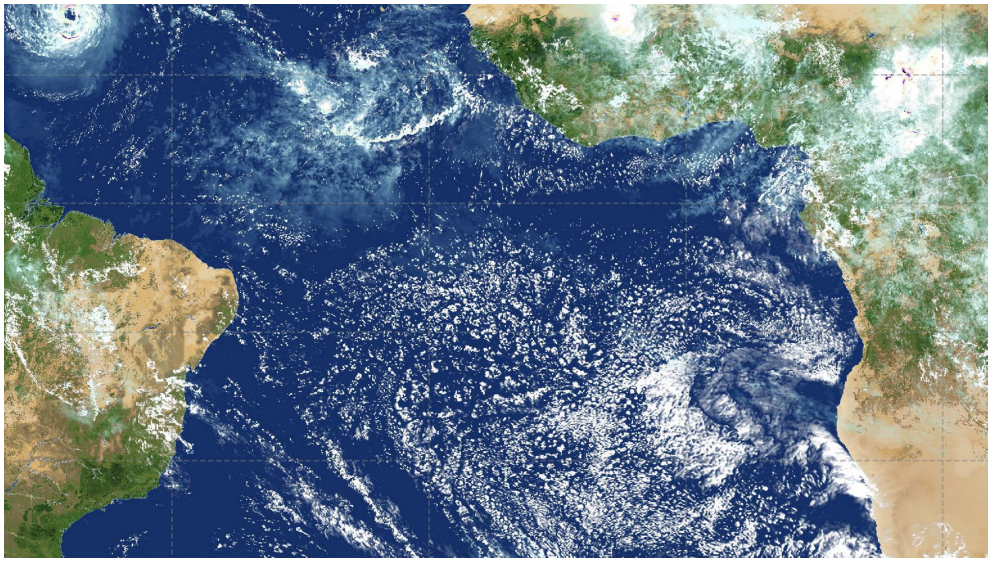
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Scientific Visualisation of High-resolution Climate Simulations

Semester Project

Claudio Cannizzaro

14.10.22



Supervisors:
C. Heim
Dr. P. Velasquez
Prof. Dr. C. Schär

D-MATH, D-USYS , ETH Zürich

Abstract

In this thesis a scientific visualization program was developed to produce realistic looking pictures from the output data of the COSMO model [1]. This software imitates satellite images to help visualize the complex weather conditions over the globe. It is highly customisable and gives satisfying results during different seasons and in most climate regions.

Contents

Chapter 1

Introduction

Climate and weather simulations are becoming very important as climate change progresses and extreme weather events occur more frequently (Carvalho 2020 [2], Luiz-Silva 2020 [3], Walsh et al. 2020 [4]). This thesis presents a visualization program based on Python 3 to create animations for high-resolution climate simulations, specifically for output of the non-hydro-static limited-area atmospheric model COSMO (Steiger 2020) [1]. Animations are extremely useful on the kilometer scales (spatial resolution), since convective clouds can be explicitly resolved and therefore better represented (Leutwyler 2017 [5]).

1.1 Motivation

When picturing weather and climate visualizations, satellite images come to mind at first. Even non-scientists are used to seeing them on the news and in weather forecasts. This leads to an intuitive understanding of satellite imagery by a broad group of people. Even though satellites provide images over large areas at high temporal resolution, they cannot barely cover a single large area at the same time. Therefore, it is necessary to create a program that is able to reproduce satellite-like images that cover huge areas without temporal interruption. This would be useful for at least two reasons. Firstly, it helps in scientific communication. The visualization is intuitive and self-explanatory, instead of overwhelming the audience with many labels. This has the additional benefit that various variables can be shown on the same visualization. This thesis considers up to 15 variables for creating a single visualization depending on the chosen configuration. Secondly, it also helps climate and weather scientists assessing the simulated variables. Namely, using this program facilitates the comparison of model output against satellite images, i.e., it functions as a qualitative tool to visually validate simulations.

During previous work, a python program was developed to create realistic

looking figures of climate and weather simulations for the Atlantic region. This thesis aims at improving this program in terms of ease of use and applicability for various regions worldwide.

1.2 Outline

This thesis is structured as follows: Section 2 presents the methodology of the program and its application, Section 3 and 4 show discuss and results of the program, respectively, and section 5 discusses future improvements.

Chapter 2

Methods

This chapter explains the program that creates the visualization; moreover, it introduces the variables and parameters that are used. Furthermore, this chapter presents a guideline about how to use the program, which includes the set up and the corresponding source code. The figures shown in this thesis were created by running the program on the ETH Piz Daint super computer [6] where the data used by this program was created by the COSMO model.

2.1 How the code works

This section shows the code's structure and describes in detail how the plots are generated.

2.1.1 Structure

The source code is structured into three parts. Firstly, the Visualization class (vis) which is run by the main method. Secondly, there are the Visualization parameter class (VisParam) and the JSON configuration file which work similarly to a name list in climate and weather simulations. These concepts are explained in more detail below.

2.1.2 Visualization class

The Visualization class is the class which generates the visualizations. It is initialized with the JSON-file 2.1.4 and the VisParam-class 2.1.3. Listing 2.1 shows how the main visualization loop works as a pseudo code. It begins by reading in all the parameters specified in VisParam 2.1.3 and the JSON file 2.1.4. Next, it checks how often every variable should be read in and how often plots should be generated. This distinction is made since typically, some variables are written out with higher temporal resolution

than others, e.g., precipitation, hail and lightning. Note that in this work the term precipitation refers to the total precipitation, i.e., liquid and solid precipitation.

2.1.3 Visualization parameter class

This class is used for the visualization parameters. This class should be used by users who want to change the variable plotting bounds if there is for example precipitation happening in the simulation, but not showing visible enough in the visualization. Users can also change color maps, toggle grid lines, change font sizes of the grid lines and the datetime. It is also possible to change the parameters of the land surface function and to define special markers for hail and lightning according to 2.6.

2.1.4 JSON config file

The JSON config file is the file that a user has to change most often. It is strongly recommend creating a new JSON config file for every new visualization. The reason to use the JSON type is because it can be directly read in as an object into python. This means the user is not required to edit a python file directly and structural error are easily caught due to the strict JSON syntax.

The listing 2.3 shows an example JSON file. Enable_plotting is a container of booleans in which the user can toggle to plot the surface, clouds, snow, hail and lightning. Const_path is the direct path to the file containing the constant values for this simulation. Paths is a container of strings which should contain the paths of the folders where the NetCDF-files are where those specific variables are stored. Namely, those variables are TQI, TQC, W_SNOW, TOT_PREC, W_SO, LPI, DHAIL_MX and TQV. Instead of strings one can write null if the variable does not exist or is not used. Data_time_stepping is used to decide how often new data should be read in. The durations can be written in different formats, because the string will be interpreted nicely by the django library, but it is recommend sticking to the example below. Vis_duration and vis_start_time are used to determine how long the visualization should take and when it should start. OutputPath is the path where pictures are stored. Plotting_time_step is how often in simulation time you want to generate a new picture.

```
1 read_in_all_the_visualization_parameters(JSON file, Visualization
      Parameter class) # initialize the Visualization class
2
3 dt = find_min_timestep() # timestep of the visualization / the
    minimum data read time
4 timesteps = anim_duration / dt # number of time steps for data
    reading
```

```
data_read_intervals = data_read_intervals() # array of data reading intervals for each specific variable
plotting_frequency = plotting_interval / dt # determines at which time steps a new picture gets generated

loop over timesteps:

    loop over every variable:
        o read data if there is new data available and:
            o in first timestep
            o it is not plotting time
                o concatenate multiple data reads together ( used for hail, lightning and precipitation)
                o skip to next time step and update time

    update time
    plot_all() # function that plots all variables in the correct order

    loop over every variable:
        o read data if there is new data available
```

Listing 2.1: Pseudo code: main visualization loop

The pseudo code (Listing 2.2) shows how the visualization is built up: (i) It starts drawing surface features such as land, oceans, elevation, ice and snow, (ii) the cloud water components, such as cloud water and ice are superposed, (iii) the weather phenomena rain, hail and lightning are visualized on top of the figure, (iv) the figure is saved in a jpg format and afterwards deleted internally to prevent memory leaks. The jpg format is used, since it is a portable format and storage efficient. The next sections go into detail on how the different layers are built up. Each section provides a visualization on what is added.

```
plot_all():
    o define figure, axis and grid lines

    o check for every variable if it needs to be plotted
        plot_land_ocean
        plot_elevation
        plot_ice
        plot_snow
        plot_TQV
        plot_TQC
        plot_TQI
        plot_TOT_PREC
        plot_DHAIL_MX
        plot_LPI

    o save figure as .jpg
    o delete figure
```

Listing 2.2: Pseudo code: plot all function, all functions called here are explained in detail in

their respective sections

2.1.5 Figure Initialization

The figure is initialized in the following manner. The figure's size is inferred by the underlying data. This is done by reading in the dimensions of the NetCDF-files. It is also possible to specify this manually. The axes are adjusted by the rotated pole transformation which are used by all the COSMO simulations used in this work, thus the program reads in the rotated pole automatically from the COSMO output files. Further, grid lines can be displayed which show longitude and latitude. The cartopy library is used for these two features [7].

2.1.6 plot_land_ocean

This function plots the earth's surface using only the information from the COSMO-model, i.e., constant and simulated fields. Note that some of the initial conditions of the COSMO-model come from satellite-based observations. This was the most involved section and required a lot of tweaking, in order to work in multiple climate regions with one single configuration. The original idea was developed by C. Heim and further elaborated in this project. Fundamentally, the idea is to modulate the land-surface color depending on the soil moisture content. More specifically, the following variables stored in the nc-files are considered: soil moisture content of soil levels, the fraction of evergreen and deciduous forest, the diffuse albedo and the location of the oceans. These variables are combined in the code of the plot_land_ocean() and load_const_data() functions in the Visualization class. With the result of this calculation, it was possible to color the land surface from very dry, beige, sand colors to moist, green, forest colors. The ocean is colored in blue uniformly. The result of this can be seen in Figure 2.1.

2.1.7 Plotting standard variables

Most work for plotting is being done by the plot_var() function. The function takes the variable to be plotted, a corresponding color map and variable plotting bounds as arguments. These are then given to the pcolormesh function which plots the variable. Another argument is whether to plot the variable over the the whole domain which is the default or only over land or ocean. This is useful for land-surface variables that should not be plotted over the ocean.

In the following subsections I will go into detail which functions called in the plot_all function (Listing 2.2) invoke this function.

Figure 2.1: Surface plot of Europe in summer with the corresponding color map for the surface

plot_elevation

One issue that was visible was that mountains were too green. Thus it was necessary to create a function which considers the elevation information and applies drier colors to the areas with higher elevation. (Shown in Fig. 2.2)

Figure 2.2: Surface plot of the Alps in summer without and with the elevation map

plot_ice

The program also plots the permanent ice cover such as glaciers which are present in the constant files as a soil type, as illustrated in Figure 2.3.

Figure 2.3: Permanent ice cover of the Alps

plot_snow

The simulated snow is also visualized. (Fig. 2.4)

Figure 2.4: The Alps in winter with snow cover, clouds are also visible

Clouds

Clouds consist of cloud water (TQC) and cloud ice (TQI) which are visualized in white and light-blue-to-white color, respectively. Note that the vertical integral of these quantities is plotted instead of the three-dimensional field. In principal, any other two-dimensional variable can be added effortlessly, since the plotting methods are already implemented i.e. the plot_var() function 2.1.7. Figure 2.5 shows the added TQC and TQI.

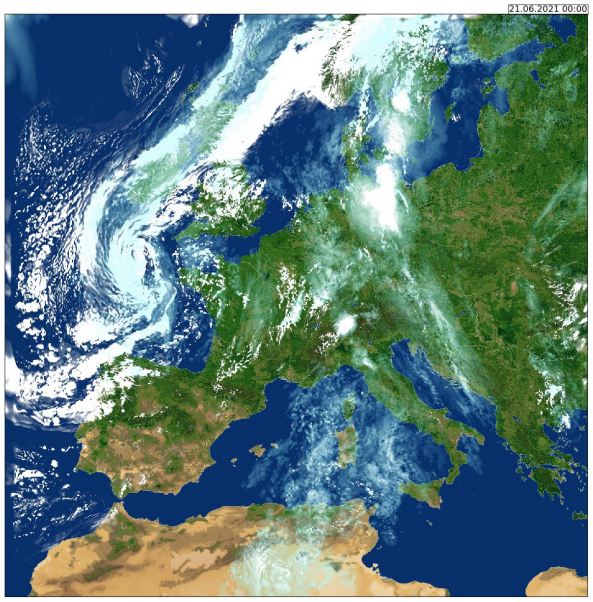


Figure 2.5: Europe in summer with clouds (TQC and TQI)

plot_TOT_PREC

As mentioned in section 2.1.2, precipitation information can be read more often than being plotted; therefore, depending on the temporal resolution of plotting the pictures, the precipitation information can be summed up in between plotting time steps to show an accumulated value. Note that precipitation falls within and below clouds and therefore being barely distinguished in the visible range (human eye's range) from satellites. To show the precipitation intensity, the precipitation is colored from yellow over red to blue (Fig. 2.6)

2.1.8 plotting marker variables

The plotting_marker_variables-function is the second plotting method. It is used for variables that are plotted as a marker when occurring like hail or lightning. This is done regardless of the hail sizes or lightning intensity. Here a scatter plot with custom markers is used to visualize lightning and hail. The lighting marker was designed by me. The hail marker was taken from The Comprehensive LATEX Symbol List [8] and generated an svg-format picture and converted it to a matplotlib path according to an online tutorial [9].

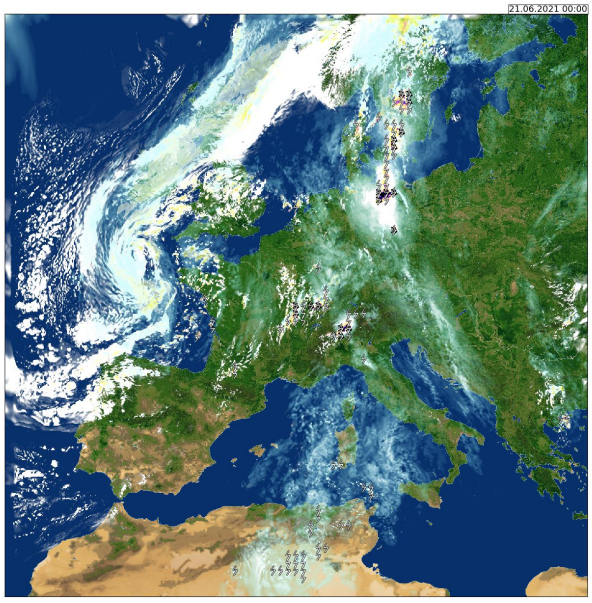


Figure 2.6: Europe in Summer with rain (here TOT_PREC), lightning and hail added.

2.1.9 Additionally used software

There are other libraries heavily used in the program: matplotlib [10], xarray [11] and Django [12]. Matplotlib is the plotting library for python and thus used for my visualization. Xarray permits to easily deal which the large number and size of the NetCDF-files. It even has some light wrappers to matplotlib which make plotting the data set really simple. Django has a method to read in durations and transform them into python datetime objects. The EOSDIS Worldview website [13] shows current and past satellite

pictures which played a key role in tweaking the color maps and validating the visualization, e.g., when Europe was covered by snow. The last software is ffmpeg which creates videos out of the generated pictures. [14]

2.1.10 Important notes

Note that the program assumes that the NetCDF-files are named in the following way: ”lffd” + YYYYmmddHHMMSS + ”.nc. Additionally the data time steppings should have to have a common divisor, because otherwise the program might skip some files, since the program takes the smallest data time step as the plotting time step. Thus if the smallest data time step is 10 min and another one is 15 min, the 15 min one will only be updated every 30 min. Furthermore the data time steppings should be smaller than the plotting frequency for precipitation, hail and lightning, because it produces a smoother video.

```
{
  "enable_plotting": {
    "TQI": true,
    "TQC": true,
    "W_SNOW": true,
    "TOT_PREC": true,
    "W_SO": true,
    "LPI": true,
    "DHAIL_MX": true,
    "TQV": false
  },
  "const_path": "~/atmosthesis/lffd20210522000000c.nc
   ",
  "paths": {
    "TQI": "~/atmosthesis/lightning",
    "TQC": "~/atmosthesis/lightning",
    "W_SNOW": "~/atmosthesis",
    "TOT_PREC": "~/atmosthesis/hail",
    "W_SO": "~/atmosthesis",
    "DHAIL_MX": "~/atmosthesis/hail",
    "LPI": "~/atmosthesis/lightning",
    "TQV": null
  },
  "data_time_stepping": {
    "TQV": null,
    "TQI": "01:00:00",
    "TQC": "01:00:00",
    "W_SNOW": "1 day, 00:00:00",
    "TOT_PREC": "00:5:00",
    "W_SO": "1 day, 00:00:00",
    "DHAIL_MX": "00:5:00",
    "LPI": "01:00:00"
  },
  "vis_duration": "00:05:00",
  "vis_start_date": "2021-06-21 00:00:00"
  "plotting_interval": "00:01:00",
  "outputPath": "./pics",

}
```

Listing 2.3: JSON config file example

2.2 How to use the program

Five pieces of software are needed to use the program.

- visualize_climate_simulation.py
- YourJSONFile.json
- Visualization.py
- VisParam.py
- a python environment with the necessary libraries installed

2.2.1 Setting up the environment

To set up the environment files described above need to be downloaded. These files can be found at [15]. Put all the files in a new folder. Alternatively there is github repository linked there as well.

Next, a conda environment should be set up [16]. In the git repository, there are two .yml files for Piz Daint and macOS. To create a new environment use the following command.

```
conda env create -f NAME_OF_YML_FILE.yml
```

Next one needs to create a JSON-file and to fill in the variables pertaining to your simulation (explained in 2.1.4). This determines what would be visualized.

2.2.2 Running the code

The next step is running the program itself with this command:

```
python3 visualize_climate_simulation.py YourJSONFile.json
```

After the program has finished running, one creates an animation from the created pictures. This can be done by running the following command in the folder where the pictures are created. This should work cleanly, since the files are inherently sorted by their time step as a file name. Alternatively, you can also download the pictures to your local computer as the pictures do not exceed 1MB per file. Thus one needs $\approx$ 500 MB storage for an animation of 20 days with hourly frames.

```
 ffmpeg -framerate 20  -pattern_type glob -i '*.jpg' \
-c:v libx264 -pix_fmt yuv420p -vf "pad=ceil(iw/2)*2:ceil(ih/2)*2" \
ANIMATION_NAME.mp4
```

Here the frame rate is set to 20 frames per seconds, but this can easily be changed by changing the number after ”-framerate”.

2.2.3 Additional information

The run time is strongly dependent on the size of the data provided by the COSMO-model. For example the pictures for Europe shown above took approximately 60s on Piz Daint per picture. Note that if this program is submitted as a job on Piz Daint it does not have access to the $PROJECT-folder. Thus NetCDF-files and generated pictures need to be in $SCRATCH. Further, many tweaks can be made to the visualization itself by changing the VisParam 2.1.3 or even the Visualization 2.1.2 class. Two examples of this are shown in Fig. 2.7. One option is to enable grid lines for latitude and longitude. If one does not need this option, turning it off is recommended to increase realism. There is also the option to change the plotting range boundaries of your variables. For instance, this needs to be done if the plotting range does not match the precipitation rate, which could happen when the setup was previously done for a region with higher precipitation rates.

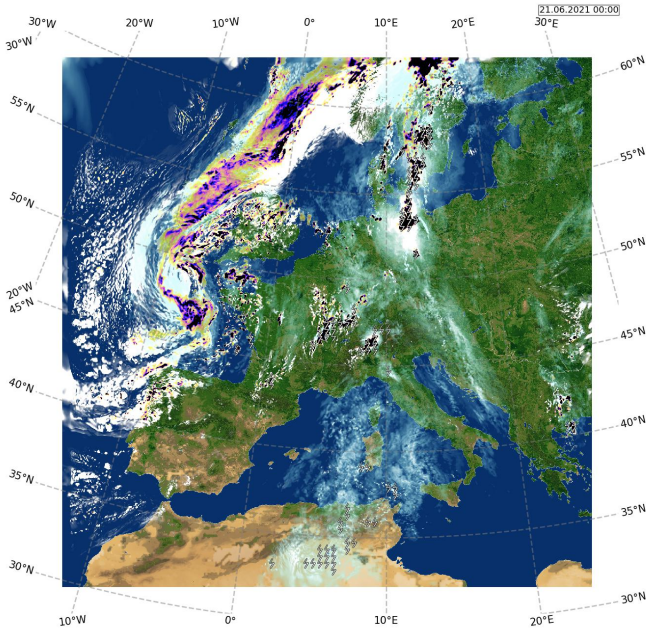


Figure 2.7: Europe in Summer with rain being plotted more visibly and added latitude and longitude grid lines.

Chapter 3

Results

This chapter shows five examples of the visualization in different regions. The goal here is to show the application of the program on different climate regions. There are animated visualizations stored online at Zenodo [15]. These videos are in the examples folder. The corresponding JSON-files is in the json-files folder. For all these examples the presets were not changed, except the turning on or off the grid lines for different regions. The first example is the Arabian Peninsula which is a very arid region (Fig. 3.1). The second example shows a large part of Asia covering the Himalayas (Fig. 3.2). Next, Figure 3.3 and 3.4 show Europe for summer and winter conditions, respectively. Lastly, the Tropical Atlantic is shown which includes large parts of Africa and Brazil (Fig. 3.5).

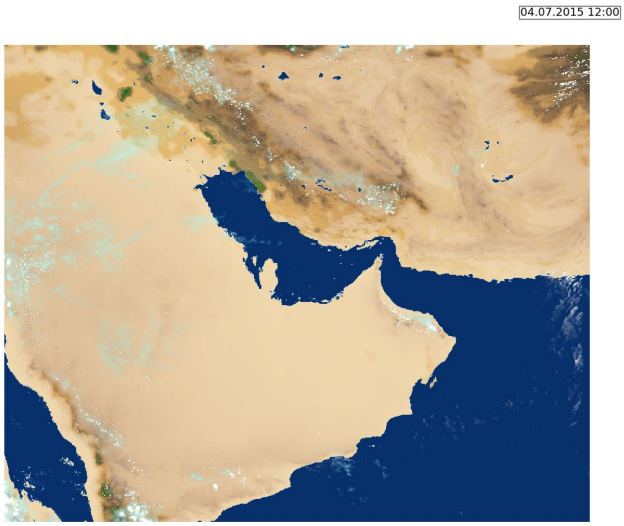


Figure 3.1: Visualization of the Arabian Peninsula

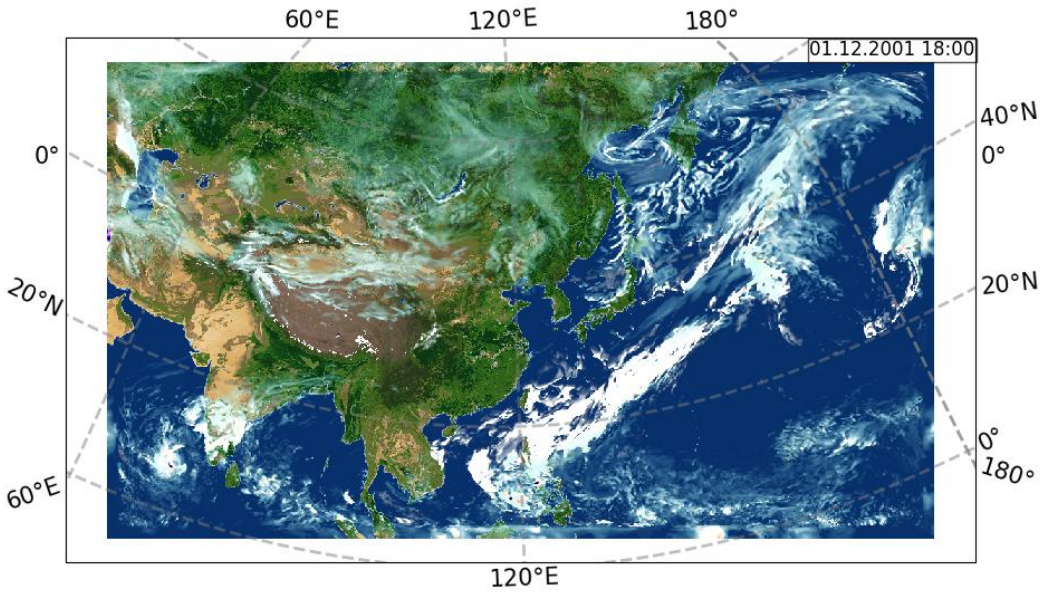


Figure 3.2: Visualization of Asia

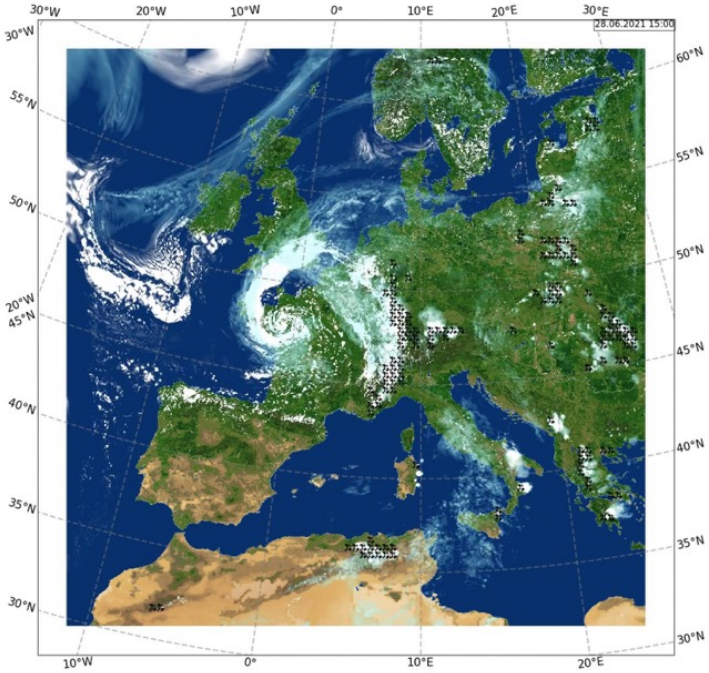


Figure 3.3: Visualization of Europe in summer with a hail storm

Figure 3.4: Visualization of Europe in winter

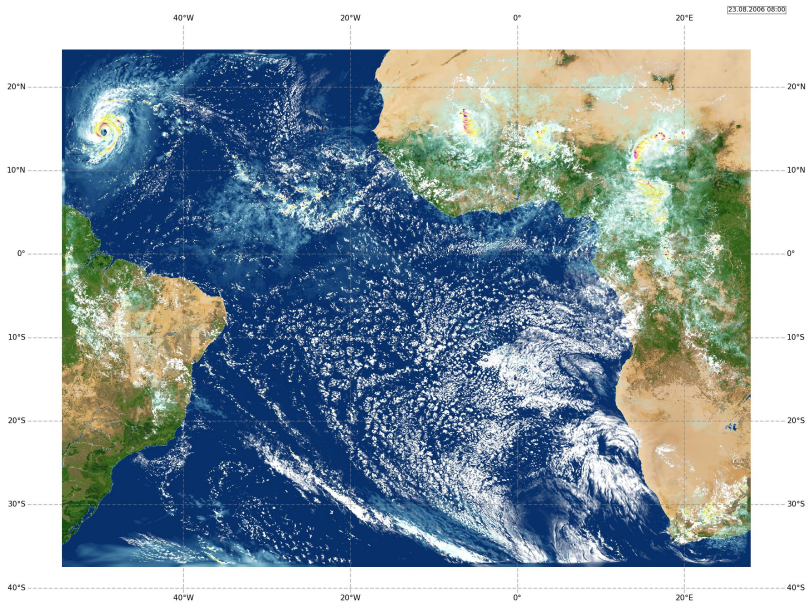


Figure 3.5: Visualization of the tropical Atlantic region

Chapter 4

Discussion and summary

This thesis aims at providing a program to create satellite-like visualizations using high-resolution climate simulations. The created visualizations look realistic (chapter 3). Therefore, the program's application on different climate regions works properly and does not need to be further adapted. Moreover, the original program made by C. Heim was improved, which was able to produced realistic visualizations only for the Atlantic region (Fig. 4.1). The current program is more user-friendly, and it renders the surface across different climate regions more accurately.

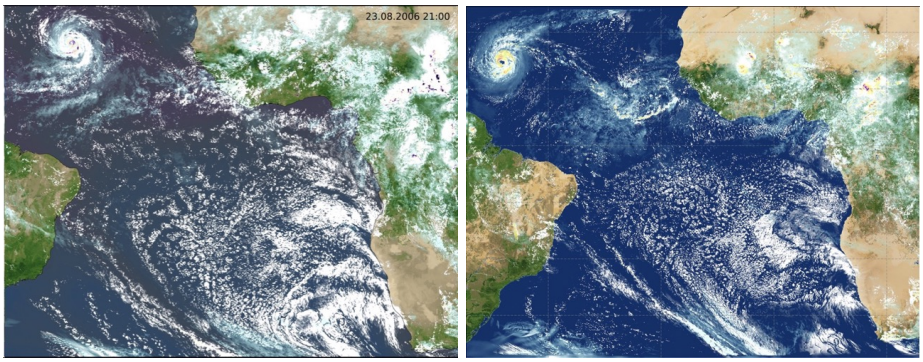


Figure 4.1: The original version of the visualization by Christoph Heim (left) compared to my visualization (right). Note that in the left panel, vertically integrated water vapor content over oceans is colored in purple, which is not done in the right panel.

When comparing the visualization of Brazil with satellite pictures on the same day (Fig. 4.2) we can see that the pictures match reasonably well, particularly surface conditions. It is arid where it is supposed to be and green where there is vegetation. Rivers also show up even though they are not explicitly modeled. Additionally, since the visualization tries to be applicable to the whole globe a few limitations can be seen in this figure, as

well. One is that the hue of the surface is not correct. Brazil's desert regions have a darker and redder hue in reality. Another one is that the ocean are just a single color without any gradient as is usually observed in satellite images.

Figure 4.3 compares satellite pictures provided by [13] and a visualization. We observe that the pictures and the visualizations look very similar. The limitations for the gulf are the same as for Brazil. Another issue arises when plotting for Europe. In satellite images the colors of the vegetation are less saturated. This makes the winter surface greener in the visualization compared to satellite images. This could be due to the fact that temperatures (soil and near-surface temperature) are not considered in this program.

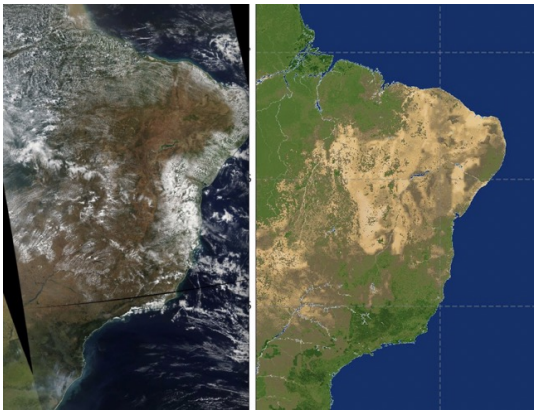


Figure 4.2: Brazil. Comparison between satellite picture [13] and surface visualization

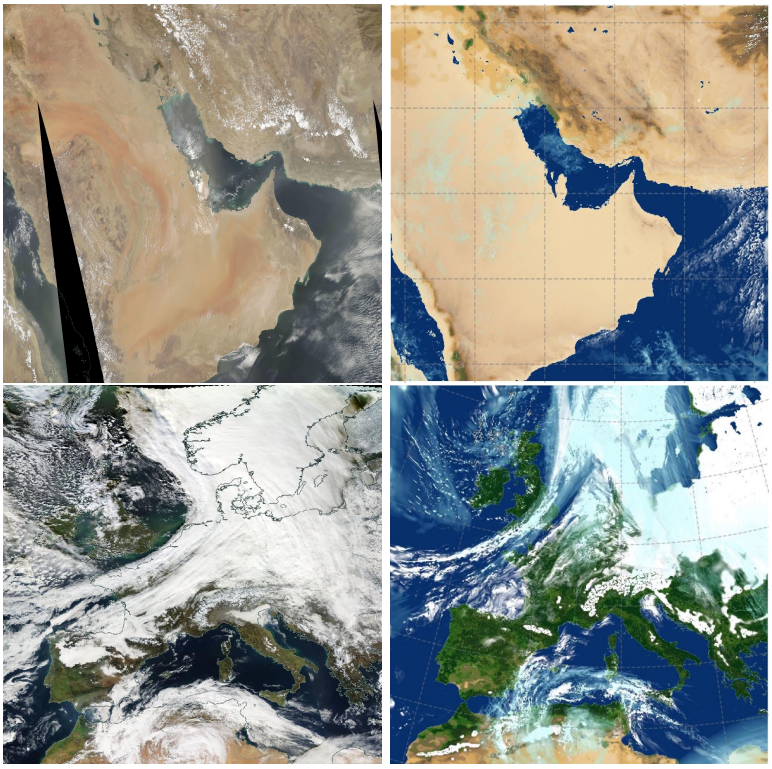


Figure 4.3: Arabian Peninsula (04.07.2015) / Europe (20.01.2006) . Comparison between satellite pictures [13] and visualization

Chapter 5

Future improvements

There is still room for further improvements of the program. The following steps are left to future work due to time constraints in this project:

Finding a method to improve the visualization of surfaces with redder hues. This could possibly be done by adding an additional parameter to the surface calculation that is already present in the NetCDF-files, or by manually creating a map where these kinds of surface colors occur. During winter, particularly in mid latitudes, the surface conditions could be more pale. A further step would be to consider the effects of temperature. With this addition the lessened growth of plants in winter could probably be visualized. Additionally, the ocean looks bland in the visualization because it is just a single color everywhere. This issue could be improved by displaying the ocean in different shades of blues depending on ocean temperature, near-surface wind velocity, and solar zenith angle.

Furthermore, a testsuite could be added to the code. In this testsuite, a NetCDF-file and a JSON file are provided to test if the environment is set up correctly. It could also be used to debug the program during future developments and to conveniently test the outcome of such developments (e.g., when changing the color maps).

Lastly, the most impactful fix would be to improve performance of the visualization. The bottleneck is the rendering of the pictures in matplotlib, whereas the reading of data is comparably faster. Thus having the rendering done in parallel would improve the performance. A first version of this has been implemented in the Visualization-class, but it ran into race conditions (a common bug in multi-core programs with parallel threads trying to access the same data simultaneously). Thus, to implement this change, one would need to inspect the program variables and prevent processes of changing each others data.

Bibliography

[1] C. Steger and E. Bucchignani, "Regional climate modelling with COSMO-CLM: History and perspectives," *Atmosphere*, vol. 11, no. 11, p. 1250, Nov. 2020, Number: 11 Publisher: Multidisciplinary Digital Publishing Institute, ISSN: 2073-4433. DOI: 10.3390/atmos11111250. [Online]. Available: https://www.mdpi.com/2073-4433/11/11/1250 (visited on 10/20/2022).

[2] L. M. V. Carvalho, "Assessing precipitation trends in the americas with historical data: A review," *WIREs Climate Change*, vol. 11, no. 2, e627, 2020, ISSN: 1757-7799. DOI: 10.1002/wcc.627. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/wcc.627 (visited on 10/20/2022).

[3] W. Luiz-Silva, A. C. Oscar-Júnior, I. F. A. Cavalcanti, and F. Treistman, "An overview of precipitation climatology in brazil: Space-time variability of frequency and intensity associated with atmospheric systems," *Hydrological Sciences Journal*, vol. 66, no. 2, pp. 289–308, Jan. 25, 2021, Publisher: Taylor & Francis, ISSN: 0262-6667. DOI: 10.1080/02626667.2020.1863969. [Online]. Available: https://doi.org/10.1080/02626667.2020.1863969 (visited on 10/20/2022).

[4] J. E. Walsh, T. J. Ballinger, E. S. Euskirchen, *et al.*, "Extreme weather and climate events in northern areas: A review," *Earth-Science Reviews*, vol. 209, p. 103 324, Oct. 1, 2020, ISSN: 0012-8252. DOI: 10.1016/j.earscirev.2020.103324. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0012825220303706 (visited on 10/20/2022).

[5] D. Leutwyler, D. Lüthi, N. Ban, O. Fuhrer, and C. Schär, "Evaluation of the convection-resolving climate modeling approach on continental scales," *Journal of Geophysical Research: Atmospheres*, vol. 122, no. 10, pp. 5237–5258, 2017, ISSN: 2169-8996. DOI: 10.1002/2016JD026013. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/2016JD026013 (visited on 10/20/2022).

[6] "Piz daint — CSCS." (2022), [Online]. Available: https://www.cscs.ch/computers/piz-daint/ (visited on 10/20/2022).

[7] *Cartopy 0.21.0 documentation*, 2022. [Online]. Available: https://scitools.org.uk/cartopy/docs/latest/ (visited on 10/11/2022).

[8] S. Pakin. "CTAN: /tex-archive/info/symbols/comprehensive." (2021), [Online]. Available: https://www.ctan.org/tex-archive/info/symbols/comprehensive (visited on 10/12/2022).

[9] P. Smith, *Custom markers*, 2021. [Online]. Available: https://petercbsmith.github.io/marker-tutorial.html (visited on 10/10/2022).

[10] *Matplotlib — visualization with python*, 2022. [Online]. Available: https://matplotlib.org/ (visited on 10/12/2022).

[11] *Xarray documentation*, 2022. [Online]. Available: https://docs.xarray.dev/en/stable/ (visited on 10/11/2022).

[12] *Django — the web framework for perfectionists with deadlines*, 2022. [Online]. Available: https://www.djangoproject.com/ (visited on 10/11/2022).

[13] "EOSDIS worldview." (2022), [Online]. Available: https://worldview.earthdata.nasa.gov (visited on 05/19/2022).

[14] *FFmpeg*, 2022. [Online]. Available: https://ffmpeg.org/ (visited on 10/12/2022).

[15] C. Cannizzaro, *Scientific-visualisation-of-high-resolution-climate-simulations*, Oct. 15, 2022. DOI: 10.5281/zenodo.7211807. [Online]. Available: https://zenodo.org/record/7211807 (visited on 10/15/2022).

[16] *Conda — conda documentation*, 2022. [Online]. Available: https://docs.conda.io/en/latest/ (visited on 10/12/2022).

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Scientific visualisation of high-resolution climate simulations

Authored by (in block letters):

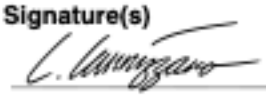
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Cannizzaro	Claudio

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Lachen, 14.11.2022	

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.