OULU UNIVERSITY OF APPLIED SCIENCES

Aku Jaakola

UNITY 3D VISUALIZATION TOOL FOR 5G CELL MEASUREMENTS

UNITY 3D VISUALIZATION TOOL FOR 5G CELL MEASUREMENTS

Aku Jaakola
Bachelor's Thesis
Fall 2019
Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information and Communication Technology, Software Development

---

Author: Aku Jaakola
Title of the bachelor's thesis: Unity 3D Visualization Tool for 5G Cell Measurements
Supervisors: Teemu Korpela, Jussi Kallio
Term and year of completion: Fall 2019 Number of pages: 34

The objective of this thesis was to create a commercial version of the previously developed proof-of-concept 3D Visualizer application. The software is used to visualise wireless cell measurement data, mainly for 5G, and it uses the Unity 3D engine. The project was commissioned by Keysight Technologies Finland.

The application had the requirement that it needed to have a generic interface for inputting data that could be integrated into any Keysight products needed. This interface was made using Windows Communication Foundation.

The result was a working product that will be released commercially soon. The interface is not as generic as it was initially planned, always expecting a certain type of data, but it works well for 5G measurements. The application has a lot of potential for a further development and the development work will continue at Keysight.

Keywords: 5G, Unity, Windows Communication Foundation

PREFACE

This thesis was mainly made during the first half of 2019.

I would like to thank Jussi Kallio and Teemu Korpela for being my supervisors and guides for this project on behalf of Keysight and OUAS respectively.

I would also like to thank Sampsa Isohätälä, who was a great help in the polishing stages and created the UI, Keijo Sipola for taking part in creating the WCF interface and taking care of the client side of it, and Jarmo Halme for creating the 3D map models used in the application.

Oulu, 1.8.2019
Aku Jaakola

CONTENTS

1 INTRODUCTION

This thesis was commissioned by Keysight Technologies Finland, a part of the international corporation called Keysight Technologies. Keysight Finland's office in Oulu focuses on developing hardware and software for wireless cell measuring.

Today's cellular antennas broadcast information in every direction at once. This is inefficient as signals cross each other constantly, causing interference and even transmission errors. To improve these shortcomings the 5G technology utilizes beamforming. This means that signals are not sent out all over anymore, but the data to be transmitted can be focused in a single beam that is then steered directly to a specific user (figure 1). This technique minimizes interference from other signals and allows base stations to handle more incoming and outgoing data at once.

FIGURE 1. The difference between 4G and 5G transmissions (1)

5G beams can be focused both vertically and horizontally towards a user. Because of their high frequency, they are prone to block easily by hazards in their way. These properties create the need for cell measurement tools to be able to

visualize these beams in a 3D environment to understand how beam attributes are affected by the user's location and the environment (figure 2).

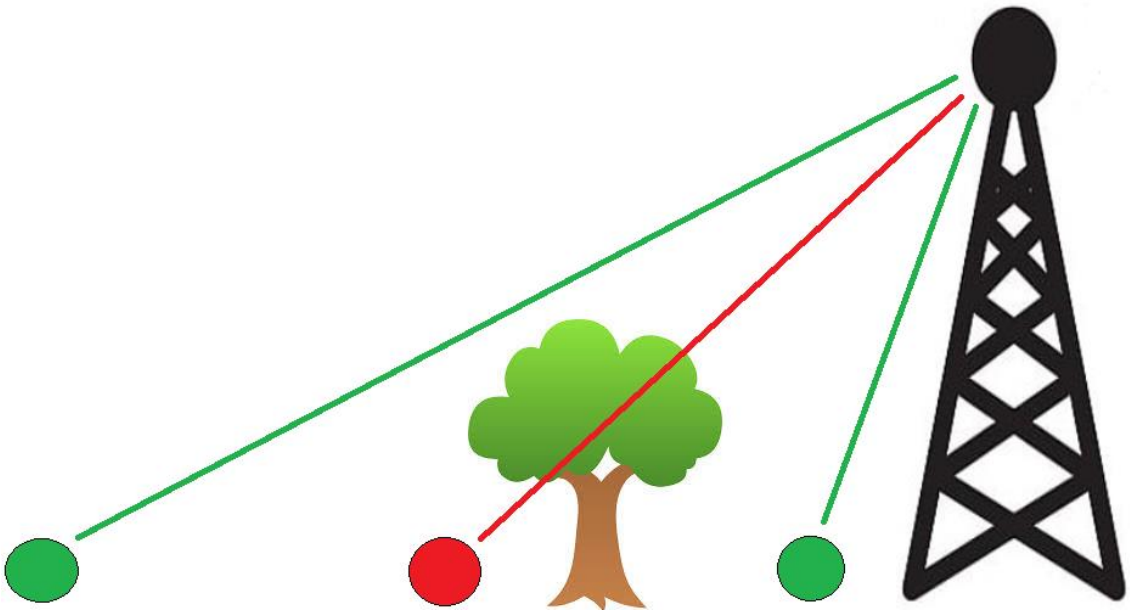

FIGURE 2. The concept of how 5G beams should be visualized, green meaning a good signal and red a bad one

The start of this project dates to Spring of 2018, when the author first tried to do indoor beam visualization using Google Earth at Keysight Technologies Finland as a project part of his studies. Google Earth was chosen because it already had a huge database of 3D environments all over the world. This proved to be a cumbersome solution, as Google Earth did not have anymore an API that could be embedded to an existing application, as the framework it used lost support from major browsers due to security reasons (2). This had the result that the user would have had to upload the files produced by a separate cell measurement post-processing tool to a separate Google Earth application. The Google Earth environment was also relatively restricted in terms of visualization options, and the user could not see inside of its 3D buildings. This meant that the user would have had to use a building defining tool developed by the author to specify the coordinates, floor plans and heights themselves to see the insides of the building the measurement was done in. The downside for this was that Google Earth blocked the user defined buildings with its own 3D building models if the user

wanted to see the 3D environment surrounding it. Because of these drawbacks, this solution was deemed unsuitable for the planned use.

After this, the experimentation with Unity started. Unity was chosen as the platform because it could be used to make a virtual reality application from the same code used in the desktop application. The result of the first phase was a proof-of-concept application where a user could view a building made using the previously mentioned building defining application in an empty 3D environment and measurement data visualized inside it. After this, the scope of the project was lifted to use real-life 3D environment models. The results of this test were so good that Keysight wanted to implement an application like this to their own tools.

The aim of this thesis was to create a commercial version of the Unity visualization tool that would have a generic interface for a data input so that it could be easily made part of any of Keysight's applications needed. In this report it will be first explained some of the key technologies part of 5G, and then it will be explained how the application was made and how it could be developed further.

2 5G

The mobile data usage is on the rise all over the world. According to the networking hardware company Cisco's estimates, by the year 2020 5.5 billion mobile users worldwide will use on average 20 gigabytes of data per month, not to mention the increasing number of devices implementing the Internet of Things (3). Modern mobile networks are struggling to keep up, and 4G base stations will easily clogg in areas with a high population density. New technologies, such as virtual reality streaming and self-driving vehicles, also demand fast and stable wireless internet connections to be viable.

5G is aiming to fix these issues. 5G is the fifth generation of mobile networking standards defined by the 3rd Generation Partnership Project (3GPP). The first 5G specification, officially called 5G NR (New Radio), was revealed in December 2017. The first phones supporting 5G will be released in 2019, but a wider launch of the technology will most likely occur in 2020. (4.)

5G has been promised to reach faster and more consistent connections with a much higher capacity and a lower latency than 4G (figure 3). According to Qualcomm, a telecommunications equipment company, it can reach typical speeds of 1.4 Gb/s (4). This is achieved by implementing different technologies that play off each other's strengths and weaknesses. The main technologies making 5G's goals possible are millimetre waves, small cells, massive MIMO, full duplex and beamforming.

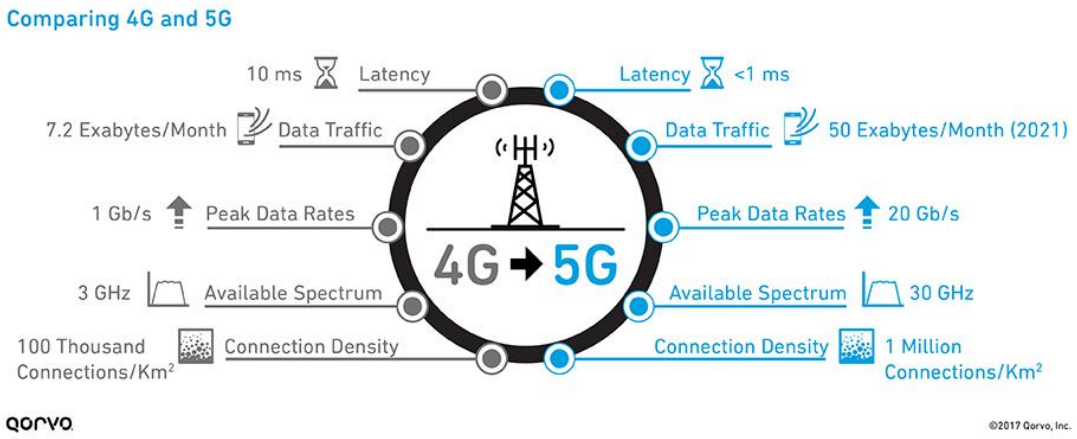

FIGURE 3. Comparisons between different attributes of 4G and 5G (5)

2.1 Millimetre waves

While all previous mobile network technologies use frequencies below 6Ghz, 5G uses millimetre waves that have frequencies between 30 and 300Ghz. Millimetre waves are named after the fact that their wavelength is between 1 to 10 millimetres, compared to the wavelengths of tens of centimetres of older networks. This high spectrum ensures much higher data transfer speeds than 4G. It is also much less congested since millimetre waves have not been used much in the past because of the expensive equipment to take full advantage of them. But with the technology advancing they have become much more affordable. (6.)

The huge drawback of millimetre waves originates from the same source as their advantages do: the frequency. The extremely high frequencies used mean that the waves have only a small range where they are efficient, and they have a very hard time going through obstacles, such as walls, trees and even people. There are, however, other methods involved in 5G that are used to try to downplay this problem, such as small cells and beamforming.

2.2 Small cells

Because of the limited range of 5G beams, the base stations need to be much closer to each other and the users. These base stations are called small cells. Like the name suggests, they are much smaller than previous generation base

stations, and portable. This makes it easy to plant them e.g. on top of buildings and lamp posts, in order to build a dense network with them (figure 4).

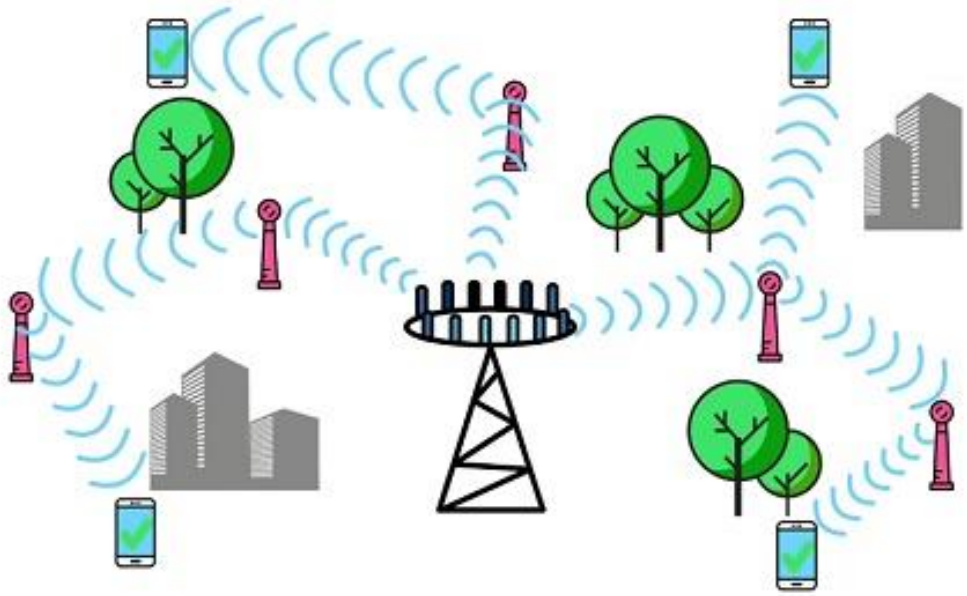

FIGURE 4. Small cells forming a network (7)

Their small range has both an advantage and a disadvantage: the same frequency bands can be used in more cells in different areas without them interfering with each other's signals, but it also makes it more challenging to build an efficient network in sparsely populated areas. Small cell base stations also have much smaller antennas thanks to the size of millimetre waves. The small size of the antennas means that more of them can be fit in a single base station, which means that 5G can take advantage of massive MIMO. (6.)

2.3 Massive MIMO

MIMO stands for Multiple-Input Multiple-Output. It is a technology that allows devices to send and receive multiple signals simultaneously over the same radio channel. This is achieved by using multiple antennas, typically two or four. MIMO increases the capacity of the network without having the need for bigger frequency spectrums, and it is already in use in many 4G base stations. (3.)

Massive MIMO, which is used by 5G, takes MIMO and increases the number of antennas to tens or even hundreds (3). This increases the capacity of the system

even further. The more antennas there are, the better the performance is in terms of speed and stability. These antennas can also be separately directed at users, implementing beamforming.

2.4 Beamforming

5G base stations do not transmit signals all around them like 4G does. Instead they target data beams directly to users. This allows for much faster speeds when a lot of data can be focused on a single beam. It also greatly reduces interference from other signals sent out from the base station.

Because of the bad ability of millimetre waves to go through obstacles, sometimes the best route from the base station to the user is not the most direct one. Beamforming is used to calculate the path that has the best combination of time taken and signal strength for beams (figure 5). For example, if there is a tree between a user and a base station, the base station can bounce a beam off a building to the user. It can also be used in making sure beams do not cross each other, decreasing interference even further. (6.)

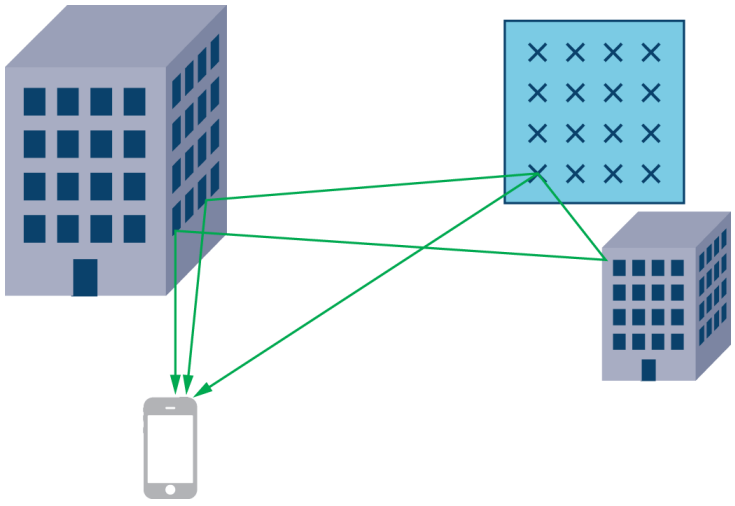

FIGURE 5. Different paths beams can take from an antenna to a user (8)

2.5 Full duplex

Full duplex allows two devices to both transmit and receive signals between them simultaneously on the same frequency. Previously communication had to be done by the transmitters using different frequencies compared to the other, or

one device sending data and the other could do nothing but listen, and vice versa when a response was given. Full duplex supported by 5G can effectively double data transfer speeds in certain use cases as devices do not have to take turns anymore, and it allows the frequency spectrum to be used more effectively by reusing bands. (6.)

3 VISUALIZING CELL MEASUREMENT DATA IN UNITY

3.1 Input data and its interface

The input data for the visualization application comes from a separate application. In the first version, the data inputting software is Keysight's Nemo Analyze, which is used to handle measurement data files. Nemo Analyze can show the results of a measurement in many different forms, such as on a graph or 2D map (figure 6). The 3D visualizer application is basically just another way to visualize a measurement, as the software window is directly opened from Nemo Analyze. The license to use 3D Visualizer is sold as a part of the Nemo Analyze package.

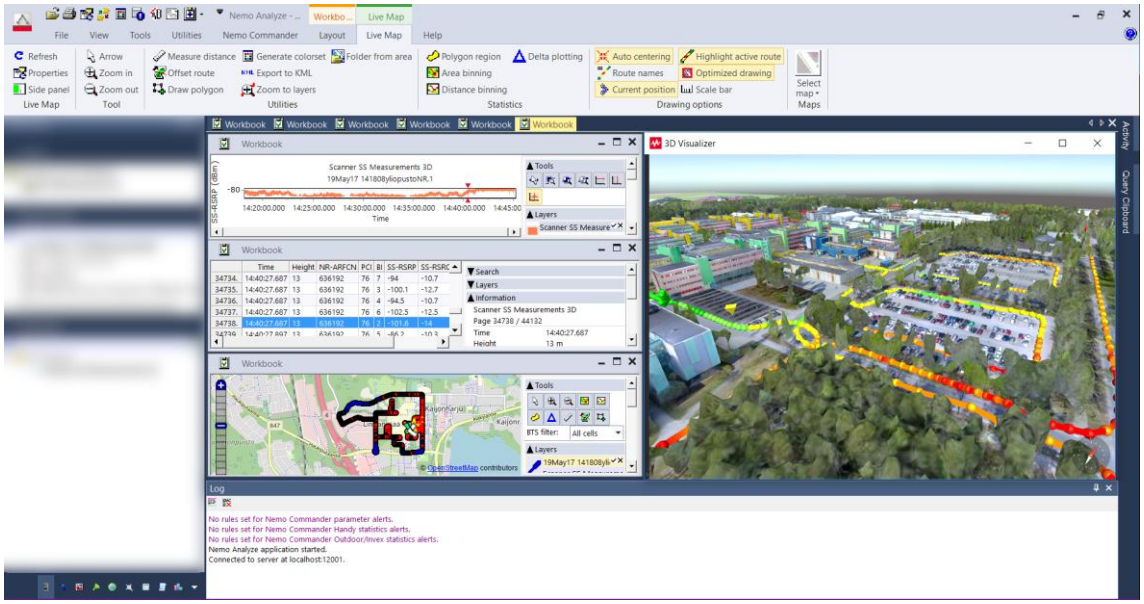

FIGURE 6. A 3D Visualizer window opened next to some other views of the same data in Nemo Analyze

The data consists of marker data and base station data. Each marker has GPS coordinates, altitude and measurement values for many different parameters. Base stations also have GPS coordinates and height, and the different cells and PCIs (Physical Cell Identities) contained in them. This data is transferred in the .NET DataTable form.

The input data is transferred to Unity through WCF. WCF stands for Windows Communication Foundation, and it is a framework used for building applications

that can act as a service (9). 3D Visualizer acts as the service host. The interface provided by WCF allows the Unity application to receive input data, as well as send and receive time stamps to and from the client application, in this case Nemo Analyze, to synchronise between the different views (figure 7). The original intention was for the interface to be as generic as possible, so that it can visualize any kinds of parameters and even 4G measurements and can be implemented to any of Keysight's other tools in the future. The current version, however, expects certain parameter titles to be in the input data table and the data is read based on the column names. This is due to time constraints.

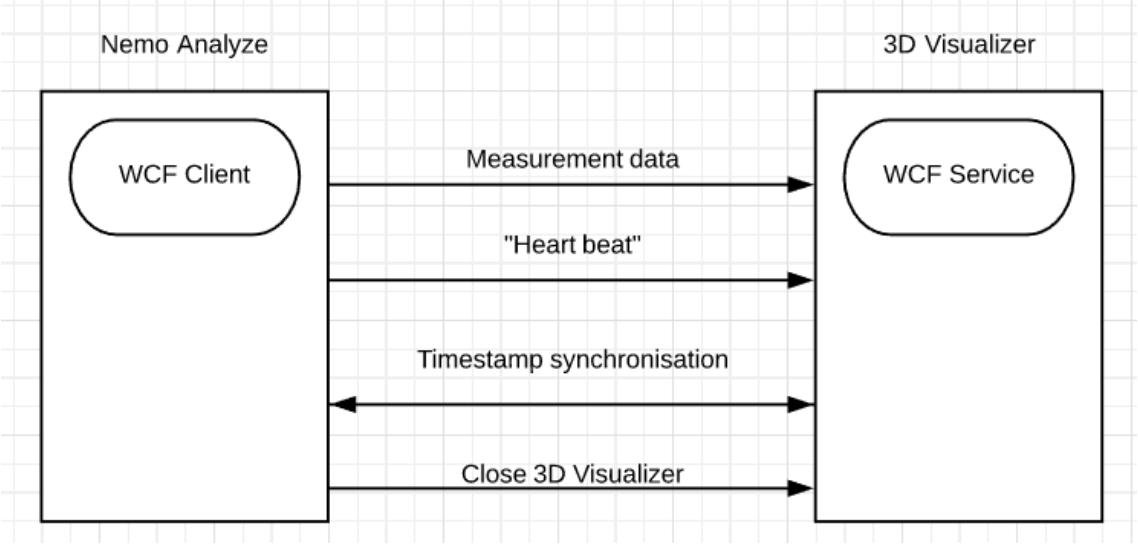

FIGURE 7. A diagram showing the connections between the WCF service and client

Unity uses Mono to run, which natively supports C# (10). Mono is an open-source development platform based on the .NET framework (11). Mono does not have full support for WCF, however, and the support that exists has very limited documentation, so there were a few problems when implementing the service into the project.

In WCF services, bindings are objects that are used to define the communication details so that clients can connect to the endpoint of the service (12). Only certain bindings have support for duplex contracts, meaning that a service can receive as well as send data over the interface. This is a required feature for the 3D Visualizer service. NetNamedPipeBinding, a binding that uses named pipes, would

have been the best choice in this project, as the service and the client will always be running on the same machine, therefore named pipes would have been the safest and most reliable option. However, Mono does not have support for NetNamedPipeBinding (13). Instead, NetTcpBinding had to be used, which is first and foremost meant for a connection between separate computers, so it is not completely optimized for this use case, but it still works. This setup also does not work unless the SecurityMode property of the binding is set to None. This is not much of a problem in this application as the data transferred is plainly seen in Nemo Analyze anyway.

Another issue was that the service has an idle timer of ten minutes, meaning that it closes all channels if there are no messages received from the client during this time. This timer should be an editable property, but for some reason no matter what properties the author tried changing in the service, no effect was seen. This was likely just another symptom of Mono's limited support for WCF. A simple fix that was made for this was to make an empty "heart beat" function in the interface where the client calls the service host in certain time intervals to keep the channels alive.

Normally, a client can connect to a WCF service using a client proxy class created by the Service Model Metadata Utility Tool or Visual Studio's Add Service Reference feature (14). These tools can generate all the methods and metadata of a service automatically. However, they could not connect to the service when it was running in the Unity application. If the same service code was run on a simple .NET console application, the client could be created with no problems. So, when the initial client class was created, the one created from the console application service was used, as that worked with the Unity service too. After that when any changes were made to the interface, it was faster to just manually edit the client class.

3.2 Measurement markers

Measurement markers are visualized as spheres in the 3D environment. The input data is turned into objects that have a time stamp and a list of data values with their parameters as properties. Each marker object has also a Game Object

property, which are the 3D objects in the Unity environment. The user can change the transparency and size of the spheres. Hovering the mouse over a marker sphere shows a label with data about the measurement point.

The marker spheres are coloured based on the value they have for the parameters the user has chosen. Most parameters colour the spheres somewhere on the black-red-yellow-green-axis, black being the worst values and green the best. The spheres can also be coloured based on which beam index has the best value with the given filters, with each beam index having its own colour. Users can filter the data visualized with toggles. There are three filter categories, which are PCI, beam index and channel. These toggles are created dynamically based on which values appear in the measurement data.

There is also an option to draw lines between subsequent markers (figure 8). This helps in perceiving routes where there are large gaps between marker points. The lines are gradient coloured based on the markers they are connected to.

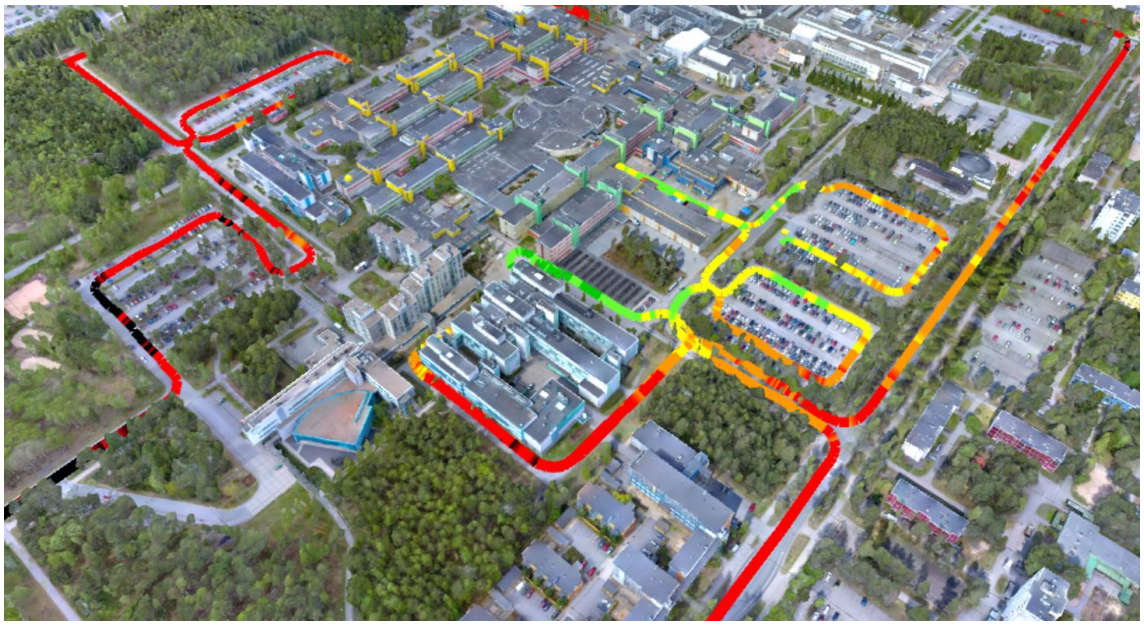

FIGURE 8. A measurement route visualized with lines

Clicking markers in the Unity application sends to the client a synchronisation time stamp, where the data cursor moves to highlight the corresponding data point in whatever view that is open, potentially showing more info about the measurement point beyond the data shown in 3D Visualizer. Likewise, a data point can

be clicked in any view on the client side, and a cursor jumps to the marker with that time stamp and highlights it on the Unity side. The cursor in 3D Visualizer is an upside-down yellow rotating pyramid floating above a marker (figure 9). This can also be used to playback a measured route with Nemo Analyze's playback feature, with the 3D Visualizer cursor moving through the route at the same pace as the Nemo Analyze playback, going through every measurement point from start to finish. Whenever the cursor moves, the camera jumps to its place and focuses on the marker the cursor is highlighting.

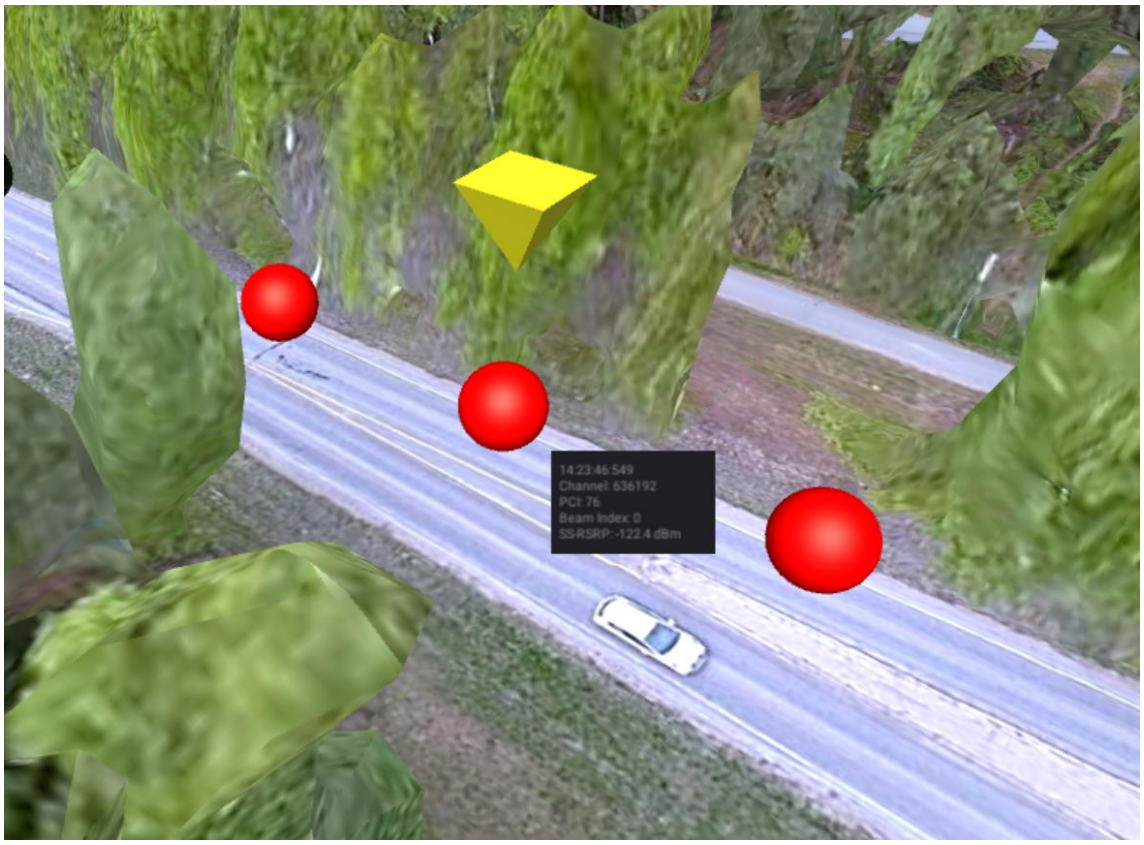

FIGURE 9. A cursor indicating which marker is selected at the moment, also shown is a text label showing the data the marker is visualizing

Keysight's cell measuring tools take GPS data once every second. Because the actual cell measurements are taken almost ten times per second, a problem is caused by measurement markers being created on top of each other, as they all use the latest GPS event data in placing them to the 3D world, and each measurement event with a unique time stamp needs to be made its own marker. Because of this, only the marker first created is rendered. This problem was not

fixed in the first version of the application, but it will be solved in a future version, most likely in the very next release version.

A few ideas have been raised to fix the issue. The first one would be to create a “set” of all markers that share GPS coordinates, and only the marker with the best measurement value for the chosen parameters is rendered, the others are set inactive. Then if a user wants to see the values of the other measurement markers, they can click the active marker and the inactive ones are activated and stacked vertically on top of the active marker. Then, the user can hover their mouse over them to see their values with the tool tip.

Another idea was to once again make sets of markers with shared GPS coordinates. Then just interpolate the markers of a set on the route to the next marker with unique coordinates. For example, there is a ten-metre distance between two sets of markers. The first set has ten markers in it. The markers are then divided one metre apart from each other on the direct line between the starting points of the two sets, and then they work just like any other markers in the visualization.

3.3 Coordinate conversion

The input measurement data to be visualized consists of data points with measurement and location data. The location is presented as GPS coordinates and altitude. Because Unity uses a three-dimensional Cartesian coordinate system, the latitude and longitude coordinates need to be converted to an XZ-plane. In this project the Mercator projection is taken advantage of to achieve this. Using the Mercator projection allows GPS movement in the north-south axis to be directly translated as a movement on the Z axis in Unity, and changing longitude only changes the X component.

The Mercator projection causes the world map to look stretched and in a larger scale than it actually is the further one goes from the Equator (figure 10). To keep the same ratio of one metre in real life, equalling one unit in the Unity coordinate system, no matter what place on Earth is being visualized, a scale multiplier needs to be used when adapting GPS coordinates to Cartesian coordinates. The scale multiplier used is directly the cosine of the latitude in radians. For example

the scale multiplier on the Equator is cos(0) = 1, and in Oulu at latitude 65° it would be cos(1.13...) = 0.422.... This scale is only calculated once for the entire measurement set and the same scale is used in every coordinate conversion in the same visualization session. The scale is determined either based on the first GPS point in the data set or the centre of a 3D environment model, depending on which one is loaded first.

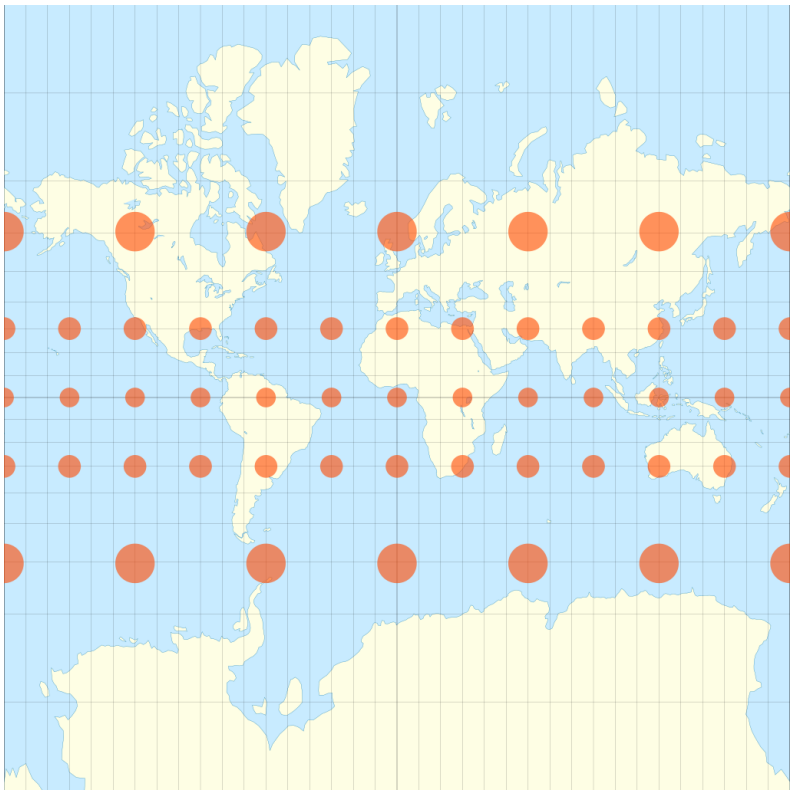

*FIGURE 10. The Mercator projection with Tissot's indicatrix of deformation (*15*.)*

All the coordinates of the objects in a Unity environment need to be floats. Due to floating-precision limitations, the range of the coordinates needs to be made as small as possible to get the most accurate visualization. So instead of setting the origin of the Unity coordinate system to latitude 0° and longitude 0° where the actual (0, 0) point would be with the conversion formula used, the point the scale calculation is based on will be set as the origin of the Unity environment.

Figure 11 shows the coordinate conversion methods. The variable R equals the mean radius of Earth in metres, 6371000.0, the scale equals the scale modifier and the origin is the 2-dimensional vector indicating the converted origin point.

```
5 references
public static float GetXCoordinate(double longitude)
{
    return (float)(R * scale * ToRadians(longitude) - origin.x);
}

5 references
public static float GetZCoordinate(double latitude)
{
    return (float)(R * scale * Math.Log((Math.Sin(ToRadians(latitude)) + 1) / Math.Cos(ToRadians(latitude))) - origin.y);
}
```

*FIGURE 11. GPS to Cartesian coordinate conversion methods (*16*)*

The 3D environment models supported by the application at the moment have their coordinates presented in the Universal Transverse Mercator coordinate system (UTM). In these cases, the coordinate conversion is done first by turning the UTM coordinates to latitude and longitude and from there the methods discussed above can be used (17).

The input data also contains the altitudes of the measurement points. However, this data presents the altitude as a height from the sea level. When creating the markers, the application checks the elevation of the ground in the position of the GPS coordinates of the first measurement point. This elevation is then subtracted from the height of every marker to bring them closer to the actual height they are related to the ground. In cases where this elevation value is incorrect in some way, users can manually adjust the altitudes of markers, OSM map tiles and 3D map models separately inside the application.

3.4 Base stations and beams

Base stations are placed into the Unity environment in much the same way as measurement markers. Their GPS coordinates are converted to Cartesian coordinates and a cube is placed in that position. The height of the base station is presented by adjusting the scale of the y-axis accordingly. The presentation is very simple (figure 12), but in the future a proper 3D model of a base station will most likely be used in its place.

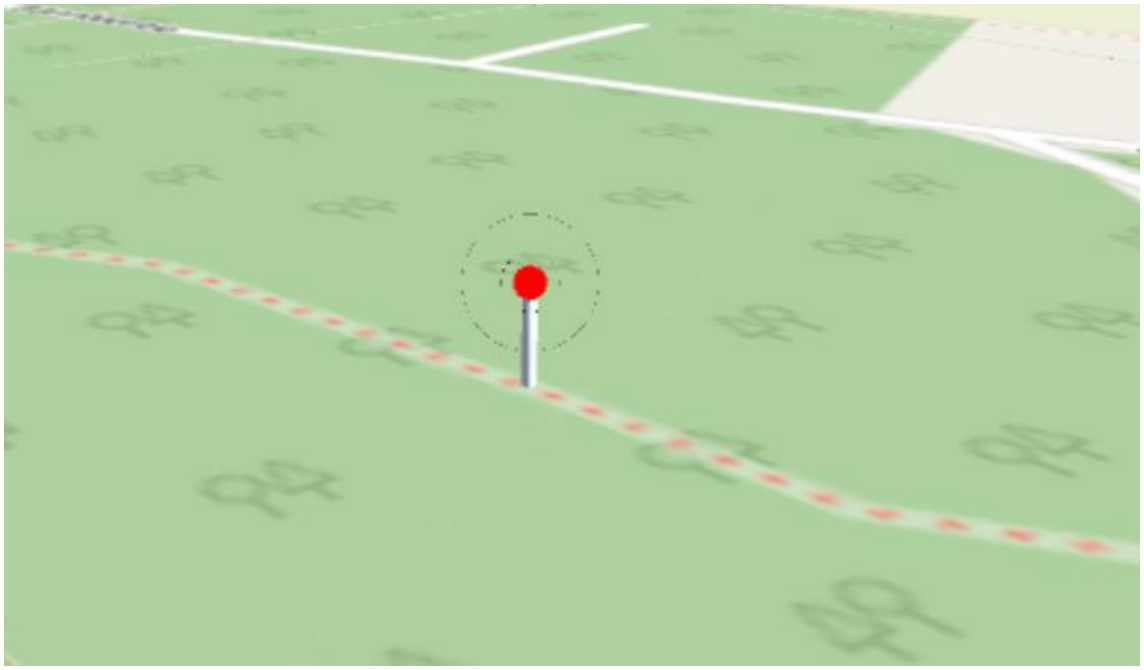

FIGURE 12. A base station on a 2D map

On top of the base station there is a two-dimensional sprite to further indicate the positions of base stations. This sprite is always facing the camera and scales in size according to how far away the camera is, so it always appears the same size on-screen. The Unity camera has a certain distance from where it stops rendering objects in the environment. When a base station falls outside this range, the sprite moves along the vector between its original position and the camera to keep it in the rendering view while still showing the direction where the base station is.

Hovering a mouse over the sprite shows a label with the name of the base station and the distance from the camera in metres. An options panel can also be opened for each station separately to adjust their positions and heights. Clicking the sprite moves the camera closer to the base station.

Beams are drawn from base stations to markers using Unity's LineRenderer components (figure 13). LineRenderer can be used to draw "billboard" lines, meaning that they exist in the 3D space but their side is always facing the camera (18). Each base station has data for the PCIs it contains. If the data set of a marker has values for any of the PCIs that a base station has, a beam is drawn between them. These beams are coloured based on the same parameters and filters as the markers.

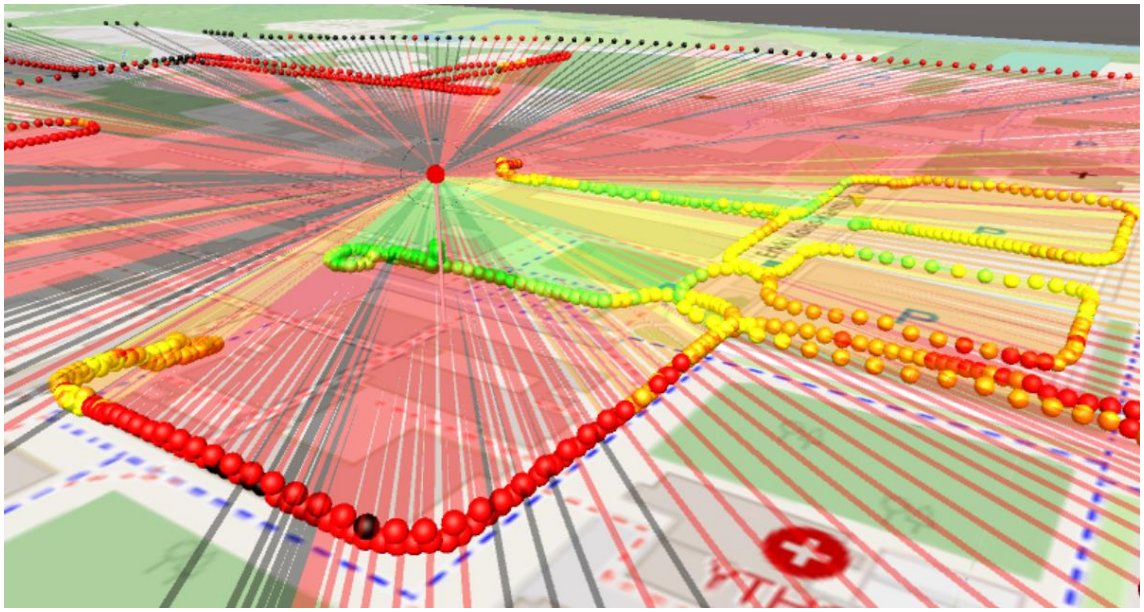

FIGURE 13. Beams drawn from a base station to measurement data markers, coloured by their SS-RSRP signal strength value

Originally, a standard transparent shader was used for the beams, but it had a problem where their transparencies would “stack”, obscuring the view completely in areas where a lot of beams appeared, especially around base stations. This was fixed by using a shader that makes it so that the alpha values of the beams do not add up when they are rendered. This shader, however, introduces a different problem where the colours of the beams do not mix, instead it only shows the colour of the beam furthest back (figure 13). Solving this issue requires more knowledge about Unity shaders, but hopefully it will be sorted out in the future.

3.5 Environment

To help understand the reasons beams behave in certain ways, they need to be visualized in the environment they appear in. The application uses three-dimensional models as the environment (figure 14). At first the application supports models created with a service called Altizure. Altizure allows users to create 3D models out of a series of photos. It was chosen as the first supported model format because it created much higher quality models compared to some other services that were tested.

FIGURE 14. A 3D model of the Raatti Stadium in Oulu created with Altizure inside the application

The photos of the models used so far were shot using a drone. The Altizure application allows users to define the area they want to have a 3D model of, and then it creates a flight path that needs to be taken to get the necessary photos. After this the drone can fly through this route automatically, snapping pictures along the way. The detail level of the model depends on both the quality and quantity of the photos used, and the altitude of the drone's flight path.

Altizure's models are in the OBJ file format. OBJ is a commonly used format for 3D models and almost any 3D modelling software can import models in this format. Thus, it should and probably will be possible at some point to import 3D models to the application from other sources as well. Altizure creates multiple versions of the same model with different levels of detail, affecting the number of polygons of the model and the quality of textures. Each model version is divided into "blocks", which are then combined as the full model in Unity. When going up a level, each block is divided into four smaller blocks. All blocks have roughly the same amount of detail, regardless of the detail level they belong to. There are just more blocks in the higher quality levels. For example, a level 5 model takes roughly 1024 times more memory and rendering power to view than the level 0 version of the same model (figure 15), as it has 4^5 = 1024 OBJ files in it compared to a level 0's 4^0 = 1 file. As the higher quality models are quite heavy to

process, use a lot of memory and take a long time to load, users can choose the version of the model they want to use depending on the hardware they use to run the application. Unity has methods for accessing system and hardware information, so they could be used in the future to recommend certain detail levels for users.

FIGURE 15. Comparison between models of detail level 0 (left) and level 5 (right)

Altizure models also include a JSON configuration file. This file contains the real-world location coordinates of the model in the Universal Transverse Mercator form. These coordinates are needed to place the model correctly to the Unity environment in relation to the measurement data. Altizure uses the same scale of one coordinate unit being one metre in real life as the Visualizer, so the models do not need any size scaling. They do not need any major rotation either, as they automatically face north. In cases where the rotation is not completely correct, users have the chance to rotate the models by a few degrees. Model positions and heights can also be adjusted by the user.

Another completely different format to visualize the environment in 3D, which was briefly considered, was 3D point clouds made using laser scanning. This would have had the advantage that a lot of point cloud data already exists. For example, the National Land Survey of Finland sells laser scanning data from certain parts of Finland (19). The point clouds were not suitable for this use, however, and the idea was dropped quickly, as the data was very rough-looking (figure 16).

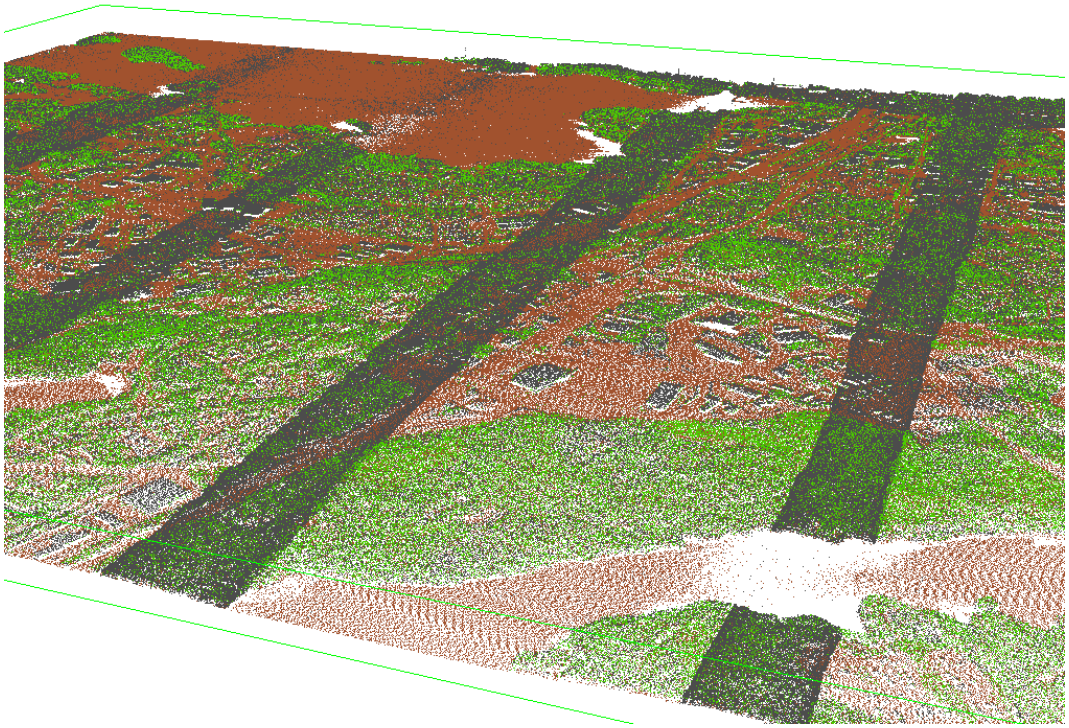

FIGURE 16. Laser scanned point cloud data from the National Land Survey of Finland

In instances where the use of a 3D model is not possible, the application loads flat map tiles from OpenStreetMap. OpenStreetMap is a community driven, open data source for two-dimensional maps all over the world (20). The tiles are downloaded from an OpenStreetMap tile server only to places with measurement data, not the whole map (figure 17).

FIGURE 17. Measurement markers on top of an OpenStreetMap map

Every marker is checked whether it has a map tile under it plus the eight tiles surrounding it. If any of them does not exist, it is downloaded from the server, inserted into the world and saved to the user's hard drive for later use. If a tile already exists on the hard drive, the tile is loaded from there.

The tile image files have a certain way they are named, and that name must be known before the file can be downloaded from the server. The names are formed like this: zoom/tilex/tiley.png, where zoom is the zoom level used and tilex and tiley indicate the horizontal and vertical placement of the tile on the Mercator projection and they have their own formula how they are calculated from latitude and longitude coordinates. Figure 18 shows how this is done (21). The rounded tilex and tiley are then converted back to latitude and longitude by going through the methods in figure 18 in reverse, so the downloaded tile can be placed correctly to the 3D environment, just like the 3D model maps.

```
// Reproject the coordinates to the Mercator projection, transform range of x and y
// to 0-1, shift origin to top left corner and multiply with number of tiles across map

double numberOfTilesAcrossMap = Mathf.Pow(2, zoom);

double xtile = numberOfTilesAcrossMap * ((longitudeDegrees + 180) / 360);
double ytile = numberOfTilesAcrossMap * (1 - (Mathf.Log(Mathf.Tan((float)latitudeRadians) + (1 / Mathf.Cos((float)latitudeRadians))) / Mathf.PI)) / 2;

// Converting them to int rounds them down
int intXtile = (int)xtile;
int intYtile = (int)ytile;
```

FIGURE 18. Converting latitude and longitude coordinates to get tilex and tiley

The metres/pixel resolution of the OSM tiles varies based on the zoom level used and the latitude coordinate of the tile. This resolution needs to be calculated to scale the size of the tiles properly. All tiles loaded are 256 x 256 pixels in size. The resolution is calculated as shown in figure 19. The number 6371000.0 is the mean radius of Earth in metres (21).

```
1 reference
private float GetTileResolutionMetersPerPixel(int zoomlevel, double latitude)
{
    float resolutionAtZoom0 = 6371000.0f * 2.0f * Mathf.PI / 256.0f;

    return resolutionAtZoom0 * Mathf.Cos((float)latitude) / Mathf.Pow(2, zoomlevel);
}
```

FIGURE 19. Function calculating a map tile's resolution of metres per pixel

4 FUTURE DEVELOPMENT

Development of the application will continue at Keysight. One of the main development points will be to make the interface more generic, as it was originally meant to be. Research on what other 3D map model formats besides Altizure's could and should be supported is under way.

Colours representing each parameter value area are now hard-coded into the application. These colours should be unified with other Keysight applications in the future and should be fetched from a colour configuration file which is shared by all of them. The colour legend should also be shown to the user, now it can only be seen in Nemo Analyze.

Dynamic rendering would help the performance of the application in a great way. The 3D maps, markers and beams combined take a lot of rendering power. When the application camera is zoomed out and there are a lot of objects on screen at once, the frame rate can dip to under half of the usual. With dynamic rendering the markers and the map could turn into lower detailed versions of themselves when zoomed out. If done correctly, this could even be done without the user noticing a difference. The LineRenderer component used in the route lines and beams cannot be simplified more than it already is and those seem to be the heaviest to render in the application, so there may also be a need to find a lighter replacement for them.

3D Visualizer could be used in visualizing on-going measurements in realtime. For example, Keysight's Nemo Outdoor is a live measurement data collecting tool that could be used as a client for the application. Different visualization styles besides markers and beams could also be a good option to have. For example, a "wall" around base stations that is then coloured based on measurement results in different directions.

It is possible for the base station configuration files to have data showing antenna directions, heights, tilts, ranges and other information that could be used to show

the “theoretical” beams and area of effect for each cell. These could also be possible to be visualized in the future, so users could see how actual measurement data looks like compared to the theoretical base station data.

For now, beams are drawn directly from base stations to measurement markers. In reality this is not always the case, as beams can and will bounce all over the place to reach their destination, not to mention that unwanted reflections can also be picked up. Keysight has a tool that can detect which direction beams are coming from, and that could be interesting to see in the data visualization.

Support for importing indoor 3D models and visualizing indoor measurements in them could also be an interesting feature.

The project has a lot of potential for a further development. Ultimately any future features and their priorities will be decided based on customer feedback and business demands.

5 CONCLUSION

The purpose of this thesis was to create a commercial version of the 3D Visualizer proof-of-concept application for visualising 5G cell measurements, and to implement a generic data input interface so that it could be integrated into any other Keysight tools necessary, with a release date set for June. These aims were achieved, besides the genericity of the interface. This does not greatly hurt the product, however, as the tool is mainly targeted to be used with 5G measurements anyway, and it will be featured in the next release version of the application.

During the project I learned a lot about Unity programming and rendering, Windows Communication Foundation services, different world map projections and their coordinate styles and of course wireless cell connections and their measuring.

The greatest issues in the project stemmed from Mono's incomplete support and documentation for Windows Communication Foundation services. This was not helped by the fact that WCF was a complete stranger to me beforehand.

The project was quite large considering that it was my first Unity project if you count the proof-of-concept phase as a part of it. However, the code in it was almost completely reconstructed once I started work on the commercial version. So, I got rid of most of the issues that existed in the "newbie" code. During my time working on this thesis, Unity has become a comfortable and easy-to-use tool to me.

Besides some visual effects affected by shaders which I have not had a lot of time to learn about, like the beams, I am happy with the application. If there is one thing I would do differently, it would be trying to divide the Unity scripts into smaller parts. Even after the reconstruction, there are still some classes that do too much, and this results in being difficult to find the methods you are looking for, especially now when there are two people working on the project.

Considering the feedback we have got from demoing the application, there seems to be interest towards a product like this. After some further development, I can see it having potential becoming a very useful tool when determining problems with 5G networks e.g. for network service providers.

REFERENCES

1. Miller, Mark 2018. Differentiate Between 4G LTE and Non-Standalone 5G NR Antennas. Date of retrieval 26.3.2019 https://www.mwrf.com/systems/differentiate-between-4g-lte-and-non-standalone-5g-nr-antennas

2. Hoetmer, Ken. Announcing deprecation of the Google Earth API. Date of retrieval 12.6.2019 https://mapsplatform.googleblog.com/2014/12/announcing-deprecation-of-google-earth.html

3. Mundy, Jon. What Is Massive MIMO Technology? Date of retrieval 28.2.2019 https://5g.co.uk/guides/what-is-massive-mimo-technology/

4. Low, Cherlynn 2018. How 5G makes use of millimeter waves. Date of retrieval 28.2.2019 https://www.engadget.com/2018/07/23/how-5g-makes-use-of-millimeter-waves/

5. Qorvo 2017. Getting to 5G: Comparing 4G and 5G System Requirements. Date of retrieval 13.6.2019 https://www.qorvo.com/design-hub/blog/getting-to-5g-comparing-4g-and-5g-system-requirements

6. Nordrum, Amy 2017. Everything You Need to Know About 5G. Date of retrieval 1.3.2019 https://spectrum.ieee.org/video/telecom/wireless/everything-you-need-to-know-about-5g

7. Rajiv 2017. Five network terminologies to make 5G possible. Date of Retrieval 12.6.2019 https://www.rfpage.com/five-network-terminologies-make-5g-possible/

8. Masterson, Claire 2017. Massive MIMO and Beamforming: The Signal Processing Behind the 5G Buzzwords. Date of retrieval 10.6.2019 https://www.analog.com/en/analog-dialogue/articles/massive-mimo-and-beamforming-the-signal-processing-behind-the-5g-buzzwords.html

9. Microsoft 2017. What Is Windows Communication Foundation. Date of retrieval 3.6.2019 https://docs.microsoft.com/en-us/dotnet/framework/wcf/whats-wcf

10. Unity. Programming in Unity. Date of retrieval 11.6.2019 https://unity3d.com/programming-in-unity

11. Mono Project. About Mono. Date of retrieval 12.6.2019 https://www.mono-project.com/docs/about-mono/

12. Microsoft 2017. Windows Communication Foundation Bindings Overview. Date of retrieval 12.6.2019 https://docs.microsoft.com/en-us/dotnet/framework/wcf/bindings-overview

13. Mono Project. WCF Development. Date of retrieval 12.6.2019 https://www.mono-project.com/docs/web/wcf/

14. Microsoft 2017. Accessing Services Using a WCF Client. Date of retrieval 18.6.2019 https://docs.microsoft.com/en-us/dotnet/framework/wcf/accessing-services-using-a-wcf-client

15. Wikipedia. Mercator projection. Date of retrieval 23.4.2019 https://en.wikipedia.org/wiki/Mercator_projection

16. StackExchange. How to convert lat long to meters using Mercator projection in C#. Date of retrieval 27.2.2019 https://gis.stackexchange.com/questions/15269/how-to-convert-lat-long-to-meters-using-mercator-projection-in-c/285045

17. Movable Type Scripts. Convert between Latitude/Longitude & UTM coordinates / MGRS grid references. Date of retrieval 17.5.2019 www.movable-type.co.uk/scripts/latlong-utm-mgrs.html

18. Unity 2019. Line Renderer. Date of Retrieval 17.6.2019 https://docs.unity3d.com/Manual/class-LineRenderer.html

19. National Land Survey of Finland. Laser scanning data. Date of retrieval 14.6.2019 https://www.maanmittauslaitos.fi/en/maps-and-spatial-data/expert-users/product-descriptions/laser-scanning-data

20. OpenStreetMap. About. Date of retrieval 29.4.2019 https://www.openstreetmap.org/about

21. OpenStreetMap Wiki. Slippy map tilenames. Date of retrieval 15.3.2019 https://wiki.openstreetmap.org/wiki/Slippy_map_tilenames