg
- Pres - Air pressure, quantitative measurement in Pa
- Ua, Va and Wa as zonal, meridional and vertical wind, each of them a quantitative measurement in m/s

All of these are scalar properties, however we combined the winds in Paraview to yield a vectorfield instead.

3 Goals

Our dataset only had three timesteps but still required a lot of memory, so we could not expand the temporal resolution by downloading the original data. Due to this, we decided to focus more on spatial properties instead of temporal ones. Our timeline looked like this:

01.04.22	Group Meeting to brainstorm ideas
04.04.22	Project Plan Presentation
05.04-01.05	Task distribution and first visualizations:
	Terrain
	Clouds
	Wind glyphs
	Non-final streamlines
	Pressure with isolines
02.05.22	Milestone presentation
03.-29.05.22	Continue visualizations and improve UI
	Custom slicer
	Custom streamtracer
	Line integral convolution
	Add more legends, better transfer functions
30.05.22	Final Presentation

4 Visualizations

4.1 Data Resampling

As explained in Chapter 2, the initial data is given on a rectilinear grid, where the x dimension corresponds to the latitude in degrees, y dimension to the longitude in degrees, and the z dimension to the height/altitude in meters. Due to this disparity in units, the grid spacing along the horizontal space (xy-plane) is much smaller than the one along the vertical space (z-axis). To meaningfully visualize the data, we therefore rescaled the z-axis of all given data sets by 0.0001 using the provided program "prog_scale_clouds.cpp". Thus, we now have units of 10 kilometers in z direction.
An additional problem is that as mentioned in Chapter 2, in contrast to the x and y dimension, the sampling along the z dimension is not regular. The data was sampled more frequently in the lower heights, accord-

ing to a given height file that irregularly maps the z values in the rectilinear grid to height values above the ground in meters. Since VTK displayed the data as if the z-sampling were regular, this irregular sampling resulted in misleading visualizations that made e.g. clouds look as if they were at a higher height than they actually are. Another issue was that the height values are given in meters above the ground as opposed to meters above the sea level, so for different x,y values, the sampling height in space actually differed too due to the dependency on the terrain height at that particular location. To avoid these artefacts, we transformed each dataset to a regular grid as follows: The x and y dimension sampling stays the same as it is already regular. It is important to note that we did not transform the x and y coordinates from lat/lon degrees to meters in order to accurately capture the earth curvature as the sampled region is small enough compared to the globe such that the transformation would not have changed much and the visualization still intuitively conveyed the data using the flat xy plane. For the resampling along the z dimension, for each specific xy location, we first accounted for the transformation from height above ground to height above sea level by subtracting the terrain height (rescaled by 0.0001 in order to match our previous scaling) from each regularly spaced z value. We then found the property's measurement value at (x,y,z) via linear interpolation in z direction based on the given height mapping file (rescaled by 0.0001). As the vertical wind has a different irregular height mapping which is provided in a second height file, our program can handle regular resampling based on different irregular height mapping files and with that, it allows to combine the three winds as the original data was not sampled at the same locations.
This resampled data generated by resampling.cpp as well as some other data such as the terrain mesh can be found in and downloaded from our Polybox folder [2].

4.2 Terrain Visualization

In order to be able to see interactions with the ground and to let the user know where the data is located, we visualized the terrain. Because the terrain is a surface, we decided to create a mesh from the z_ifc data, which contains for each (x,y) point its terrain height in meters. To match the rest of the data, we also rescaled this height by 0.0001 so the difference between the spacing of the x and y axis is more similar to the spacing of the z axis. Using libigl, we then created vertices at each such (x,y) position and set their z value to be at the rescaled z_ifc value. We then connected these vertices in a regular manner, yielding a 3D model of the ground. However, as the data resolution is quite high, this mesh took up a lot of space and therefore took a very long time to be loaded in VTK. Hence, we wanted to downsample it, but in a way that fits the data well. That is, the more high-frequency regions such as the mountains should be sampled at a higher frequency to better preserve their shape, while flat regions don't contain very interesting information and can be sampled more coarsly to reduce the memory taken up by the mesh. Thus, the mesh should be adaptively resampled. However, along the flat coast, even small differences should be well-preserved as well, as these very small differences decide between a point being in the sea or on land. Hence, we opted for using the adaptive Simplification Scheme Quadric Edge Collapse Decimation implemented in Meshlab, but manually selected non-coast regions only to be simplified by this method, so the coastline is fully preserved. The resulting mesh takes up only about 1/4-th of the original one but has no visible difference.
In order to visualize it, we used the vtkOBJReader and applied a transfer function going from green to brown to white to the mesh to create a natural look of the terrain and to make height differences more visible. However, for the sea, we set the value that decides on the color to nan instead of the height and added a special blue color entry for nan. This allowed a discrete transition from sea to land, while setting the transfer function to be blue for height 0 and green for a value slightly bigger than 0 always resulted in parts of the land getting seemingly flooded by water. To give the observer an idea about the scale of the data, we finally also added a legend to represent which color corresponds to which height. The result can be seen in Figure 1.

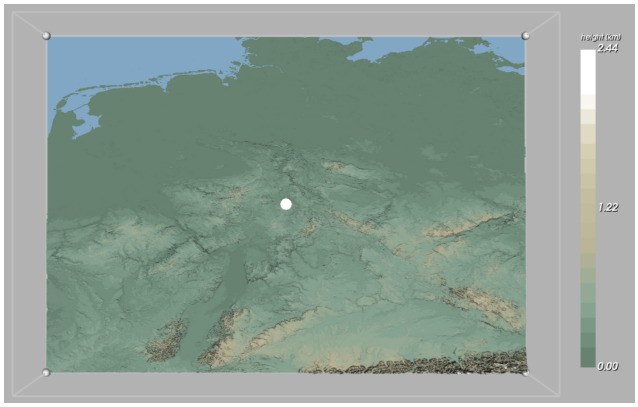

Figure 1: Terrain mesh colored by transfer function with its legend

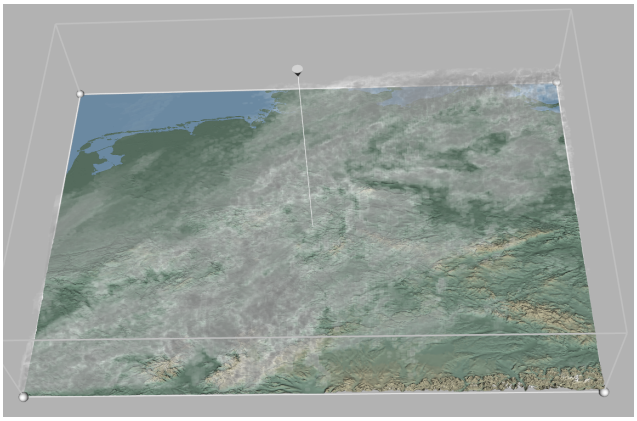

Figure 2: clw (gray) and cli (white) visualized through direct volume rendering

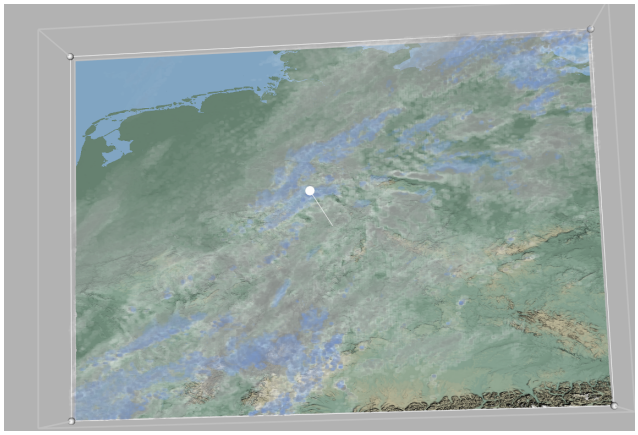

Figure 3: Clouds visualized together with rain (blue)

4.3 Cloud and Rain Visualization

One of the most important features of every weather forecast is some measure for how cloudy a day is going to be at some area. Besides that precipitation in the form of rain or hail is indispensable. Since there is an obvious correlation between these two aspects of weather, we decided to visualize them together. From the dataset we received, we handle the properties clw (cloud water content), cli (cloud ice content) and qr (rain mixing ratio). Qr we simply use as a measure of how much rain is falling in a certain area, although we have no notion of (metric) amount of water per area. clw and cli are shown separately where the absence of both simply means the absence of clouds.

The visualization of these quantities is done through direct volume rendering. In essence we use 3 separate renderers from the vtkOpenGLGPUVolumeRayCastMapper component. Rain is visualized in gray color, ice in white and rain is blue.

The result of the visualization can be seen in Figures 2 and 3

The separation of the 3 renderers instead of combining them into one scalar field, helps visualizing the rain, as this, when viewed from above, would mostly be hidden by the white and gray from the clouds if handled otherwise. This way we allow the user to toggle the rain visualization separately. What can nicely be observed in Figure 2 is that the ice content is typically higher up in the atmosphere compared to the water content.

4.4 Air Pressure with Isolines

In TV weather forecasts, it is common to include some air pressure isolines and an associated indication of low and high pressure areas. However, our data is sampled only over central Europe which is a too small region to capture the large-scale phenomenon of low and high pressure areas. In fact, the pressure in our data set mainly inversely corresponds to the altitude. We visualized this dependency via a vertical 2D slice where all quantitative pressure measurements are mapped to a color transfer function, where the global maximum is orange and the global minimum is blue. Together with a legend for these global measurements, this can be seen in Figure 4. We did not visualize any isolines on the vertical slice as it would not have added a lot of information as the change of pressure along the height is very uniform.

In contrast to the vertical slice, on a horizontal 2D slice at some specific height, the range of measurements is very small compared to the global measurement range. This is exactly because pressure inversely corresponds to height. In the global scale, these horizontal differences were barely visible. Thus, at each height, we mapped only the local measurement range of the corresponding horizontal slice to the color transfer function. Together with displaying isolines, this helps visualize the small local pressure differences better. The resulting visualization includes a corresponding legend for the local measurements that adapts according to the local range in order to still indicate the meaning of the transfer function, even though it changes based on the chosen slice. This horizontal pressure visualization can be seen in Figure 5.

A custom VTK slicer where the user can interactively change the height and depth of the horizontal and vertical slicer, respectively, was used to display the 2D slices. The key “h” can be used to switch between the horizontal and the vertical slicer. The vtkContourFilter was used to visualize the isolines. The slicer widget, which can be controlled by mouse, is an adaption of the vtkPlaneWidget component with constrained position and orientation.

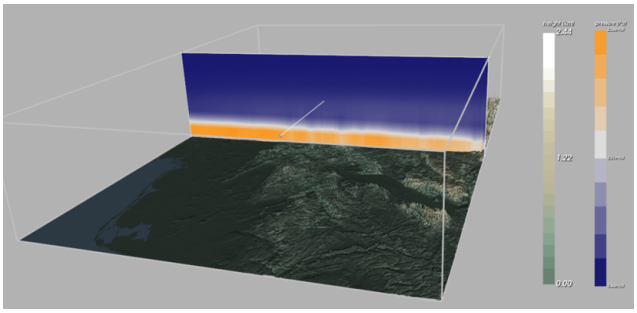

Figure 4: Vertical 2D slice of the air pressure with the terrain underneath: The higher we are in height, the lower is the sampled air pressure.

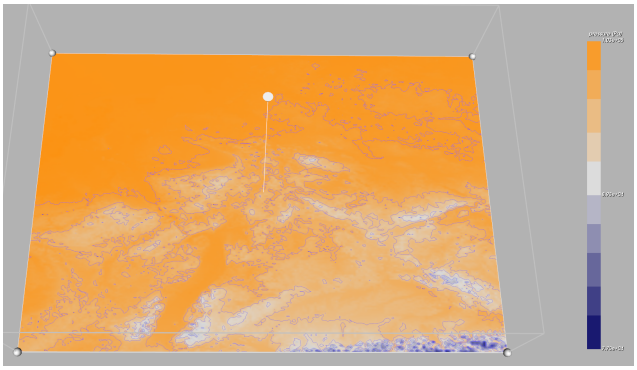

Figure 5: Horizontal 2D slice of the air pressure near the ground including isolines. The colored local transfer function and the isolines help visualize small differences in the pressure at some fixed height.

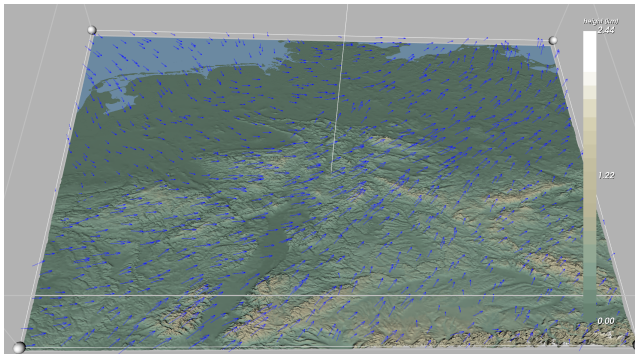

Figure 6: Slice of Wind glyph visualization near the ground

4.5 2D Wind Glyphs

The visualization of wind glyphs allowed us to understand how clouds move above central Europe. We decided to focus on 2D glyphs since they provide more intuitive results than 3D glyphs. Moreover, 2D glyphs are usually present in weather forecasts, and therefore they provide a familiar and intuitive visualization. Since the data contains 3D vectors for the wind, we considered only the zonal (ua, along x-axis) and meridional (wa, along y-axis) components. To create an interactive visualization, "slices" of wind can be selected dynamically using a slider. Once the 2D data is obtained, the vectors are fed into vtkGlyphSource2D and mapped for the visualization using vtkPolyDataMapper. One of the major issues we faced is the clutter. The original data contains many vectors, that may overlap in the final visualization. To solve the problem, we used vtkMaskPoints, which allows us to select vectors at random and put a cap on the number of vectors. Finally, to avoid overlapping between vectors, we set a scale factor of 0.02 for the glyphs. Regarding the results, the wind glyphs allowed us to observe interesting patterns in the wind data. For example, wind magnitude tends to be small near the ground, it then grows rapidly while reaching middle altitudes, and then it finally decreases again at greater heights. Moreover, the wind glyphs give information about wind orientation, which is roughly directed from South West to North East. Nonetheless, it is difficult to capture global flow behavior from the glyphs, and we addressed this problem using streamlines and line integral convolution. The glyphs can be seen in Figure 12. To retrieve slices of 2D wind vectors from the data tensors we assembled using Paraview, we had to implement a VTK component that can slice image data that features vector valued variables because the component for that functionality provided by VTK (vtkImageReslice) seemingly only supports vtkImageData with only scalar variables.

4.6 2D Wind Streamlines with Custom Streamtracer

Since our goal is to understand the spatial relationships of the wind with the ground and the clouds, we decided to implement streamlines. They provide an intuitive representation of wind behavior while encoding information about global motion. Again, we decided to focus on 2D slices of the wind, inspired by the weather forecast. At first, our implementation made use of vtkStreamTracer, but it suffered from several artifacts. In fact, the wind inside a 2D slice has great variation in magnitude, and therefore big velocity vectors tend to dominate during integration. This phenomenon leads to the loss of details and local information as its integration step are sometimes too large to capture some local phenomena. Therefore, we implemented our own custom streamtracer based on forth order Runge Kutta integration with a normalized step size that should avoid overstepping interesting local behaviours. More precisely, a step from x_i to x_{i+1} was calculated as follows:

$$x_{i+1} = x_i + s * normalize\left(\frac{v_1}{6} + \frac{v_2}{3} + \frac{v_3}{3} + \frac{v_4}{6}\right) \quad (1)$$

with

- $v_1 = normalize\left(v\left(x_i\right)\right)$
- $v_2 = normalize\left(v\left(x_i + \frac{s}{2} * v_1\right)\right)$
- $v_3 = normalize\left(v\left(x_i + \frac{s}{2} * v_2\right)\right)$
- $v_4 = normalize\left(v\left(x_i + s * v_3\right)\right)$

where s is a small constant controlling stepsize and $v(x)$ is calculated via bilinear interpolation of the discretely sampled 2D horizontal wind vector field at position x. The new stream tracer produces more stable results and allows obtaining more precise streamlines. In Figure 7, we can observe that the custom streamlines are now able to represent local behavior. For example, at low altitudes, they are influenced by obstacles. For the integration, we used 1000 steps of size 0.01. The new stream tracer provides us with a set of points (stored in vtkPolyData as vtkPoints), which are then transformed into vtkLines to perform the visualization. Regarding the seeds, they are sampled uniformly over the 2D wind slice, and then they are fed into the new stream tracer. This technique provided us with a clean representation, with little to no clutter in every slice. Interestingly, the streamlines underline some characteristics of the data. Near the ground, the streamlines look irregular and curve around hills and mountains. In contrast, when proceeding to middle altitudes, the streamlines show a uniform behavior. Finally, the streamlines become irregular again at the highest altitudes, where it is possible to observe attachment and detachment points and small vortices. This behavior can be also observed using the LIC texture, in Figures 8, 9, 10 Finally, the wind glyphs allowed us to test the correctness of our streamlines since the streamlines are tangent to the wind vectors at every point.

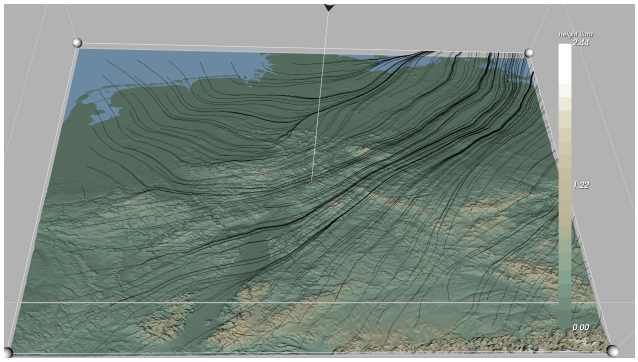

Figure 7: Slice of streamlines near the ground

4.7 Line Integral Convolution for 2D Wind

After the streamlines, we decided to implement a line integral convolution of the 2D wind slices. This way, we obtained a more informative visualization that shows both the local and global behavior of the wind vectors. Like glyphs and streamlines, it is possible to visualize the LIC interactively by moving a slider. But, in contrast to the previous techniques, the LIC textures are not constructed in real-time, since the computation is slow. Instead, high-resolution LIC images (720p) are precomputed, and then dynamically accessed by the slider. Regarding the implementation, we decided to build everything from scratch. First, we created a noise texture of grey pixels sampled from a uniform distribution. Then, for each pixel, a streamline is computed by using the mentioned normalized Runge-Kutta integration, both in the forward and backward directions. We used a step size of 0.01 and 15 steps in both directions (for a total of 30). Finally, on the output image, we assign the average color of the pixels accessed during the integrations steps to the pixel corresponding to the seed. Since points on the same streamline are correlated, this method produces continuous lines, which in turn encode the behavior of the wind. We used our custom steam tracer also to produce the LIC textures. This way, we were able to capture local behavior more precisely. The behavior of the LIC texture at different altitudes can be observed in Figures 8, 9, 10. The LIC texture visualization underlined the behavior observed in the streamlines while producing a more intuitive representation. We tested the correctness of our implementation by comparing the glyphs,

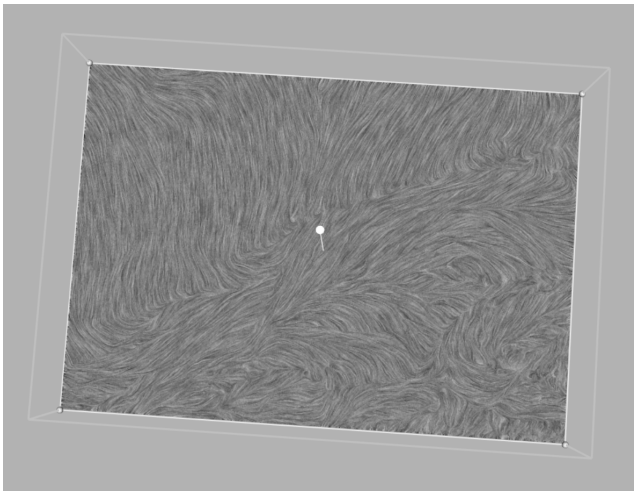

Figure 8: LIC texture near the ground

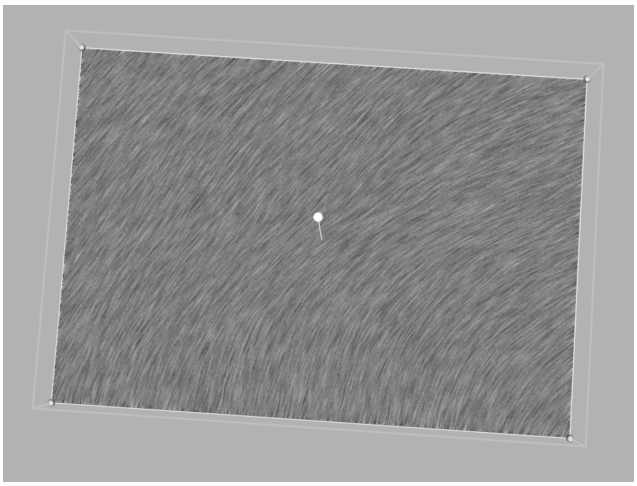

Figure 9: LIC texture at mid altitude

the streamlines, and the LIC texture results.

4.8 Divergence Visualization

As air in a large scale setting like this is approximately incompressible, we wanted to show where air flows vertically through the atmosphere in an implicit way, i.e. without using the provided "wa" (vertical wind) quantity. This can be done by considering the divergence of the wind vector field that we assembled by combining the "ua" and "va" wind (x,y / horizontal directions). With our assumption that air in this setting is incompressible, a divergence value $\neq 0$ suggests that air escapes the corresponding horizontal plane (be it up or down) and thus there must be vertical wind.

The visualization of this scalar field is done through direct volume rendering (using vtkOpenGLGPUVolumeRayCastMapper). The computation of the divergence is done through the VTK component for gradients (vtkGradientFilter). Negative divergence (sinks) are shown in red and positive divergence (sources) are shown in blue. Low values (close to 0) are cut off through transparency. We observed large values of divergence at the boundaries of our data tensor. This might be due to missing values at the borders when considering a central difference computation of divergence. To reduce the noise in our visualization we simply cut off the border area of the tensor.

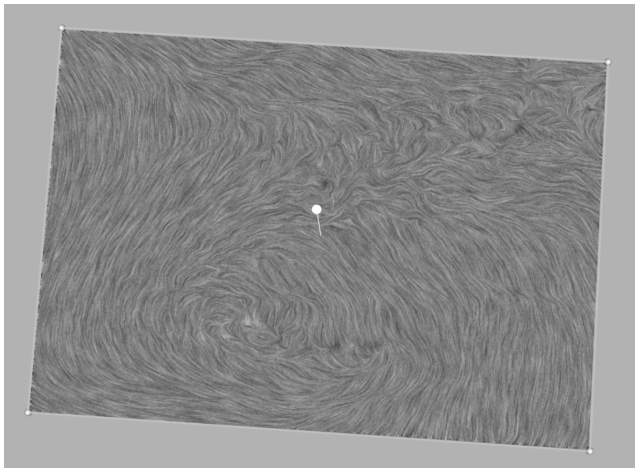

Figure 10: LIC texture at high altitude

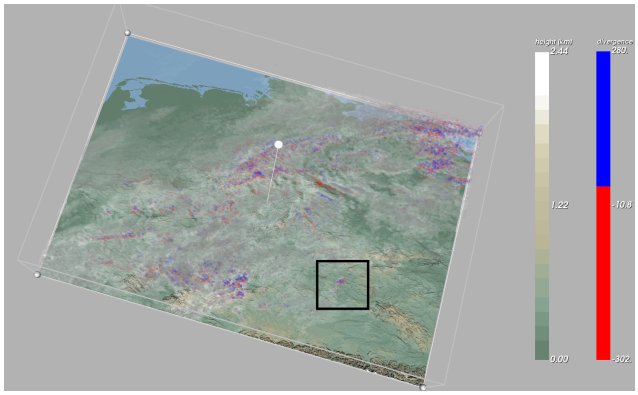

Figure 11: Wind divergence and Clouds. Regensburg area can be seen within the highlighted rectangle

Figure 11 shows the wind divergence combined with the clouds. The visualization suggests nicely that the absence of clouds implies less turbulent and less vertical airflow.

In the area around Regensburg there was a storm happening during the time the data represents. We can see through our divergence visualization that there must be vertical airflow, which would be a typical indicator for the buildup of a cumulonimbus cloud. In Figure 12 it is observable that the wind vectors converging towards the area where there is divergence.

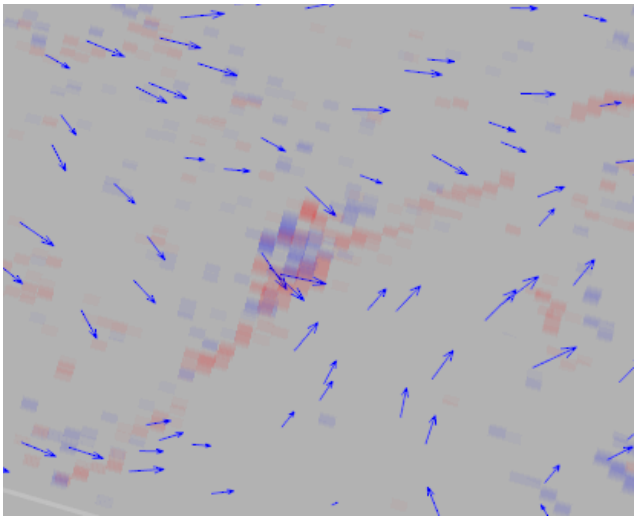

Figure 12: Divergence and wind glyphs around Regensburg indicating vertical airflow

Additionally, the region shows negative divergence in lower heights and positive divergence above that. This indicates a suction effect typical for storms.

5 Contributions

5.1 Davide Corigliano

- Understanding how to access and use the data
- Wind glyphs visualization
- Streamlines using a custom streamtracer
- Custom line integral convolution using a custom streamtracer

5.2 Peter Haas

- Adaption of volume rendering for clouds and wind divergence
- Computation and visualization of wind divergence
- Setup of application to integrate single components, integration of slicer related components
- Custom VTK slicer
- Adaption of interactive VTK plane widget

5.3 Frawa Vetterli

- Splitting up cloud visualization into cloud water and cloud ice content as well as adding the rain and color coding these properties
- Pressure with isolines
- RK4 Integrator and bilinear interpolation for custom streamtracer
- Custom data resampling to make vertical sampling regular and account for terrain height

5.4 Annika Öhri

- Terrain visualization
- Pressure with isolines
- RK4 Integrator and bilinear interpolation for custom streamtracer
- Custom data resampling to make vertical sampling regular and account for terrain height

6 Discussion

6.1 Limitations

6.1.1 Storm near Regensburg

We did not explicitly focus on highlighting the storm phenomena happening in the area around Regensburg. The divergence measure suggests that there is a typical cloud build up happening in that area, however in context with the other divergence structures around the area shown, this storm is small and easily passes unnoticed.

6.1.2 Temporal Slices

The whole application solely relies on the data given to us. However, we only used the first of 3 provided data sets with respect to time. This goes with the risks that certain phenomena we could have seen were hidden because we did not regard for the temporal dimension. Especially for the storm near Regensburg this is likely the case. By using more timeslices, in particular by getting additional data from the Visualization Contest, we could have included other visualizations such as animations over time.

6.2 Future Work

6.2.1 Performance

In its current state, the application that contains all the features we implemented has a slight delay upon every interaction as some of the quantities are recomputed or reevaluated whenever slices change. This could be avoided by doing this re-computation asynchronously or distributed over more than one frame. Additionally, the start up time of the application should be reduced by loading data asynchronously.

6.2.2 Storm near Regensburg

With the given data, a zoom function to see structures around Regensburg more closely and also with higher resolution would come by with the limitation mentioned above.

6.2.3 Vertical Wind

Throughout all the features we implemented, we ignored the "wa" scalar property which refers to the vertical wind. A good addition for the future would be to add 3D wind glyphs. This would be a particularly useful addition to the divergence rendering as with this we could check whether compression manifested through non-zero divergence causes upward or downward draft. Together with a more dense visualization in an area of interest (e.g. the storm near Regensburg), this could make the cumulonimbus build up happening nicely visible to the user. A disadvantage of using the 3D wind instead of the 2D wind is that the visualizations could become more crowded, so more advanced seeding techniques would have to be considered.

References

[1] Data courtesy DKRZ/MPI-M. HD(CP)^2 data sets. `https://scivis2017.dkrz.de/data.html`.

[2] Davide Corigliano, Peter Haas, Frawa Vetterli and Annika Ohri. Our processed data. `https://polybox.ethz.ch/index.php/s/rW5ULLZJKK7SMdK`.