Visualization of downdraft from AW101 SAR Queen helicopter

Kristoffer Berg Wilhelmsen

UiT - The Arctic University of Norway, P.O. Box 385, N-8505 Narvik, Norway

Submitted 2022-12-19

Abstract

The SAR Queen is a rescue helicopter set to replace the Sea King. Despite being an upgrade, it has its flaws. The helicopter creates a significantly stronger downdraft than its predecessor's, which can pose a danger to human life. Using visualization techniques from the Applied Computer Science curriculum, the objective is to identify how the downdraft affects pedestrian areas and the patient entrance of the hospital tunnel at the University Hospital of Northern Norway in Tromsø. The results from the techniques applied show that the downdraft has a varying effect on the areas. The pedestrian areas are affected the most during the initial and the final stages of landing, while the patient entrance is affected the most at a point in between.

Keywords: Fluid Mechanics; SAR Queen; AW101; Sea King; FLOW-3D; Python; Godot; imageio

1 Introduction

The AW101, domestically known as the Search & Rescue (SAR) Queen in Norway, is a helicopter specialized for use in rescue missions (Sundsbø, 2022). It is a replacement for Norway's previous rescue helicopter, known as the Sea King. The SAR Queen, seen in Figure 1, is designed to fly long distances, withstand the arctic climate, strong winds and rough seas.

Figure 1. The AW101 helicopter, also known as the SAR Queen (Jarslett, 2021).

Unfortunately, the SAR Queen also has its downsides. The downdraft created by the helicopter is much more significant than its predecessor's, and can be a potential danger for persons in proximity. An article published by NRK (Toftaker and Kleven, 2020) mentions test landings from 2019 at St. Olav's University Hospital and Haukeland University Hospital, where the conditions were deemed unsafe for persons close by during landing, as a direct consequence of the helicopter's downdraft.

The objective of the work is to analyze the given flow problem, which takes place at the University Hospital of Northern Norway in Tromsø. Results from a numerical simulation using sensor data from a real landing was provided by professor Per-Arne Sundsbø, along with the 3D models used in the simulation. The results from the simulation and the models used are attached in Appendices A.1 and A.2.4. By applying modern visualization techniques from the Applied Computer Science curriculum to create alternate visualizations, the analysis will look to determine how the helicopter's downdraft affects pedestrians in proximity and the hospital tunnel's patient entrance during landing.

2 Material & Methods

Various pieces of software have been used in order to perform the analysis of the given problem. This section describes which software was used, to what extent, and how the results were obtained.

2.1 Blender

Blender (Blender, 2022) is a modelling application that can be used to create and animate 3D models. In order to use the supplied 3D models for animation purposes, their formats must be compatible with the software we intend on using. In this case we will use Godot to create animations, which does not currently support the models' *stl* format. Godot's preferred format for 3D models is *glTF* (.glb file extension). Converting the models can easily be done by importing them to Blender, and exporting them to the desired format.

2.2 FLOW-3D

FLOW-3D (FLOW-3D, 2022a) is an application used in computational fluid dynamics that has produced the raw data which forms the basis for the animations in the report. Due to the application being quite advanced, its use in this report outside providing data has been limited. It is also important to note that the application has not been a part of the curriculum, and licenses were not facilitated for use on personal computers. Thus, due to the short timeframe of this project, its limited use is reasonable. It has however also been used to create a similar animation to what we seek to obtain as a part of the work's main objective.

The raw data exported from FLOW-3D contains XYZ-coordinates corresponding to individual cells in the mesh pictured in Figure 2, with associated values for volume fraction (float) and velocity magnitude (UVW-vector). Despite reducing the area that data is exported from drastically, each text file containing data for a single time step corresponded to ∼78 megabytes. The timeframe we want to animate is approximately 14.4 seconds long, with a step size of 0.4. This implies that there will be ∼36 different time steps, setting the cumulative file size to ∼2.8 gigabytes. In our case, a file of such a large size is not only undesirable to work with, but the format of its data must also be processed.

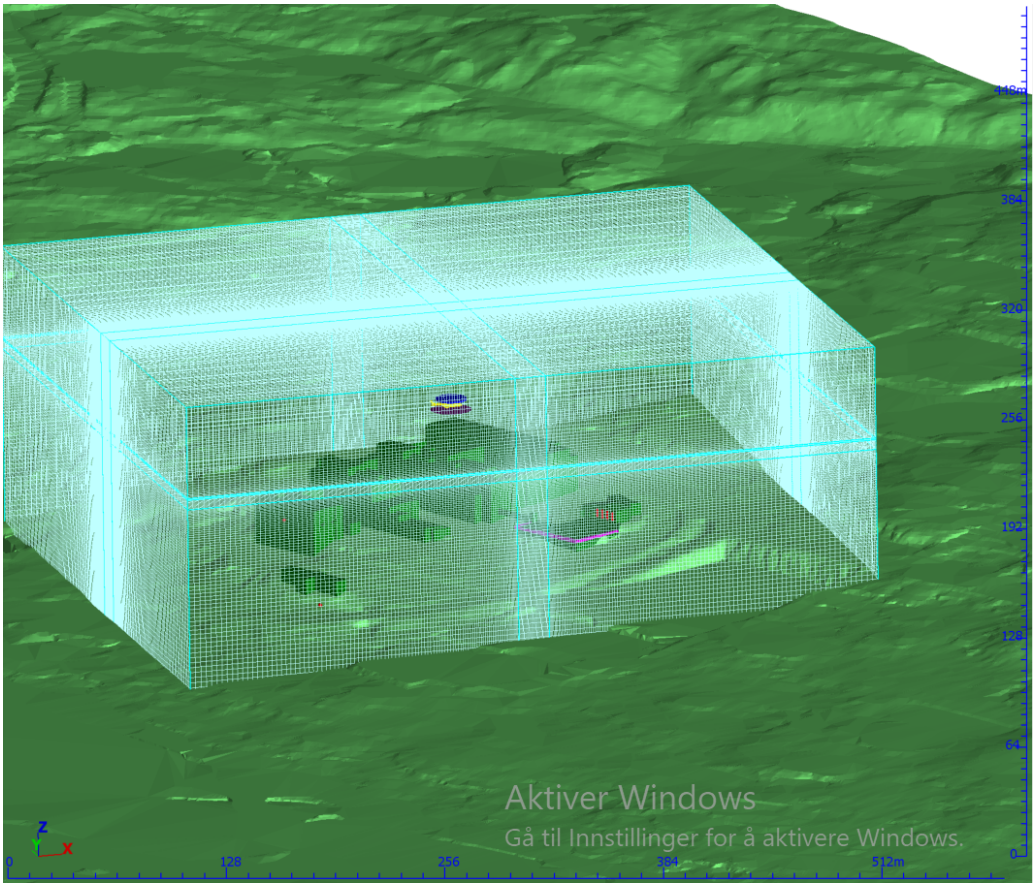

Figure 2. Model setup in FLOW-3D.

2.3 Python

Python (Python, 2022) is a programming language that is frequently used within data science, making it an excellent choice to further process the FLOW-3D data. As explained in Section 2.2, the FLOW-3D output corresponds to individual cells in the mesh from Figure 2. This includes cells that are not a part of the surface, which we are not interested in.

To filter out non-surface values, we can use the associated volume fraction. This is a float value from 0.0 to 1.0, which specifies how much of the cell is filled. If the volume fraction is 0.0, the cell is empty, hence is in the air. A volume fraction of 1.0 means that the cell is below the surface, as the entire cell is filled. To find values at the surface, we need to find XYZ-coordinates with an associated volume fraction that is in between.

Each cell's velocity magnitude is given in vector form, as mentioned in Section 2.2. The color of a given point on the surface solely depends on the length of this vector ($\sqrt{u^2 + v^2 + w^2}$), which means that we can calculate it at this step to further reduce the size of the processed file.

After filtering out the non-surface values and calculating the length of the velocity magnitude vectors by using the script in Appendix A.2.1, the file size for a single time step is reduced by 99% ($\sim$78 megabytes $\rightarrow\sim$ 600 kilobytes).

Using the processed data, we can create a texture to color the corresponding areas on the 3D models. A texture is simply an image with color values. To create the texture, we must determine its dimensions. First, we use the script in Appendix A.2.2 to find the minimum and maximum value for the X- and Y-direction, as well as the minimum distance between values in the data set. By finding the difference between the minimum and maximum value in both directions and dividing it by the minimum distance, we find the texture dimensions. The minimum distance is a consequence of the mesh having more detail around the helipad. In this case the calculations are as follows:

- X-min $= -60.43$, X-max $= 53.56$, X-diff ≈ 114
- Y-min $= -61.22$, Y-max $= 66.37$, Y-diff ≈ 128
- Minimum distance between values $= 0.62$
- X $= \frac{114}{0.62} \approx 184$, Y $= \frac{128}{0.62} \approx 206$

Note that X-diff, Y-diff, X and Y have been cast to integers due to the fact that image dimensions can not be floats. The script in Appendix A.2.3 creates a NumPy (NumPy, 2022) matrix according to the calculated X and Y dimensions. Subsequently, the processed data containing XYZ-coordinates and velocity magnitude is read and placed in the NumPy matrix at indices relative to where the coordinates would be on the original FLOW-3D surface.

To calculate the texture dimensions, we used the smallest distance between values. This implies that there are greater distances in the data, resulting in gaps between indices in the texture. This is illustrated in Figure 3, which has been generated using the Python library imageio (imageio, 2022).

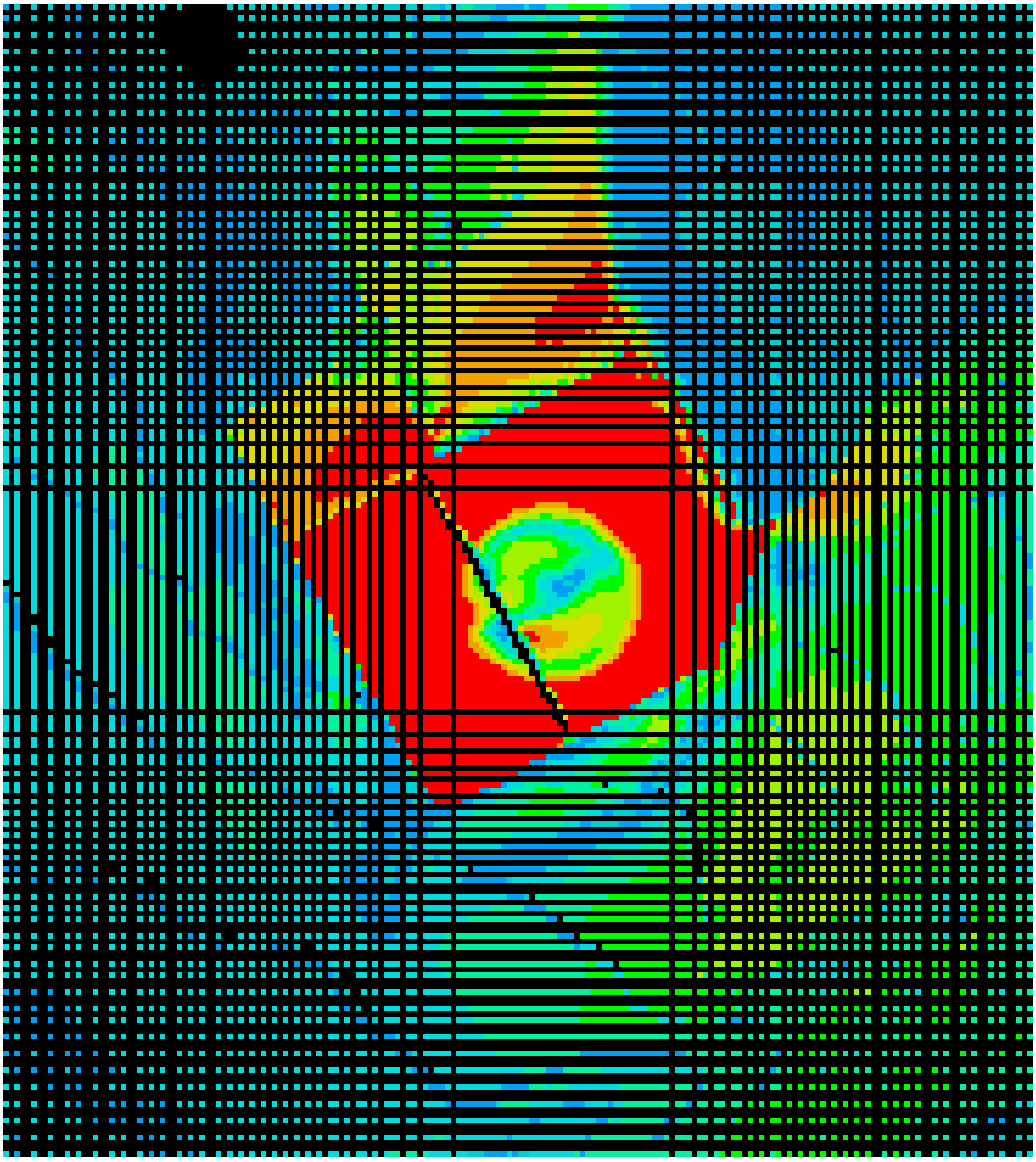

Figure 3. Texture with gaps in between indices.

A possible way of filling in the gaps is to copy nearby values. We can do this by sequentially iterating over each axis in the NumPy matrix, setting each empty value equal to the closest non-empty value. Figure 4 illustrates a complete texture where empty indices have been filled.

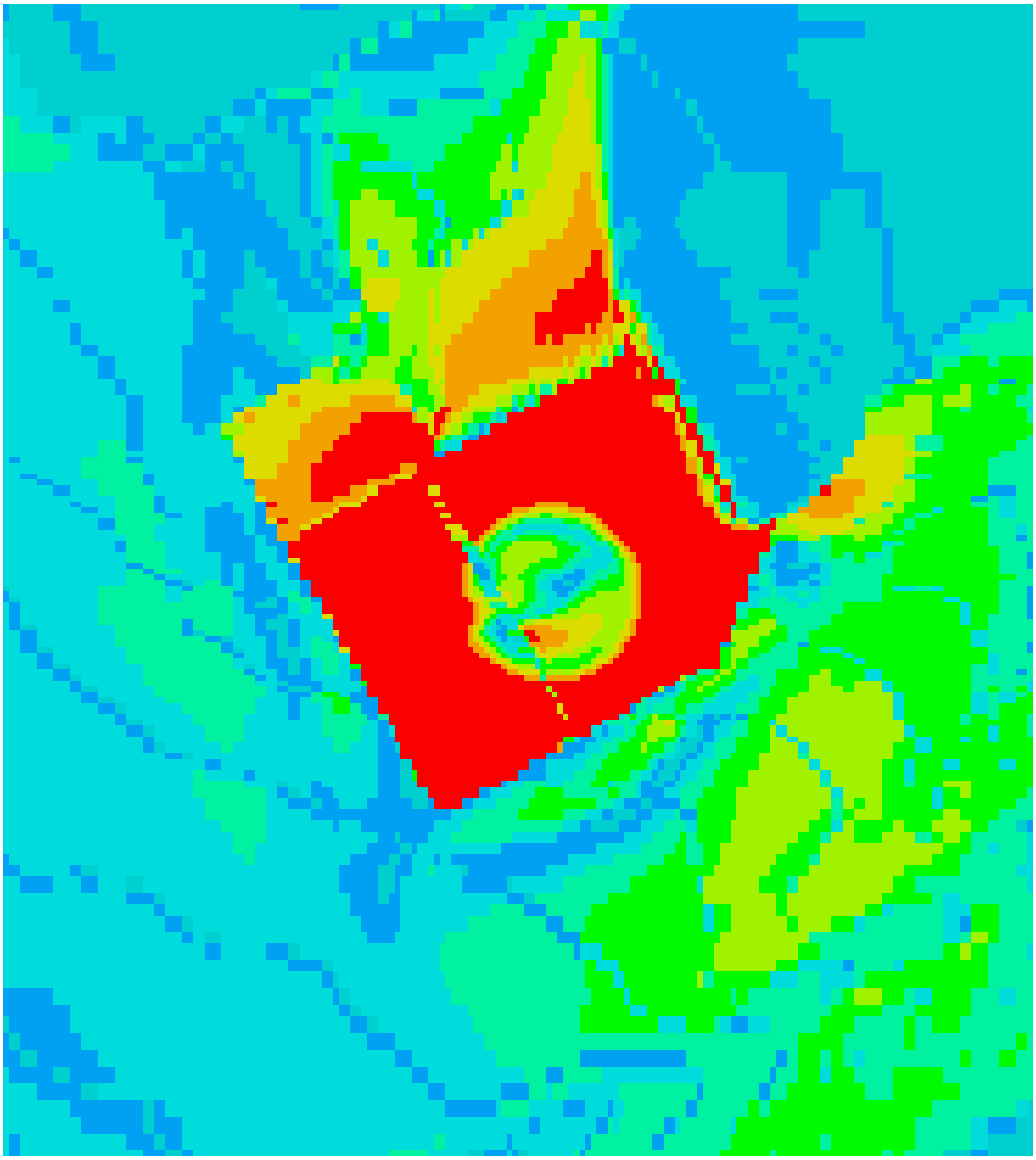

Figure 4. Example of a texture created in Python.

2.4 Godot

Godot (Godot, 2022d) is a game engine which primarily is used for game development. However, we can also use Godot to create animations. We set up the initial environment by importing the models converted in Section 2.1. Additionally, we import the textures created in Section 2.3.

Tween (Godot, 2022c) and Timer (Godot, 2022b) are two types of objects in Godot. A Tween can be used to interpolate values (e.g. movement), and a Timer can be set to execute code when its timer expires. To simulate the movement of the helicopter, a Tween interpolates the values in between the helicopter's initial and final position from 0 to 14.4 seconds. The Timer triggers every 0.4 seconds (equivalent to a time step), and updates the texture to the corresponding time step.

A shader (Godot, 2022a) is a computer program that operates on the graphics card, performing tasks such as setting colors. To map the current texture to the buildings and the terrain, we can make use of shaders. The shader program operates in two steps. First, the vertex shader is executed, allowing the program to manipulate individual vertices. Here, we store the current position. Next, the fragment shader is executed, which colors individual pixels between vertices. In the fragment shader we check if the current position is within a set of bounds we define. Note that in this case, the data from FLOW-3D did not translate well to Godot. As such, the bounds for the texture have been determined by trial and error. Lastly, the pixels are colored by translating the current position to UV-values.

These values range from 0.0 to 1.0 (floats), and correspond to a color on the texture.

The Godot project's repository can be found in Appendix A.3.

2.5 OBS Studio

OBS Studio (OBS, 2022) is a multipurpose application that can be used both for livestreaming and video recording. To capture the animations made throughout the project, OBS Studio has been used to record videos, which have then been uploaded to YouTube.

3 Results & Discussion

The main results from the project work are the Godot animations. Supplementary animations from FLOW-3D have been added for comparisons.

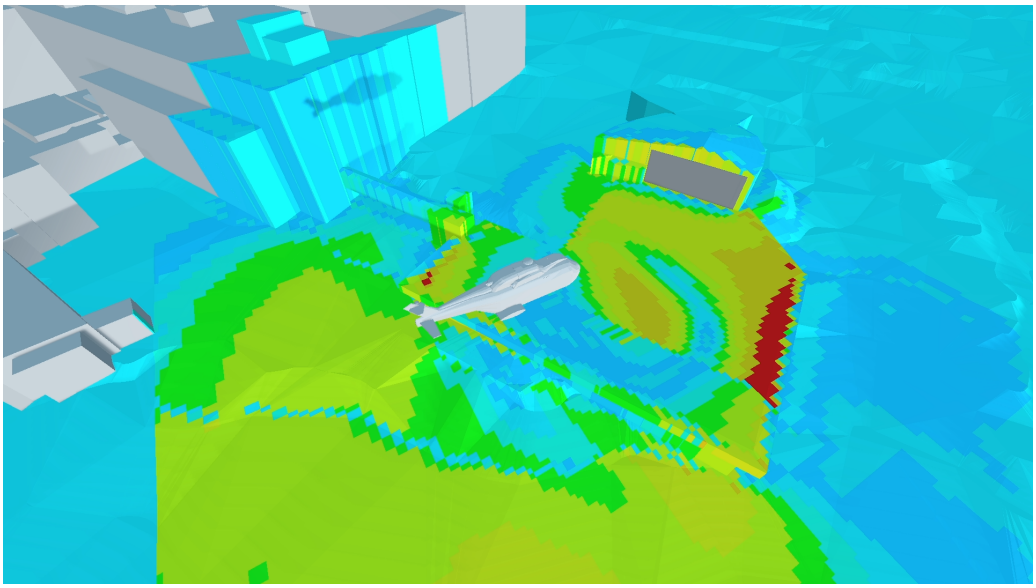

Figure 5. Start of Godot animation with the camera attached to the helicopter.

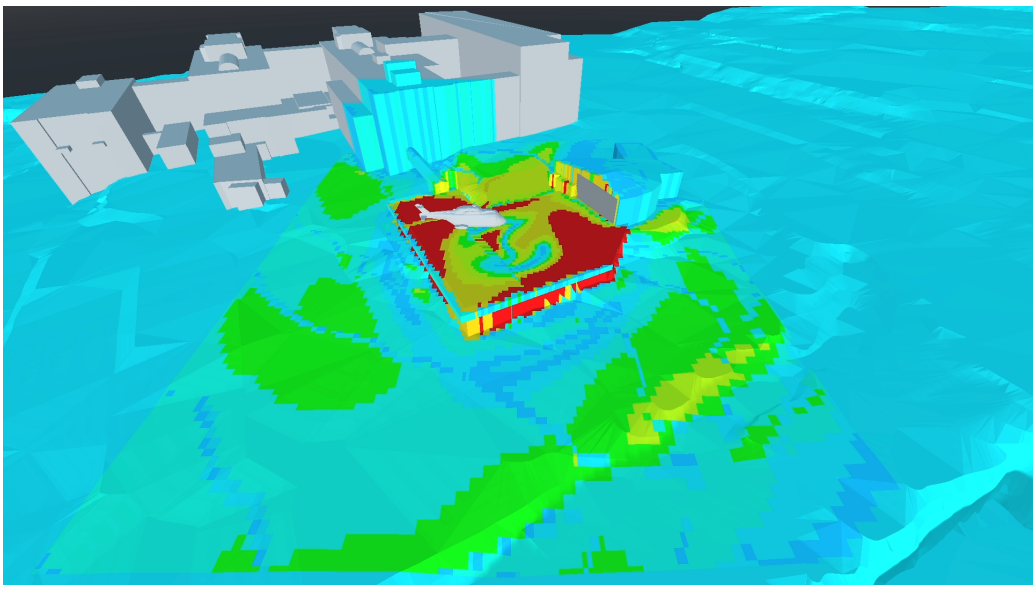

Figure 6. Midpoint in Godot animation with stationary camera to the side.

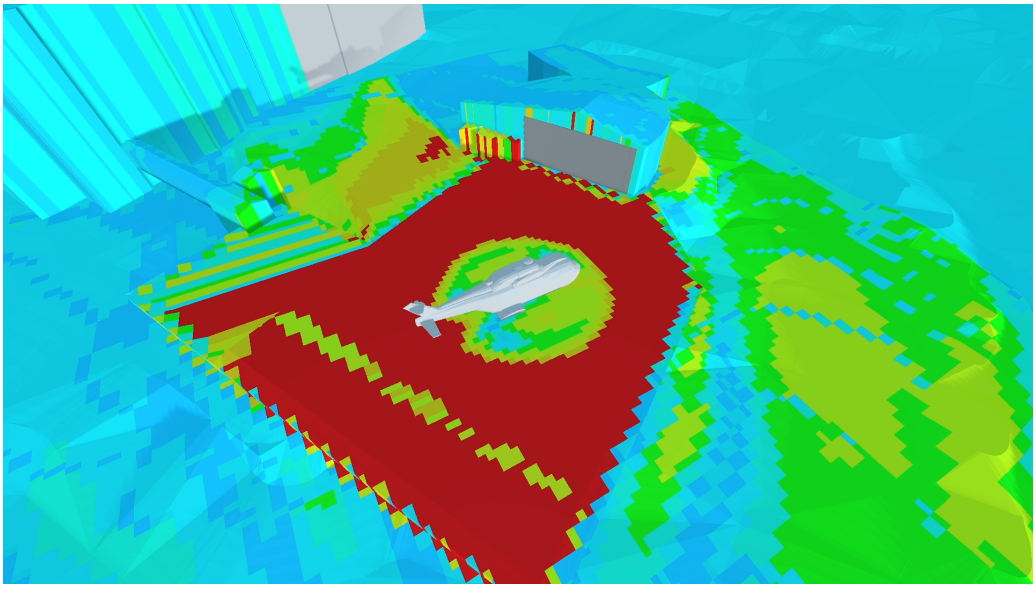

Figure 7. End of Godot animation with the camera attached to the helicopter.

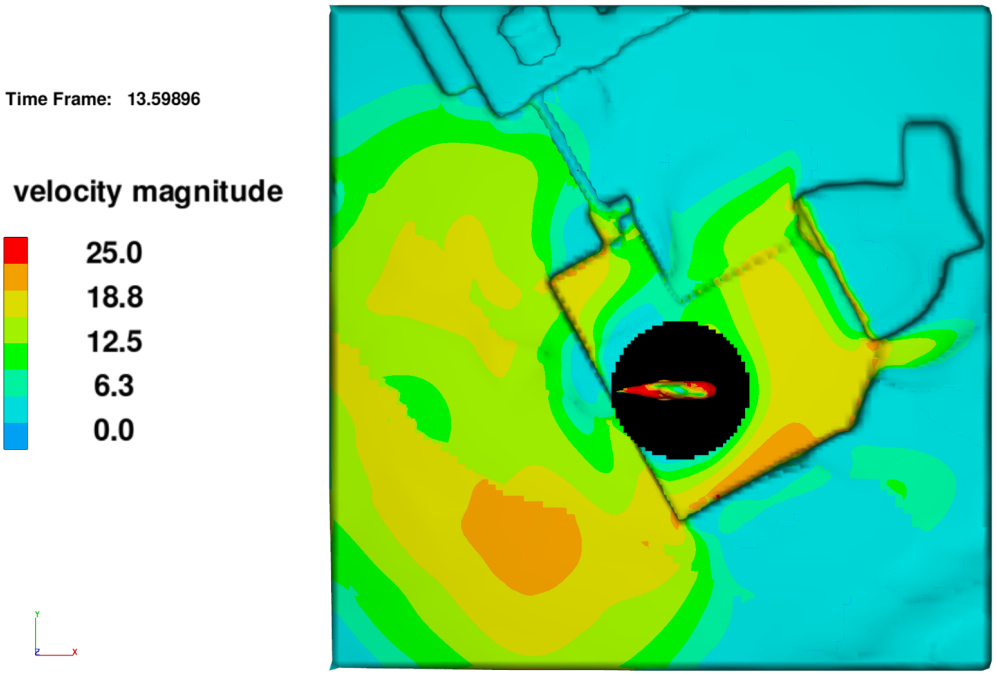

Figure 8. Start of FLOW-3D animation.

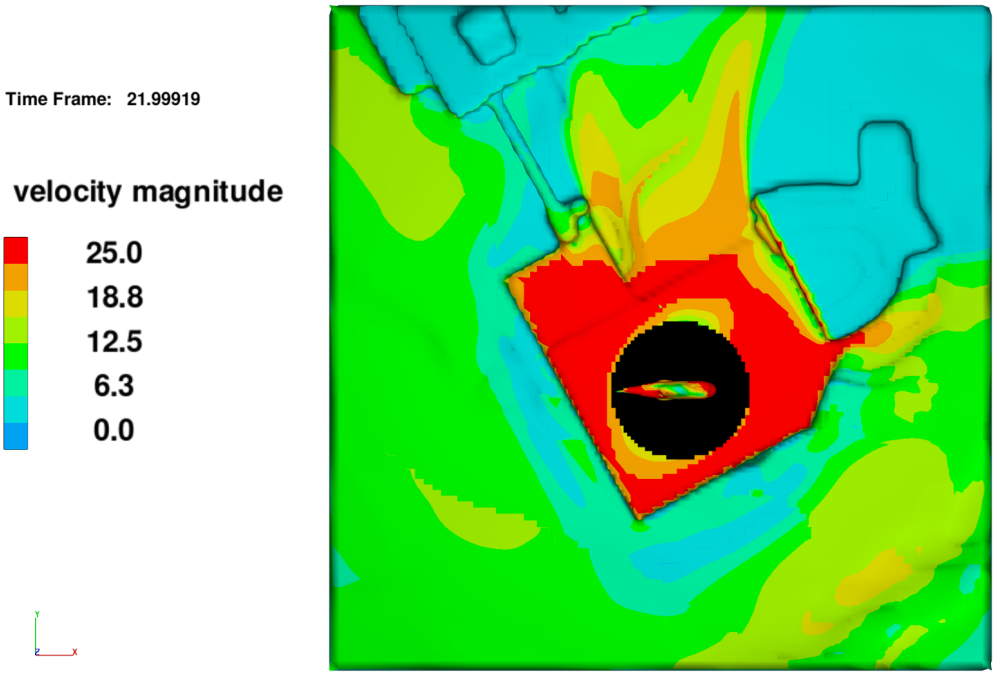

Figure 9. Midpoint of FLOW-3D animation.

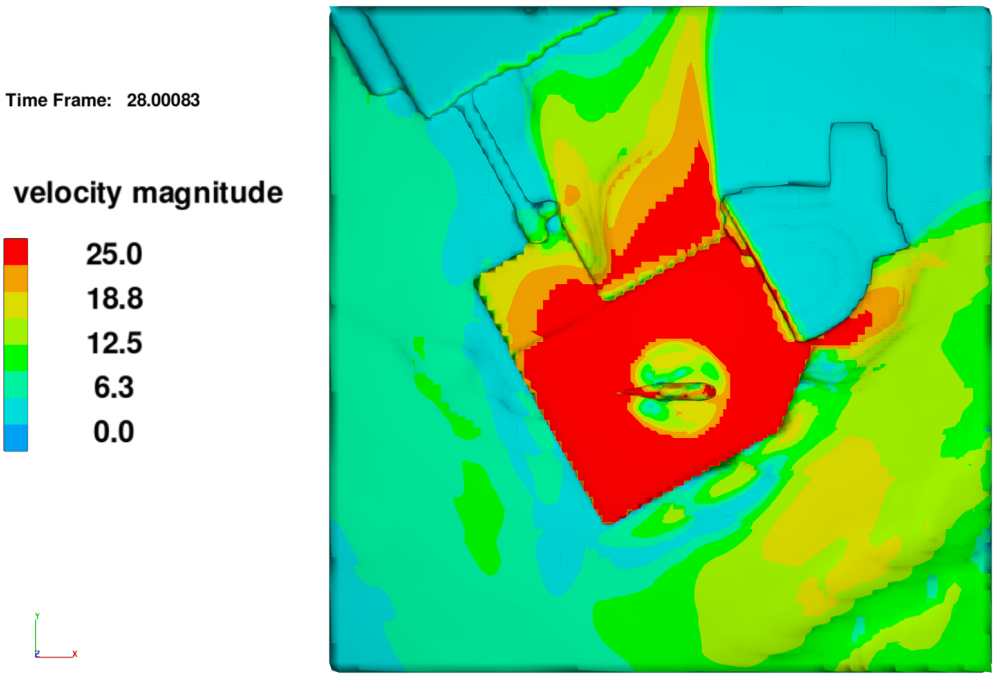

Figure 10. End of FLOW-3D animation.

Figures 5, 6, 7 and the animations in Appendix A.4 clearly show how the effect from the helicopter's downdraft on different areas changes over time. We see that the pedestrian areas west and south of the helipad are affected the most during the initial and final stages of the animations. Note that the downdraft's effect on the hangar's gate has not been accounted for in the animation. Furthermore, the entrance to the patient tunnel is affected the most around halfway into the animation. Figures 8, 9 and 10 confirm the observations that have been made by providing an alternate form of visualization.

The helicopter's movement in the Godot animation is linear. This is due to the movement being linearly interpolated between two time steps containing XZ-coordinates. It is possible that partitioning the

movement into more steps using data from several time steps, however this was ultimately not pursued due to the amount of time it would take to implement and the minor impact it would have on the results.

In Section 2.3 when creating textures, there are gaps in between cells that must be filled. In some cases, such as the top left corner of Figure 3, we see that some gaps can be quite large. The size of such gaps indicates some form of inconsistency in the data that was used, which could be explained by how it was processed with respect to volume fraction. It is possible that edge cases that have not been accounted for exist, e.g. where the volume fraction is 1.0 or 0.0 and the cell is a part of the surface.

The resolution of the textures is quite low. Increasing the resolution would not necessarily have been difficult or very time-consuming, however it would not have made much of an impact on the results, and is more of a cosmetic issue in this case.

The animations created in Godot do not necessarily affect any conclusions that can be drawn from the results overall. It would be more accurate to claim that they function as reinforcements. The animation generated in FLOW-3D was completed within minutes, whereas the animations created in Godot were quite complex and time-consuming to complete. This was despite previous experience with both Python and Godot.

Many students from previous years have used FLOW-3D's FlowSight (FLOW-3D, 2022b) application to perform post-processing. FlowSight is able to produce visualizations with features such as streamlines, which could have been interesting to include in our results. Unfortunately, FlowSight was not installed on the available computer during the project work. Thus, we were unable to produce visualizations with streamlines.

4 Conclusions

Regardless of the points discussed in Section 3, it is reasonable to suggest the approach taken in this report was successful. Throughout the report, we have visited several interesting concepts such as data processing and texture generation, which have yielded great results.

As illustrated by the figures in Section 3 and the animations in Appendix A.4, we can see that the downdraft caused by the helicopter to the pedestrian areas and the patient entrance to the hospital tunnel varies. The pedestrian areas are affected the most during the initial and final time steps, while the patient entrance is affected the most around halfway in to the animation.

Possible future work includes to investigate the velocity magnitude on the hangar's gate, which is located toward the side of the helipad. This comes as a natural recommendation, due to it being discussed with professor Per-Arne Sundsbø as a potential part of the task at an early stage of the project. This was ultimately not pursued, due to the project's short time frame and the amount of time that went in to create the report's final result.

5 Acknowledgements

Special thanks to my brother Kenneth Wilhelmsen for feedback, advice and inspiration on possible ways to approach the task. I would also like to thank professor Per-Arne Sundsbø for providing the FLOW-3D simulation that was used in the analysis.

6 References

Blender (2022). Blender 3.4. https://www.blender.org/. Accesed on 16.11.2022.

FLOW-3D (2022a). FLOW-3D. https://www.flow3d.com/. Accesed on 18.12.2022.

FLOW-3D (2022b). FLOW-3D: POST. https://www.flow3d.com/products/flow-3d-post/. Accesed on 18.12.2022.

Godot (2022a). Godot API: Shading language. https://docs.godotengine.org/en/stable/tutorials/shaders/shader_reference/shading_language.html. Accesed on 18.12.2022.

Godot (2022b). Godot API: Timer. https://docs.godotengine.org/en/stable/classes/class_timer.html. Accesed on 18.12.2022.

Godot (2022c). Godot API: Tween. https://docs.godotengine.org/en/stable/classes/class_tween.html. Accesed on 18.12.2022.

Godot (2022d). Godot website. https://godotengine.org/en. Accesed on 18.12.2022.

imageio (2022). imageio documentation. https://imageio.readthedocs.io/en/stable/. Accesed on 18.12.2022.

Jarslett, Y. (2021). AW101. https://snl.no/AW101/. Accesed on 18.12.2022.

NumPy (2022). NumPy website. https://numpy.org/. Accesed on 18.12.2022.

OBS (2022). OBS Studio. https://obsproject.com/. Accesed on 18.12.2022.

Python (2022). Python programming language. https://www.python.org/. Accesed on 18.12.2022.

Sundsbø, P.-A. (2022). Visualization of downdraft from AW101 SAR Queen helicopter. https://uit.instructure.com/courses/27052/files/2085941?module_item_id=699902. Accesed on 11.12.2022.

Toftaker, J. and Kleven, R. (2020). Se den dramatiske landinga til Norges nye redningshelikopter. https://www.nrk.no/trondelag/enormt-lufttrykk-fra-nye-redningshelikopter-skaper-trobbel-1.15203020. Accesed on 19.12.2022.

Appendix A External links

This appendix contains external links to the simulation data, models, source code and the full animations from which selected frames have been included in the report.

A.1 Simulation data

The data from the simulation used as a basis for this report can be found here: `https://drive.google.com/drive/folders/1WqYDRVTcwMmO_oYsYifn1NDk9adu7utZ?usp=sharing`. Below is a brief description of its contents:

- **Directory:** `/raw/` — Raw data exported from FLOW-3D containing vectors for wind velocities at given XYZ-coordinates. Each file contains data for multiple time steps.
- **Directory:** `/parsed/` — Parsed data divided into individual files for each time step. Each file contains XY-coordinates and the length of the associated wind velocity vector.
- **File:** `movement.txt` — The movement of the models in the simulation. Note that in this case, all the models except the helicopter are moving. The buildings and the terrain move towards the helicopter to simulate movement.

A.2 Python scripts and models

The Python scripts used to process data and create textures can be found at: `https://github.com/kribw/fluid-mechanics`. The most notable scripts and their usages are as follows:

A.2.1 Processing data

`animations/parse_files.py` — Process data from FLOW-3D and write to individual files for each time step.

A.2.2 Determining texture dimensions

`animations/parse_input.ipynb` — Jupyter notebook used for several purposes, one of them being determining the dimensions of the texture.

A.2.3 Creating textures

`animations/create_texture.py` — Create textures from processed data.

A.2.4 3D models

`/models/` — 3D models used in simulation and Godot animation. Both *stl* and *glTF* formats are included in separate subdirectories.

A.3 Godot project

The repository for the Godot project used to create the animations can be found at: `https://github.com/kribw/VR-H2022/tree/main/fluid_simulations`.

NOTE: The Godot project will not compile without errors with the code available in the repository, due to a missing model which exceeds GitHub's file size limit of 100 MB. This was discovered while pushing the final changes, right before submitting the report. To compile the project, the missing file `Environment.glb` must be downloaded from Appendix A.1 and copied to the `/models/` directory.

A.4 Animations

There are three different animations:

1. https://youtu.be/IedGXmY8tm0 — **Godot animation**: Camera attached to the helicopter itself.
2. https://youtu.be/Ptdo3120azE — **Godot animation**: Static camera angle from the side, where we can see the helicopter landing.
3. https://youtu.be/DvduZhGfeHU — **FLOW-3D animation.**