F3

Faculty of Electrical Engineering
Department of computer graphics and interaction

Master's Thesis

Augmented reality in city

Petr Varga

May 2023

Supervisor: Ing. David Sedláček, Ph.D.

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Varga** Jméno: **Petr** Osobní číslo: **483768**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačové grafiky a interakce**

Studijní program: **Otevřená informatika**

Specializace: **Počítačová grafika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Rozšířená realita ve městě

Název diplomové práce anglicky:

Augmented reality in city

Pokyny pro vypracování:

1) Seznamte se s lokalizačními službami použitelnými ve městském prostředí pro rozšířenou realitu (AR). Dále se seznamte s otevřenými mapovými daty města Prahy [4].
2) Navrhněte a implementujte AR aplikaci, která bude vizualizovat vybrané datové vrstvy viz (1) v ulicích města. Vhodné datové vrstvy konzultujte s vedoucím práce.
3) Seznamte se s technikami vykreslování vhodnými pro vizualizaci vybraných vrstev (anglicky situated visualization [2] - např. zastínění, odhad okolního osvětlení, siluety, ...) a implementujte je v rámci aplikace pro dosažení co nejvěrohodnějšího zakomponování 3D modelů do reálné scény.
4) Výslednou aplikaci otestujte alespoň s pěti uživateli. Zhodnoťte také kvalitu zarovnání dat vůči realitě, kvalitu pokrytí lokalizační službou, datovou náročnost a rychlost zobrazování na různých typech zařízení.

Seznam doporučené literatury:

1] Steve Aukstakalnis. Practical Augmented Reality: A Guide to the Technologies, Applications, and Human Factors for AR and VR (Usability). Addison Wesley 2017.
2] Dieter Schmalstieg, and Tobias Hollerer. Augmented Reality: Principles and Practice (Usability), Addison Wesley 2016
3] Micheal Lanham. Augmented Reality Game Development. Packt Publishing, 2017.
4] opendata Prahy: https://www.geoportalpraha.cz/cs/data/otevrena-data/seznam

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. David Sedláček, Ph.D. katedra počítačové grafiky a interakce FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **17.02.2023** Termín odevzdání diplomové práce: **26.05.2023**

Platnost zadání diplomové práce: **22.09.2024**

Ing. David Sedláček, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. | Datum převzetí zadání

Podpis studenta

Acknowledgement / Declaration

I would like to thank Ing. David Sedláček, Ph.D. for giving me the opportunity to work on a topic that I find very interesting. I would also like to thank him for his advice and support, even though I have not been very communicative.

I would also like to thank my family for supporting me throughout my studies.

I declare that this thesis represents my work and that I have cited all the sources of information used in accordance with the Methodological Guideline on Compliance with Ethical Principles in the Preparation of Final Theses.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Abstrakt / Abstract

V posledních letech zažila oblast rozšířené reality (AR) mimořádný růst, podpořený inovacemi v geoprostorových technologiích. Tato diplomová práce představuje aplikaci, která využívá Geospatial API - technologii uvedenou na trh v květnu 2022 - pro vizualizaci podzemních inženýrských sítí v ulicích města Prahy. Aplikace je založena na datech poskytnutých Institutem plánování a rozvoje Prahy. Úvodní výzkum se soustředí na hlubší pochopení těchto dat a na zkoumání AR technologií vhodných pro jejich efektivní vizualizaci. V následující fázi návrhu jsou podrobně popsány požadované funkce a vzhled aplikace. Tato fáze je následována detailním popisem procesu implementace. Závěrečná část této práce hodnotí přesnost aplikace, její omezení a uživatelsé testování. Tímto poskytuje zhodnocení tohoto API.

Klíčová slova: Rozšířená realita, Geospatial API, Unity, Geografická data

Překlad titulu: Rozšířená realita ve městě

In recent years, augmented reality (AR) has seen remarkable growth, bolstered by advances in geospatial technology. This thesis presents a novel application that leverages the Geospatial API, launched in May 2022, to visualize underground utility lines in the streets of Prague. The data, provided by the Prague Institute of Planning and Development, serve as the basis for the AR application. Initial research focuses on understanding the data set and exploring AR technologies for effective visualization. The subsequent design phase details the intended features and appearance of the application. This is followed by a discussion of the step-by-step implementation process. Finally, the thesis evaluates the application performance, limitations and user testing, providing insight into the capabilities of the API.

Keywords: Augmented Reality, Geospatial API, Unity, Geospatial data

Contents /

1 Introduction **1**

2 Research **3**

2.1 Geospatial data 3
2.1.1 Prague Institute of Planning and Development . 3
2.1.2 Formats 3
2.2 Coordinate Reference Systems . . 6
2.2.1 World Geodetic System 1984 7
2.2.2 Universal Transverse Mercator 8
2.2.3 Křovák's coordinate system . 9
2.3 Global Navigation Satellite System 10
2.4 ARCore 13
2.4.1 Motion tracking 13
2.4.2 Environmental understanding 14
2.4.3 Light Estimation 14
2.4.4 Other capabilities 14
2.5 Geospatial API 15
2.6 Unity 17
2.7 Augmented Reality 18
2.7.1 Making virtual objects seem more realistic 18
2.8 Related work 19
2.8.1 Applications using the Geospatial API 19
2.8.2 Applications using Geographic Information System data 22
2.8.3 Summary 25

3 Design **27**

3.1 Why underground utilities? . . 27
3.2 Application Design 28
3.3 UI design 28

4 Implementation **31**

4.1 Parsing the input data 31
4.2 Acceleration Structure 32
4.3 Server side 34
4.4 Setting up the project in Unity 35
4.5 Communication with the server 35
4.6 Utility line visualizaiton 36
4.7 Geospatial API and accuracy . 38
4.8 Customization 40
4.9 Utility line information 40
4.10 Occlusion and lighting 41
4.11 Other encountered problems . . 42

5 Testing **43**

5.1 Application functions testing . 43
5.1.1 Visualization accuracy . . . 43
5.1.2 Device accuracy 43
5.1.3 Coverage by Geospatial API VPS 45
5.1.4 Internet data usage 46
5.1.5 The speed of visualization . 47
5.2 User testing 48
5.2.1 The first test 49
5.2.2 The second test 50
5.2.3 The third test 50
5.2.4 The fourth test 52
5.3 Summary 52

6 Conclusion **55**

References **57**

A Glossary **63**

B Custom binary file for quick data retrieval from a grid **65**

C Manual **67**

Tables / Figures

2.1 A graticular network 7
2.2 Different types of projections 8
2.3 UTM zones 9
2.4 Křovák's projection 10
2.5 The GPS control segment 11
2.6 Position calculation 12
2.7 Geospatial VPS 16
2.8 Geospatial Anchors 16
2.9 Comparision of GPS, Cloud Anchors and VPS 17
2.10 Pocket Garden 20
2.11 Gorillaz Application 21
2.12 Google Live View 21
2.13 Google Geospatial Creator 22
2.14 AugView 23
2.15 SiteVision Positioning System . 24
2.16 vGIS 25
3.1 UI sketch 29
4.2 Description of the polylineZ record 31
4.1 Description of the main file header 32
4.3 Shapefile rendered data 33
4.4 Sort lines into grid 34
4.5 Request data algorithm 36
4.6 Misalignment in visualization . 38
4.7 Comparision with and without correction 39
4.8 Threshold effectiveness 40
4.9 Menu and color picker 41
5.1 A utility line accuracy test visualization 44
5.2 Comparison of the visualization with QGis visualization ... 45
5.3 A position accuracy visualization 45
5.4 Visualization of the utility lines with reference 47
5.5 VPS coverage 48
5.6 Change of menu after the first testing 50
5.7 Pop-up sign 51
5.8 Change of the signs after the fourth test 53

4.1 Precisions 39
5.1 Visualization accuracy table ... 44
5.2 Device accuracy measurements 46
5.3 Application data usage 48

Chapter 1
Introduction

Augmented Reality (AR) has grown substantially over the past few years, becoming more widely recognized and used by the general public, especially with the release of a mobile game called PokemonGo, which allowed users to interact with digitally created objects embedded into the real-world environment. However, many challenges remain to be overcome to achieve perfect AR experiences.

The accurate placement and tracking of virtual objects in real-world environments remains a significant challenge in the widespread adoption of augmented reality (AR). While various technologies, including global navigation satellite systems (GNSS) like GPS, GLONASS, and Galileo, contribute to making the location data in AR applications more accurate, achieving perfect accuracy is still impossible. GNSS can provide valuable positioning information, particularly in outdoor settings where satellite signals are accessible. However, it is important to note that GNSS has limitations in urban or indoor environments where signal obstruction can occur, impacting its accuracy for precise object placement and tracking. Therefore, AR systems rely on a combination of technologies, including computer vision, simultaneous localization and mapping(SLAM), Inertial Measurement Units (IMUs), and marker-based tracking, to overcome these limitations and enhance the accuracy of object placement.

In May 2022, Google introduced a new application programming interface (API) called Geospatial API that can significantly improve the accuracy of getting the device's position in the real world. It allows developers to utilize the vast amount of data they have collected for their Street View project over the years, and by doing so, it opens the door to new different AR applications that surpass the previous limitations. This opens up opportunities for developers to create AR experiences that are not only much more accurate but also scaleable across diverse environments.

The focus of this thesis is to test the limits of the new Geospatial API while creating an application that uses data provided by the Prague Institute of Planning and Development. The idea is to examine whether the improved accuracy offered by this API is sufficient to develop a practical AR application.

Chapter 2
Research

In this chapter, we will go over the individual core components that the resulting application uses to create the AR experience. In the first section, we briefly discuss the types of datasets available from the Web page of the Prague Institute of Planning and Development. Then we will dive a bit deeper into the geospatial data and it's representation and the coordinate systems that these data use. In the next part, we will discuss global navigation satellite systems, which are the foundation of any geospatial application. The third part will introduce ARCore and its functionalities, before getting to the newly released Geospatial API. Ultimately, we will look into the Unity game engine, before finally describing some already developed applications that visualize geospatial data or use the Geospatial API.

2.1 Geospatial data

As this work is about the utilization of data offered by the Prague Institute of Planning and Development, I will start this section with a very brief overview of this institute, its goals and importance for Prague. After that, I will discuss the different geospatial data formats based on their occurrences on the Geoportal[1].

2.1.1 Prague Institute of Planning and Development

The Prague Institute of Planning and Development (IPR Prague)[2] is a key entity responsible for overseeing the architectural and urban development strategy of Prague. Sponsored by the city, it represents Prague on matters related to spatial planning.

Its primary responsibilities include the drafting and coordination of strategic and spatial planning documents. These cover a variety of areas including public spaces, transport, technical infrastructure, landscape, and economic infrastructure. Key documents produced by IPR Prague include the Prague Building Regulations, the Prague Waterfront Concept, and the Prague Public Space Design Manual.

Two significant ongoing projects for IPR Prague are the development of a new land use plan, known as the 'Metropolitan Plan', and the revision of the Prague Strategic Plan.

In addition to these responsibilities, IPR Prague also plays a crucial role in the handling of geographical data and information. This data processing work supports applied research and the creation of documentation that aids the city's development.

IPR Prague is also the administrator of the Geoportal Prague website[3], which offers a wide range of Prague maps to the public[4].

2.1.2 Formats

There are several different types of data sets that can be accessed by the general public, ranging from information about city vegetation to positions of public toilates in the city. Most of these datasets are provided in one or more geospatial data formats. As the

majority of the available information utilizes vector representation, the most common geospatial data formats are Shapefile, GeoJSON, DXF and CityGML. However, the only format that is provided for all vector-represented datasets is the Shapefile, and thus it will be described in more detail than the others.

- **Shapefile**

  Shapefile is one of the most prevalent nontopological geospatial data formats utilized in various fields like urban planning, transportation, and environmental studies. Non-topological means that the individual geometries in the dataset are independent of each other. Any alterations made to one geometry will not affect the others, which provides flexibility in modifying the data without unintentional ripple effects. This independence also means that the Shapefile format usually requires less storage space compared to topological formats, which maintain relationships between entities.

  This format is capable of storing a variety of information, ranging from discrete locations such as points of interest (points), to linear features such as roads, networks, or rivers (lines), and large spatial extents or boundaries such as districts, lakes, or parcels (polygons)[5].

  In total, according to the specification[6], fourteen different geometry types are supported. *(Null shape)* which represents no geometry, *(MultiPoint, Point, Polygon, and Polyline)* for two-dimensional geometries, *(MultiPointM, PointM, PolygonM, and PolylineM)* for two-dimensional geometries with a measurement attribute *(MultiPointZ, PointZ, PolygonZ, and PolylineZ)* for three-dimensional geometries, and finally a *(MultiPatch)* to store a collection of triangles in 3D space. It is important to note that a .shp file cannot contain multiple different geometry types.

  Additional data attributes can be attached to these geometries within the dBase tables (.dbf files)[7].

  Required components of a Shapefile:

  - .shp: The .shp is the main file of a Shapefile. It stores the primary geometrical data for the features. Each shape record in this file is comprised of a shape type (point, line, polygon, etc.), and the coordinates of the shape's vertices. For instance, a line is represented as a sequence of points, while a polygon is represented as a series of rings.
  - .shx: The .shx file acts as an index to the.shp file, improving the data retrieval speed. It stores the index position of each feature in the .shp file. When a spatial operation needs to access a specific feature in the .shp file, it first retrieves the index in the .shx file, which in turn directly points to the exact location of that feature's geometry in the .shp file, thereby significantly improving the efficiency of spatial operations.
  - .dbf - This dBase table file accompanies the .shp file to store attribute data for features. Each row in the .dbf file corresponds to a shape in the .shp file. The .dbf file uses a tabular structure where columns represent different attributes (e.g., name, area, population), and rows represent individual records or features.

  Additional files in a Shapefile:

  - .prj: The .prj file stores the coordinate system and projection information for the dataset. This ensures that the geospatial data can be correctly aligned with other spatial data in a GIS (Geographic Information System). This is critical for operations that involve multiple datasets, ensuring they all align correctly within the same geographic space.

- .sbn and .sbx: These files are used to further optimize spatial queries. They contain a spatial index of the features, which allows for faster spatial searches and operations, such as selecting features within a defined area.
- .cpg: The .cpg file describes the character encoding applied to the Shapefile. This file is used to interpret special characters correctly in the attribute data, ensuring that text data are displayed correctly across different systems.
- .xml: This optional file contains metadata associated with the Shapefile. Metadata may include a description of the data, its source, its creation date, the attribute definitions, contact information, and more. The XML format allows for easy sharing and interpretation of metadata across different platforms.

- **GeoJSON**

  GeoJSON is a JSON-based format that encodes geospatial data. It is feature-based, meaning that each individual object, complete with its geometry and attributes, is encapsulated within a single JSON object. This approach directly binds the attributes to the geometry, ensuring self-contained features.

  GeoJSON supports a variety of geometric objects, including simple structures (points, lines, polygons) and complex ones (MultiPoint, MultiLineString, MultiPolygon, GeometryCollection). GeometryCollection is a special type capable of containing multiple geometric objects.

  Like Shapefile format, GeoJSON is nontopological, treating each feature independently. Changes to one feature will not impact others[8].

- **DXF**

  DXF, or Drawing Exchange Format, is an open source vector file format that is widely used by designers, engineers, and architects for 2D and 3D drawings. It was created by Autodesk to enable sharing of designs between various CAD applications.

  DXF files are appreciated for their precision. They ensure accurate results in applications such as CNC machining or printing, with no loss of quality when changing the size of a vector file. However, they cannot be used for 3D printing directly and need to be converted to STL format first.

  The benefits of DXF files include their open source nature, enabling wide usability and collaborative work on the same design across different CAD programs. They also offer impressive detail scalability with up to 16-bit floating-point numbers.

  On the downside, DXF files store information as plain text, which may extend the transfer time of the design. They do not support application-specific CAD elements, and, while they can work with 3D elements, they fully support only 2D objects[9].

- **CityGML**

  CityGML is an open data model and XML-based format for the storage and exchange of virtual 3D city and landscape models. It is a common standard developed by the Open Geospatial Consortium (OGC) that defines classes and relations for the most relevant topographic objects in cities and regional models.

  One key aspect of CityGML is its focus on physical objects, their properties, relations, and extensibility to include user-defined attributes. It supports multiscale representations, and objects in CityGML can be classified according to their semantics, geometry, topology, and appearance.

  CityGML models are used for many applications, such as urban planning, noise and environmental simulations, energy demand estimations, and disaster management. Despite their complexity and large file sizes, the advantage of CityGML files lies in their detailed 3D representation and the ability to carry extensive metadata[10].

2.2 Coordinate Reference Systems

Coordinate systems are fundamental frameworks used in geospatial applications to accurately represent and locate features on the Earth's surface. They provide a standardized reference system for spatial data, allowing for seamless integration and analysis across different datasets and disciplines. They serve as the basis for representing geographic locations and are essential for various tasks such as mapping, navigation, spatial analysis, and visualization. These systems define a consistent spatial reference that enables the precise positioning of features and facilitates the measurement of distances, areas, and directions on the Earth's surface[11].

Coordinate systems consist of several components:

- **Coordinate system**
  Coordinate systems are essential for integrating geographic datasets and ensuring consistent spatial reference. A coordinate system provides a reference framework that allows geographic features, imagery, and observations (such as GPS locations) to be accurately represented within a shared geographic context.[11].
- **Reference Datum**
  A reference datum provides the origin, orientation, and scale of the coordinate system. It establishes a consistent starting point for measuring positions on the Earth's surface.[12].
- **Units of measurement**
  Coordinate systems use specific units to measure distances, angles, and other geometric properties. Common units include meters, feet, degrees, and radians.

Coordinate Transformations: Coordinate transformations are mathematical algorithms used to convert coordinates between different coordinate systems. These transformations enable data integration and ensure consistency when working with multiple datasets that may use different coordinate systems or reference datums.

There are different types of coordinate systems used in geospatial applications, each with its own characteristics and applications:

- **Geographic Coordinate Systems**
  A Geographic Coordinate System (GCS) is a spatial framework that uses latitude and longitude to precisely locate points on the Earth's surface. Latitude measures the distance north or south of the equator, whereas longitude measures the distance east or west of the prime meridian. Together, they provide a two-dimensional reference system for spatial positioning[13].

  In addition to latitude and longitude, a GCS incorporates other components, including an angular unit of measure, a prime meridian reference (typically Greenwich) and a datum based on a spheroid. The GCS establishes a graticule2.1, forming a grid system with parallels (lines of equal latitude) and meridians (lines of equal longitude). This graticule enables accurate spatial referencing and division of the Earth into quadrants. GCS plays a vital role in applications such as mapping, navigation, and spatial analysis, providing a global system for precise location determination and integration of spatial data between various disciplines[14].
- **Projected Coordinate Systems**
  A projected coordinate system is a two-dimensional planar system that is used to represent the three-dimensional surface of the Earth, which is spherical or spheroidal. This system is based on a geographic coordinate system and is used to provide a framework for representing spatial data in a two-dimensional space, while aiming to

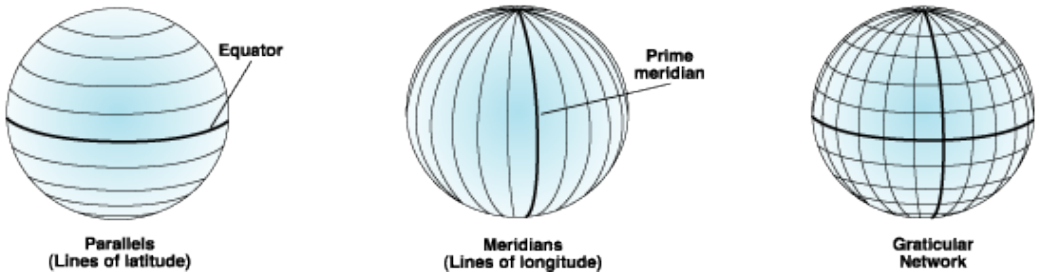


Figure 2.1. The parallels and meridians forming a graticular network [15].

maintain consistent measurements and relationships between points on the Earth's surface[16].

Locations in a projected coordinate system are identified by x- and y-coordinates on a grid. The x coordinate represents the horizontal position, while the y coordinate represents the vertical position. The grid's origin (0,0) can be located anywhere, depending on the specific design of the projection, and is not necessarily at the center of the grid. Positive values are generally to the right and above the origin, while negative values are to the left and below.

Different types of projections aim to preserve specific qualities, such as distance, angle, or area. However, it is important to note that no map projection can perfectly preserve all these properties. Instead, different map projections prioritize one or more of these properties, which inevitably leads to distortions in other aspects. The type and degree of distortion depend on the specific map projection used[17].

A crucial aspect of any projected coordinate system is the concept of scale, which refers to the relationship between the distance on the map and the distance in the real world. This can be constant throughout the map in conformal projections (like the Mercator projection) or can vary in equal-area projections.[16].

The choice of coordinate system depends on the specific requirements of the application. Factors to consider include the area of interest, the level of accuracy needed, the distortion characteristics of the projection, and compatibility with existing datasets.

The datasets provided by IPR use either the World Geodetic System 1984 or Křovák's coordinate system. The following subsections also include an overview of the Universal Transverse Mercator system, because it ended up being used in the final application.

2.2.1 World Geodetic System 1984

World Geodetic System 1984 (WGS84) is a global geographic coordinate system that serves as the standard reference framework for GPS positioning and satellite imagery. It is based on an ellipsoidal model of the Earth known as the WGS84 reference ellipsoid, which closely approximates the shape of the Earth. The ellipsoid is defined by a set of parameters such as the semi-major axis and the flattening factor[19].

This geodetic system defines positions on the Earth's surface using a geographic coordinate system (GCS) composed of latitude and longitude. The former measures the angular distance north or south of the equator, and the latter measures the angular distance east or west of the prime meridian, typically associated with Greenwich, England. The coordinates are expressed in degrees, with a latitude ranging from -90 ° (south pole) to +90 ° (north pole) and longitude from -180° to +180°[20].

As a global coordinate system, WGS84 enables accurate positioning and navigation on a worldwide scale. It serves as the foundation for GPS devices, allowing users to de-

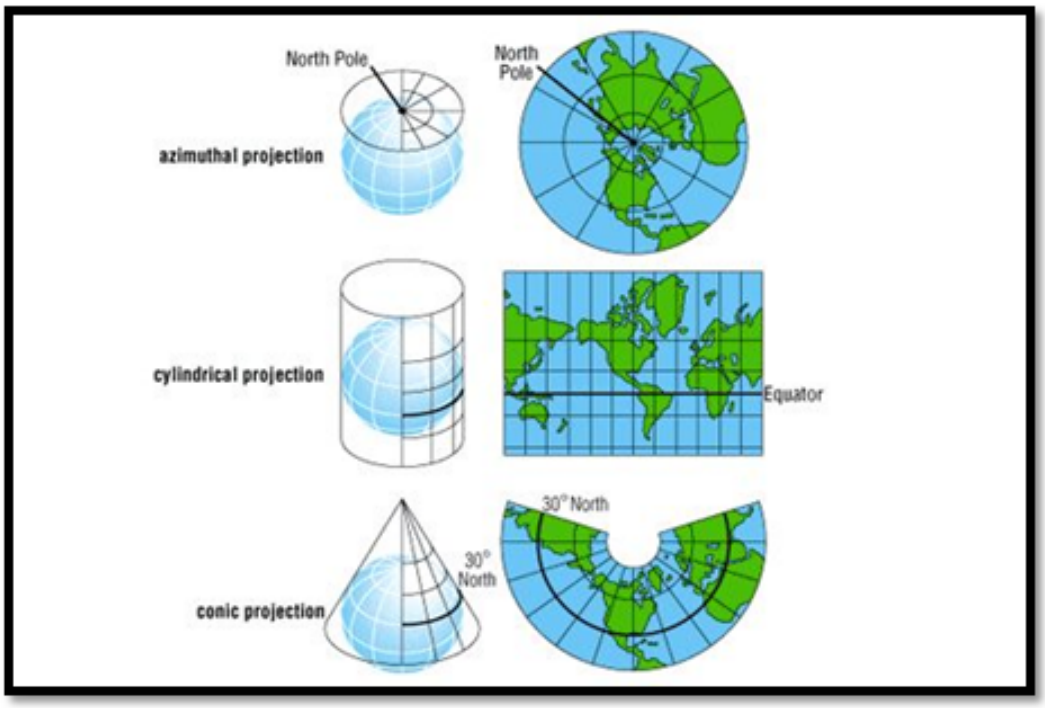


Figure 2.2. Different types of projections [18].

termine their precise location and navigate routes with accuracy. Additionally, WGS84 is widely adopted in satellite imagery, cartography, and geospatial data integration, facilitating interoperability and data sharing across different platforms and applications.

2.2.2 Universal Transverse Mercator

The Universal Transverse Mercator (UTM) is a system of geographic coordinates that provides detailed spatial reference information throughout the world. It is a specialized application of the transverse Mercator projection, which is a type of cylindrical map projection[21].

The UTM divides the world into 60 zones, each spanning six degrees of longitude, from 80 degrees south latitude to 84 degrees north. Each zone has its own central meridian. Zones are numbered 1 through 60, beginning at the International Date Line (longitude 180°) and moving east. For areas above 84 degrees north (North Pole area) and below 80 degrees south (South Pole area), the Universal Polar Stereographic Coordinate System is used instead[22].

Within each zone, coordinates are measured in meters. Easting is measured from the central meridian of each zone and northing is measured from the equator (with different baselines for the northern and southern hemispheres). This arrangement provides a two-dimensional Cartesian coordinate system that covers the globe, excluding the extreme polar regions.

The UTM system is particularly useful for regional-to-mid-scale topographic mapping and allows easy distance calculation between points within the same zone. One of its key advantages is that it reduces distortion within each zone because the transverse Mercator projection is applied on a narrow strip of the Earth's surface.

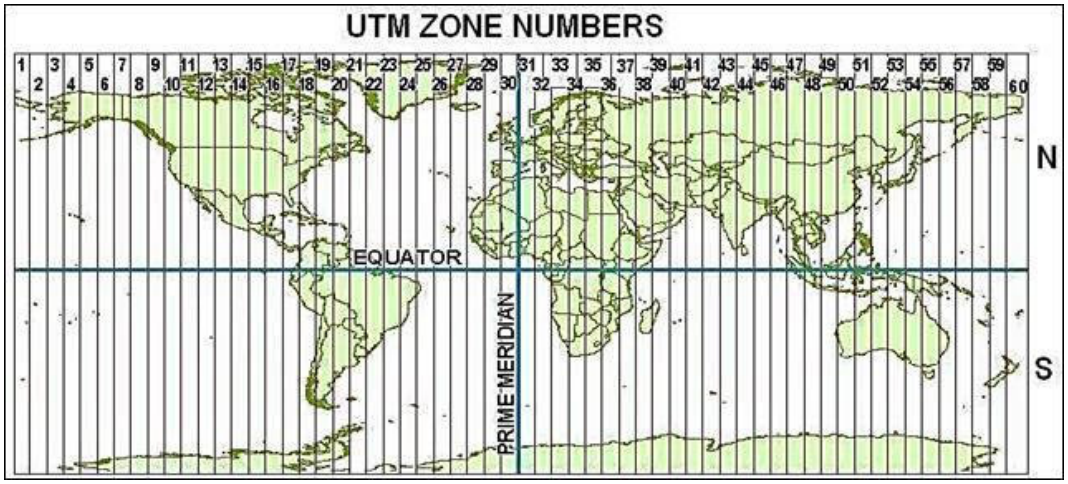


Figure 2.3. UTM zones [23].

2.2.3 Křovák's coordinate system

Krovák's coordinate system is predominantly used in the Czech Republic and is well-suited for the region because of its unique geographic characteristics. As a conformal, oblique, and equidistant system, it is designed to accurately represent data on a two-dimensional plane while maintaining the shapes, angles, and distances of the features being mapped[24].

These are its three key attributes:

- **Conformality**

  Křovák's coordinate system is conformal, meaning it preserves angles and shapes within a local areas. As a result, small areas on the map maintain their true shapes, making it suitable for applications that require accurate local geometries.

- **Obliqueness**

  Unlike many coordinate systems, Křovák's is oblique, which means that it is not aligned parallel or perpendicular to the Earth's equator. Instead, it is oriented to match the predominantly diagonal extent of the Czech Republic, which helps minimize distortions across the country.

  It is important to note that this obliqueness also manifests itself in the alignment of the coordinate system with respect to the north. Specifically, the system is rotated slightly in a positive direction, with a deviation ranging from 4.5 to 9.5 degrees from east to west[25].

- **Equidistance**

  The projection2.4 also maintains accurate distances along the meridians, or lines that run from north to south. This equidistant property is particularly useful for maintaining scale accuracy in directions important for the representation of the country.

Křovák's coordinate system is a valuable tool for tasks requiring precise measurements because of its real-world unit of measurement: the meter. This direct relation to real-world distances and areas greatly simplifies calculations and enables more accurate representations.

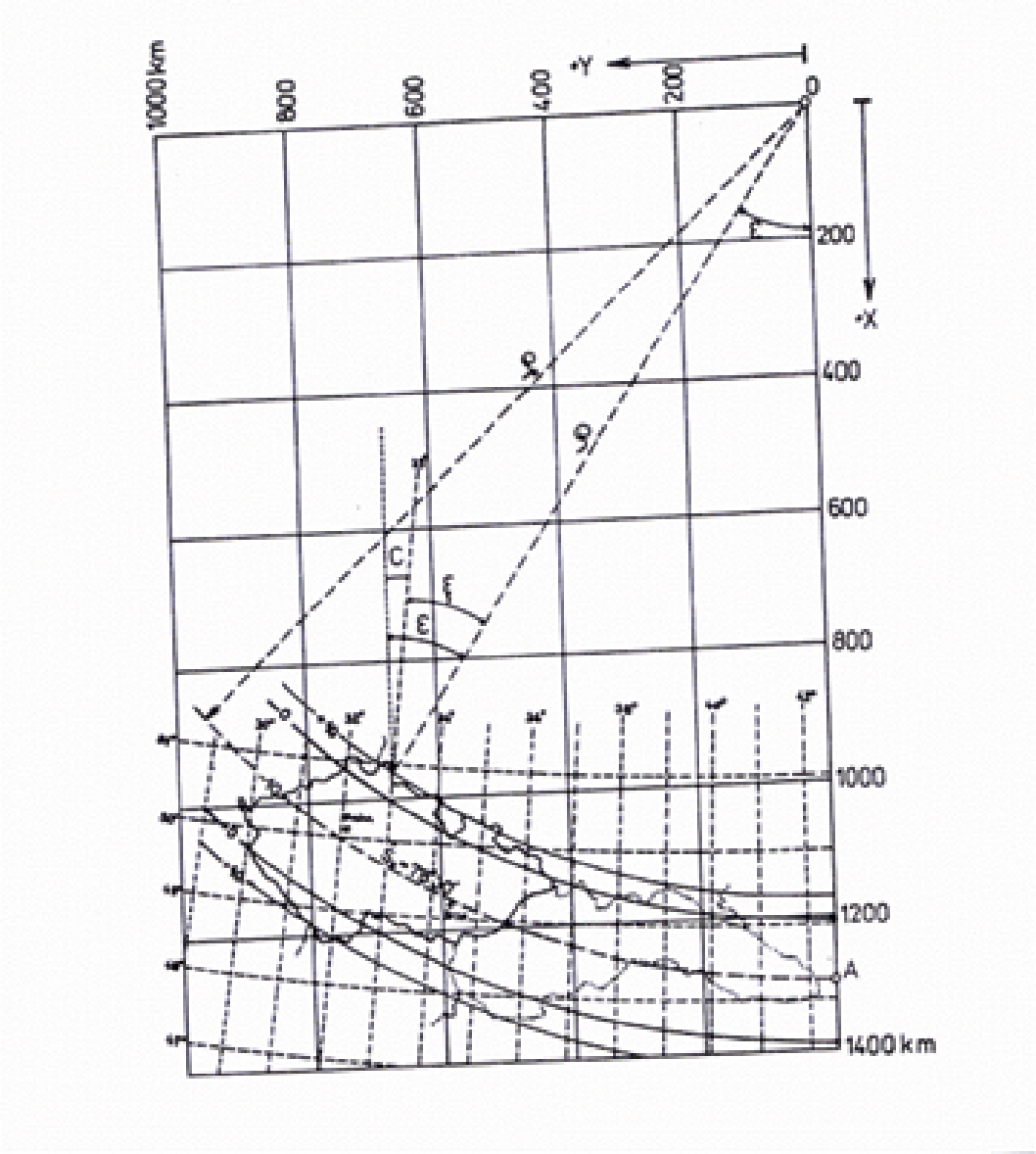


Figure 2.4. Křovák's projection [26].

2.3 Global Navigation Satellite System

Global Navigation Satellite System(GNSS) is used in people's everyday lives. This system includes multiple independent satellite constellations: the American GPS[27], the Russian GLONASS[28], the European Galileo[29], and the Chinese BeiDou[30] system. All of these satellite constellations operate on similar principles, but with varying numbers of satellites and specific operational nuances.

Each GNSS is a system consisting of three parts: the space segment, the control segment, and the user segment.

The space segment is a constellation of satellites orbiting Earth. For example, in GPS, there are at least 24 satellites, distributed in six orbital planes at about 20,200 kilometers above the ground. The extra satellites in the system, over the baseline of 24, help maintain coverage when the main satellites are serviced or decommissioned, and

can increase system performance[31]. Each satellite has several atomic clocks, which can keep the time very accurately, and even then, they are regularly corrected by even more accurate atomic clocks that are kept on the ground [32].

The control segment of each GNSS consists of ground-based stations that monitor, control, and maintain the satellite constellation. For GPS2.5, this includes a master control station, an alternate master control station, 11 command and control antennas, and 16 monitoring stations[33]. These stations track satellites as they pass overhead and collect data from their transmissions. These data are sent to the master control station, where the precise locations of the satellites are computed and navigation messages for the satellites are generated.

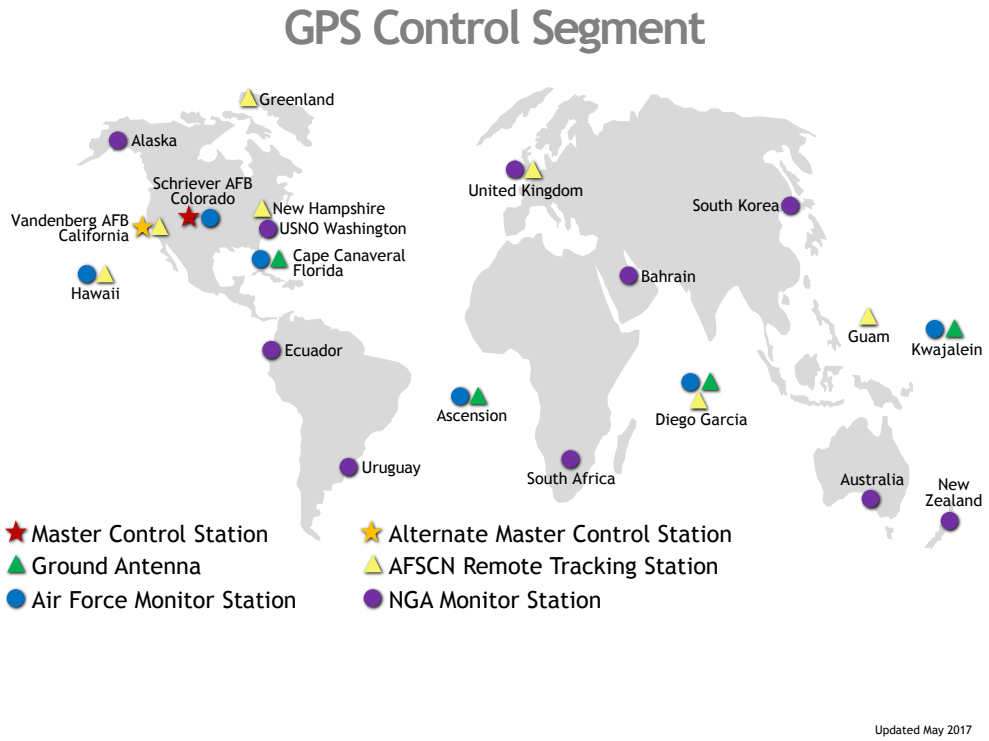


Figure 2.5. The GPS control segment [34].

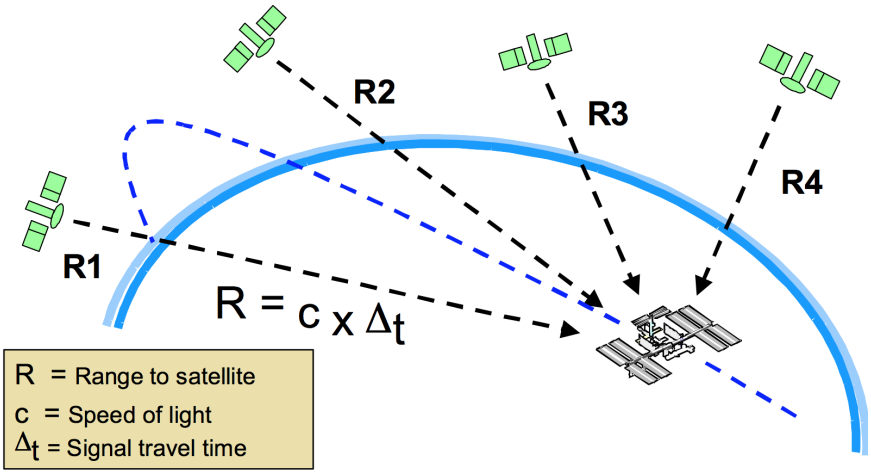
The user segment consists of all the devices that receive GNSS signals and use them to calculate their position. These devices can range from smartphones and cars to boats and aircrafts. More modern GNSS receivers are capable of receiving signals from multiple GNSS systems simultaneously, a feature known as multiconstellation reception. This is beneficial for a number of reasons: it increases the number of signals that can be used to calculate a position, improves accuracy, provides better coverage in challenging environments (such as urban canyons), and offers redundancy in the event that one system should have an outage[35].

To calculate the position, a receiver needs to process the signals from at least four different satellites. It calculates the distance from each satellite according to the time delay of the received signal. With signals from four satellites from one or more GNSS systems, the typical accuracy under open skies can be within a few meters[36]. However,

accuracy can be degraded in challenging environments such as cities due to signal multipath, blockages, and reflections.

To calculate a 3D position (latitude, longitude, and altitude) and time, a receiver needs signals from at least four satellites. Each satellite transmits a signal that includes its exact position and the time the signal was transmitted. The receiver records the time it received the signal and uses the difference between transmission and reception times to calculate the distance to each satellite. This is possible because the signal travels at a known speed, the speed of light.

The equations to calculate the position based on these distances are derived from the principle of trilateration2.6. In a 3-dimensional space, the signal from one satellite defines a sphere, with the satellite at the center and the calculated distance as the radius. This means that the receiver could be anywhere on the surface of this sphere.



$$(x_{iss} - x_1)^2 + (y_{iss} - y_1)^2 + (z_{iss} - z_1)^2 = (r_1 - \varepsilon_{clock})^2$$
$$(x_{iss} - x_2)^2 + (y_{iss} - y_2)^2 + (z_{iss} - z_2)^2 = (r_2 - \varepsilon_{clock})^2$$
$$(x_{iss} - x_3)^2 + (y_{iss} - y_3)^2 + (z_{iss} - z_3)^2 = (r_3 - \varepsilon_{clock})^2$$
$$(x_{iss} - x_4)^2 + (y_{iss} - y_4)^2 + (z_{iss} - z_4)^2 = (r_4 - \varepsilon_{clock})^2$$

Figure 2.6. The equations to obtain the receiver position [37].

With signals from two satellites, the receiver must be located on the circle created by the intersection of the two spheres.

With a third satellite, the receiver will be at one of the two points where this third sphere intersects with the circle from the first two spheres.

In a perfect scenario, the fourth satellite would simply confirm one of these two points. However, under real conditions, there are various sources of errors, such as the inaccuracy of the satellite's clock and the imprecision of the receiver's clock. Therefore, the fourth satellite is used to deal with these errors, particularly the offset of the receiver's clock relative to GNSS time[38].

The integration of multiple GNSS increases overall system robustness, providing a reliable service that can be used globally for various applications, including personal navigation, logistics, surveying, timing, and much more.

2.4 ARCore

ARCore[39], developed by Google, is a platform designed to make it easier to create augmented reality(AR) applications. It offers several different APIs that allow a phone to sense its environment, understand the world, and interact with it. Three key capabilities, namely motion tracking, environmental understanding, and light estimation, allow the user to see virtual content seamlessly integrated with the real world through the phone's camera[40].

2.4.1 Motion tracking

Motion tracking in ARCore is based on a process known as visual simultaneous localization and mapping (vSLAM)[40]. This process uses multiple sensors, including the device's camera and the Inertial Measurement Unit (IMU), to estimate the device's relative position and orientation in the surrounding environment. The SLAM process consists of three primary modules, Initialization, Tracking, and Mapping. There are two additional modules, Relocalization and Global Map Optimization, which enhance stability and accuracy[41].

- **Initialization**

  Initialization sets the stage for the vSLAM. It establishes the coordinate system and creates a preliminary map of the surroundings. It also involves identifying and tracking specific visual features in the environment, providing reference points for mapping and tracking later[42].
- **Tracking**

  Once initialization is complete, vSLAM begins tracking the device's motion. The camera's pose (position and orientation) relative to the map is estimated by matching features in the current image with those in the map. This is a complex task that often involves resolution of the Perspective-n-Point problem, which seeks to determine the camera pose given n 3D-to-2D point correspondences[41].
- **Mapping**

  The system continuously updates and expands the map by triangulating the 3D positions of features in the environment. This dynamic process allows for an increasingly detailed representation of the environment as the device moves around.
- **Relocalization**

  Sometimes, the device might lose track of its position due to rapid movement or a temporary occlusion of the camera. In such cases, relocalization techniques are employed to calculate the camera's position relative to the established map, restoring accurate tracking.
- **Global map optimization**

  Over time, small errors can accumulate in the estimated camera pose, leading to significant distortions on the map. Global map optimization reduces these errors, using loop closure techniques to identify when the device returns to a previously visited location. This error correction significantly improves the consistency and accuracy of the map[43].

2.4.2 Environmental understanding

ARCore tries to interpret the real world environment by detecting feature points and planes. It identifies clusters of feature points that lie on a common surface and presents them to the AR application as geometric planes. Users can then position virtual objects on these surfaces, and thus effectively integrate virtual and real elements.

However, there are currently still some limitations to the ARCore's abilities. For example, ARCore may struggle to identify flat, textureless surfaces (like a plain white wall), extremely bright or dimly lit environments, and reflective or transparent surfaces[44].

2.4.3 Light Estimation

ARCore's Light Estimation capability leverages the Lighting Estimation API to analyze and replicate a range of environmental lighting conditions. This comprehensive understanding of the lighting scene aids AR applications in adjusting the lighting of virtual objects, resulting in a more cohesive AR experience.

The API is capable of mimicking lighting cues such as shadows, ambient light, shading, specular highlights, and reflections, adding depth and realism to AR visuals. It also offers an environmental HDR mode, which employs machine learning to synthesize environmental lighting in real time, facilitating realistic rendering of virtual objects. This mode includes main directional light, ambient spherical harmonics, and an HDR cubemap for reflections[45].

Additionally, ARCore also utilizes Ambient Intensity mode to measure average pixel intensity and color correction for images, and Environment Probes to create environment textures for realistic lighting of virtual objects. All these capabilities work together to seamlessly integrate virtual objects into the real-world environment, enhancing the AR experience.

2.4.4 Other capabilities

There are many other capabilities that ARCore uses to improve the user experience.

- **Depth understanding**

  ARCore's Depth API facilitates more realistic AR experiences by understanding the size, shape, and distance of real-world objects. Key features include object occlusion for accurate rendering of virtual objects behind real-world ones, scene transformations that allow virtual elements to interact with real objects, and improved hit-test results for better object placement. Depth information also enables unique user interactions with AR objects, like allowing virtual content to collide with real-world elements. This feature is device dependent, as it is only supported by devices with enough processing power[46].
- **User interaction**

  Users can interact with the AR environment through simple actions like tapping on the screen. When the user taps the screen, a ray is projected from the camera. If it intersects any geometric planes or feature points, it will return information about the intersection. This allows the user to place an object in the AR environment.[47].
- **Oriented points**

  ARCore's ability to estimate the orientation of the surface from feature points allows the placement of virtual objects on angled surfaces, leading to a more convincing AR experience[40].
- **Anchors and trackables**

Anchors play a pivotal role in maintaining the relative positioning of virtual objects to real-world planes or objects. Anchors establish a connection between the virtual object and a plane or feature point, usually identified through a raycast hit. This prevents the virtual object from appearing suspended in space when ARCore updates the position of the geometric planes and feature points.

If the virtual object is not anchored to any trackable, it would maintain its original position even as the plane moves. On the other hand, if the virtual object is securely anchored, it would move with the plane or feature point, ensuring that it appears to stay in the same place.[40]

- **Cloud anchors**

  Cloud Anchors are a unique type of ARCore feature that facilitate shared and enduring Augmented Reality experiences. These anchors, unlike local anchors, are stored in the ARCore Cloud. This allows multiple users to participate in shared AR experiences through a single application, allowing for synchronous interactions in a digital environment. For example, if one user places a virtual sticky note on a refrigerator within the application, another user would be able to see the same note at the exact location[48].

  Users can adopt two primary roles within this framework: the host and the receiver. The host, who initiates the anchor, first constructs a 3D map of the vicinity around the desired anchor location. This is achieved by employing the phone's camera to scan the immediate environment. The captured spatial information is subsequently uploaded to the server for future use by receivers.

  On the receiver's side, the AR experience unfolds when they direct their camera towards the area originally mapped by the host. As this action triggers a resolution request to the server, the ARCore Cloud initiates periodic comparisons between the incoming images and the pre-existing 3D map. This process facilitates the precise determination of the receiver's position and orientation in the AR space[49].

- **Augmented images**

  This feature enables the AR application to recognize specific 2D images, such as QR codes or product packaging. When the camera detects a known image, it can trigger an AR experience, adding a layer of interactivity and personalization to the application[50].

2.5 Geospatial API

The ARCore Geospatial API[51] introduced in May 2022 allows developers to remotely attach AR content to any area covered by Google Street View, allowing the creation of AR experiences on a global scale. By harnessing device sensor and GPS data, it approximates the device's location and further enhances precision using Google's Visual Positioning System (VPS).

VPS uses Street View images from Google Maps captured around the world for more than 15 years. A deep neural network identifies and describes parts of these images that are likely to be recognizable over long periods of time. An example of such image parts can be seen in Figure 2.7. Pedestrians, cars, and other transient elements are filtered out to create a 3D point cloud of the global environment, forming the localization model that spans nearly all countries[51].

When a request is made to the Geospatial API, the neural network processes the pixels to identify recognizable parts of the user environment, matches them to the

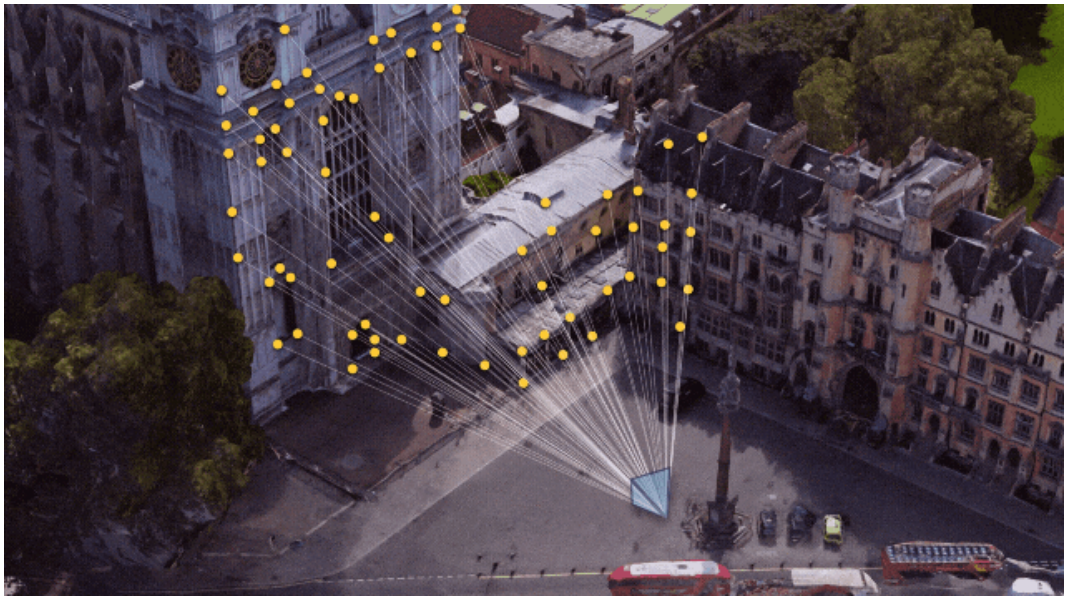


Figure 2.7. Feature points recognizable by VPS. [51].

VPS localization model, and computes the position and orientation of the device. This provides a location that is much more accurate than what GPS alone could offer.

The Geospatial API also harmonizes the user's local coordinates with the geographic coordinates from VPS, allowing developers to work within a unified coordinate system.

Developers can use the Geospatial API to place geospatial anchors almost anywhere in the world at a given latitude, longitude, and altitude without needing to manually map the space, following the WGS84 specification. It supports various types of Geospatial Anchors: WGS84 anchors, terrain anchors, and rooftop anchors, each suited to specific applications and scenarios. Their basic overview can be seen in Figure 2.8

	WGS84	Terrain	Rooftop
Horizontal Position	Latitude, Longitude	Latitude, Longitude	Latitude, Longitude
Vertical Position	Relative to WGS84 altitude	Relative to terrain level determined by Google Maps	Relative to rooftop level determined by Google Maps
Needs to be server-resolved?	No	Yes	Yes

Figure 2.8. Anchors provided by Geospatial API [52].

The Geospatial API opens up a range of applications for developers. For example, it allows users to navigate to specific locations with a level of precision that surpasses that of GPS, making it useful for locating objects in densely populated areas or navigating busy spaces[51]. Additionally, it enables the creation of compelling, immersive location-based AR experiences, without the need to construct and maintain maps of multiple locations. A comparison of GPS, cloud anchors and Geospatial API can be seen in figure 2.9.

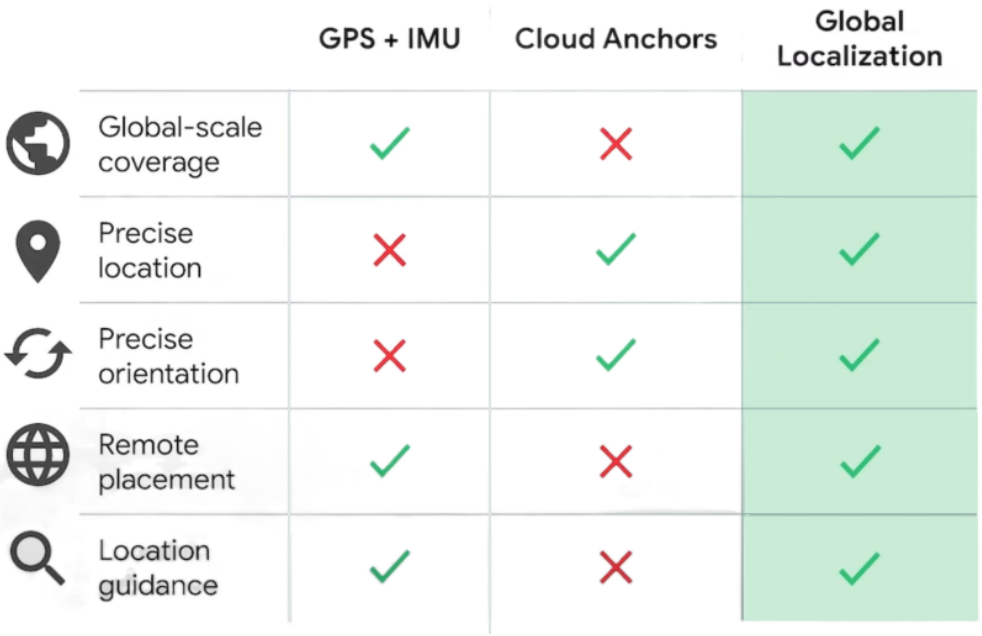


Figure 2.9. Comparision of GPS, Cloud Anchors and VPS [53].

2.6 Unity

Unity[54] is a versatile and widely-used development platform that provides powerful tools and features for creating augmented reality (AR) applications, games, simulations, virtual reality (VR) experiences, and more. With its intuitive interface, extensive asset library, and robust scripting capabilities, Unity offers a solid foundation for developing immersive and interactive projects across various domains.

Unity's cross-platform compatibility allows developers to target a wide range of devices and platforms. Whether it's smartphones, tablets, desktops, consoles, or even emerging technologies like VR headsets, Unity enables seamless deployment of applications on different operating systems, including iOS, Android, Windows, macOS, and more. This flexibility ensures compatibility and accessibility for the target audience of the project[55].

Unity's scene editor serves as a visual interface for designing and building virtual environments, whether they are AR scenes or other interactive experiences. The scene editor provides a rich set of tools for object placement, level design, lighting adjustments, asset management, and more. Its real-time preview capabilities allow developers to rapidly iterate, visualize, and refine their projects to achieve the desired outcomes[56].

Scripting in Unity, powered by the C# programming language, empowers developers to create custom behaviors, implement game logic, handle user interactions, and integrate external APIs or services. This powerful scripting system provides flexibility and extensibility, allowing the project to be tailored to specific requirements and allowing developers to bring their creative visions to life.

Furthermore, Unity's Asset Store serves as a valuable resource, providing an extensive library of pre-built assets, including 3D models, animations, audio files, scripts, and plugins. Leveraging these assets can significantly accelerate development time, allowing the project to focus on core functionalities and unique aspects. The Asset Store also

offers plugins and extensions that expand the capabilities of Unity, further enhancing the project's potential[55].

2.7 Augmented Reality

Now, that all of the core elements that are used to create the augmented reality application are described. It would be great to look at what Augmented Reality(AR) really is. The Oxford Learner's Dictionary[57] defines Augmented Reality (AR) as technology that merges computer-generated images with the real-world scene viewed through a device. In this context, AR can be seen as a tool that enhances the real-world environment by layering it with relevant digital information. This not only provides a more interactive experience but also improves the user's understanding of their surroundings. It is essential to clarify that AR is not simply an overlay of computer graphics. It is an advanced technology that provides a more interactive and contextually rich user experience[58].

Ronald Azuma's seminal research paper identifies three crucial elements that define Augmented Reality (AR)[59]. To start with, AR is a fusion of reality and digital illusion, seamlessly integrating the real world with the virtual reality.

Secondly, AR is interactive in real time. Users don't just observe a static digital world, they actively engage with dynamic AR components that respond instantly, creating an immersive and participatory environment.

Finally, AR is about depth and precision in positioning digital content. It is not a simple two-dimensional overlay but a complex, three-dimensional integration into the user's field of view. This ensures spatial coherence, positioning the digital augmentation perfectly in relation to the user's real-world context, giving the illusion that the virtual and the real exist in one cohesive space

On handheld devices like smartphones or tablets, AR achieves this by combining digital graphics with the device's camera view. Through AR glasses, this immersive experience is taken a step further by superimposing these graphics directly onto the user's field of vision.

The potential applications of AR are vast and varied. It is used in medical visualizations, where AR can improve surgical procedures and patient care by providing 3D visualizations of internal body structures[60]. In industries such as manufacturing and aviation, AR can help with maintenance and repair by overlaying instructions or information directly onto machinery or equipment.

AR also finds use in annotation, where digital information can be superimposed on real-world objects to provide additional context or explanation[61]. It is applied in robot path planning, where robots can use AR to navigate complex environments. In the entertainment industry, AR is revolutionizing gaming and media consumption by creating more immersive experiences.

In military operations, AR plays a crucial role in navigation and targeting, offering enhanced situational awareness to soldiers on the battlefield. These examples merely scratch the surface of the vast potential of AR, which continues to be explored and expanded across various sectors.

2.7.1 Making virtual objects seem more realistic

Creating realistic virtual objects is a fundamental goal in the development of augmented reality (AR) experiences. To achieve this, several key aspects must be considered and implemented effectively. This section explores various techniques and concepts that can enhance the realism of virtual objects in AR environments[62].

- **Occlusion**
  Occlusion is the effect of objects hiding other objects from view, depending on the perspective. Achieving realistic occlusion in AR requires a mechanism for detecting when occlusions occur and rendering images so that hidden parts of the AR objects are not displayed. The solutions include depth maps, 3D models of environments or contour models, each with its advantages and limitations[63].
- **Collision detection**
  For AR objects to interact realistically with real-world objects, the system must be able to detect collisions between them. The solutions are similar to those for occlusion: depth maps and virtual models.
- **Gravity**
  To make virtual and real objects appear to coexist in the same space, AR objects should behave as if influenced by gravity. Some solutions involve the use of inertial sensors in hand-held devices, such as smartphones, to compute the gravity vector[64].
- **Lighting and shadowing** Shadows and lighting are critical to enhance the realism of AR. The challenge is to make AR objects reflect light and cast shadows similar to those of real-world objects. This requires an estimation of the illumination conditions in an image. There are different lighting aspects to the scene that have to be mimicked based on real-world conditions[64].

  Ambient lighting refers to the overall light present in a scene, without a specific direction or source. Ambient light affects all objects in the scene equally, regardless of their location. This kind of light provides a base level of illumination but does not contribute to the creation of shadows.

  Directional lighting is a type of lighting that simulates the light that is being emitted from a specific source, such as the sun or a lamp. It is crucial to creating a sense of depth and texture in the scene.

  Shadow mapping is a technique used to create shadows from AR objects. Shadows are essential for grounding virtual objects in the real world and for giving a sense of spatial relationships among objects. To generate realistic shadows, the AR system needs to understand the position and intensity of light sources, the shape and orientation of the AR object, and the surfaces onto which the shadow will be cast[65].

  Light probes are used in the AR to sample and store information about the light available at a specific point in the environment.

2.8 Related work

In this section, we will take a look at the currently available projects that use the Geospatail API, and projects that do not use the API but are visualizing GIS data.

2.8.1 Applications using the Geospatial API

The Geospatial API has been utilized in various projects, showcasing its capabilities in visualizing geospatial data. These projects include both open-source initiatives and contest entries that aimed to demonstrate the API's features and possibilities.

- **Pocket Garden**
  One notable project is Pocket Garden[66], an interactive game where players can plant virtual seeds in real-world locations, creating and tending to their own personalized gardens. This innovative use of geospatial data merges the physical and digital

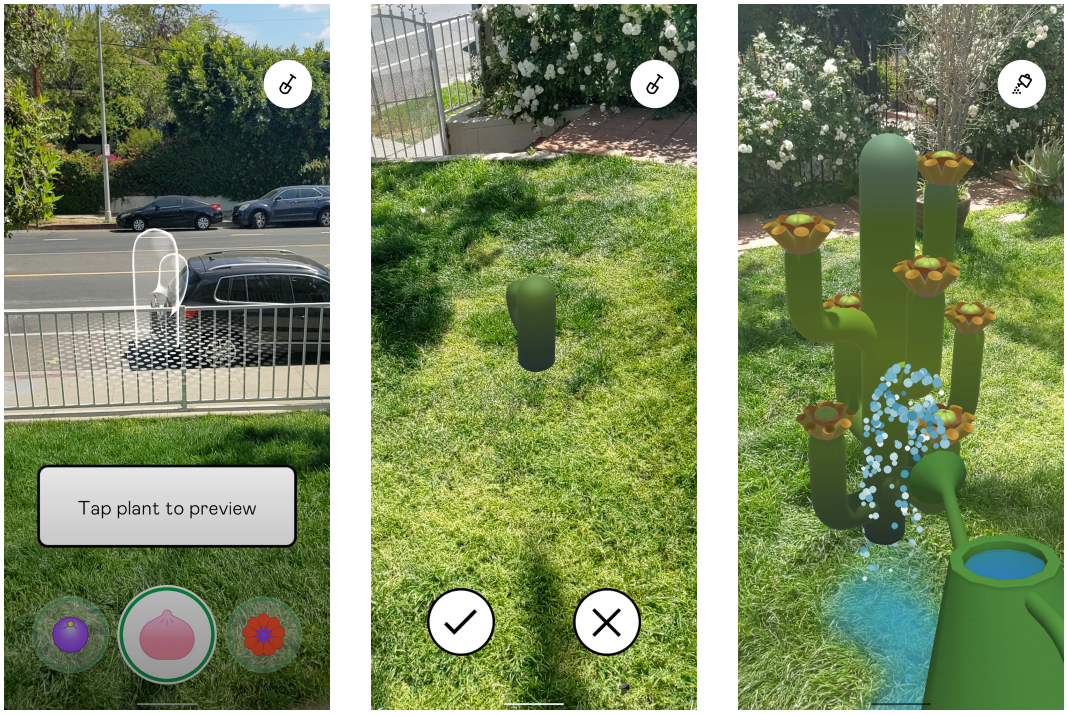


Figure 2.10. Pocket garden [66].

realms, providing a unique and engaging experience for players. The application can be seen in figure 2.10. It was created to showcase the abilities of Geospatial API and was released as an open-source project.

- **Contest projects**

  Additionally, the Geospatial API contest spurred the development of several exciting applications. These projects explored different aspects of geospatial visualization and leveraged the API's functionalities to create innovative solutions. Preview versions of these applications can be found on the official contest page, showcasing the diverse applications and creativity of the developers involved. The preview of these applications can be found on the official contest page [67].

- **Gorillaz Presents**

  Another notable demonstration of the Geospatial API's potential was the Gorillaz[68] concert experience. Through augmented reality technology, users could enjoy an immersive concert by the band Gorillaz at specific locations in New York and London. This fusion of music, geospatial data and AR technology provided a unique and memorable experience for concert-goers, blurring the lines between virtual and physical worlds, as can be seen in figure 2.11.

- **Google Live View**

  Google Live View[70] is an augmented reality (AR) feature available in the Google Maps app. It allows users to overlay digital information, such as directions and points of interest, onto the real-world view seen through their smartphone's camera. Using a combination of GPS, camera, and motion sensors, Google Live View provides real-time visual guidance to help users navigate and explore their surroundings. When enabled, it superimposes arrows, markers, and other graphical elements on the live camera feed, helping users to orient themselves and follow directions more easily. It can be seen in figure 2.12.

- **Geospatial Creator**

Figure 2.11. AR created for the Gorillaz concert [69].

Figure 2.12. Google's live view AR [71].

In May 2023 Google announced yet another tool for developers and artists to create more AR applications. Geospatial Creator[72] is a tool that combines the capabilities of ARCore and the Google Maps platform to enable developers and creators to design and launch 3D digital content in real-world locations. It leverages Photorealistic 3D Tiles to bring robust and engaging experiences to life.

The Geospatial Creator platform supports both Android and iOS devices, allowing developers to build world-anchored and cross-platform experiences. It enables users

to visualize their creations in the physical world through real-time localization and enhancement.

With Geospatial Creator, users can import real-world locations into their editing environment. By selecting a specific location, the platform retrieves the 3D geometry data for that area, providing a view similar to that of Google Earth. This 3D view allows users to preview and develop their augmented reality experiences, giving them a sense of flying through the space and exploring the environment.

The platform can be used by both developers and creators, enabling them to quickly build and publish immersive experiences. Photorealistic 3D tiles are available in 49 countries, enabling users to create content with high-quality visuals and realistic representations of the physical world.

It is currently supported by Unity and Adobe Aero[72].

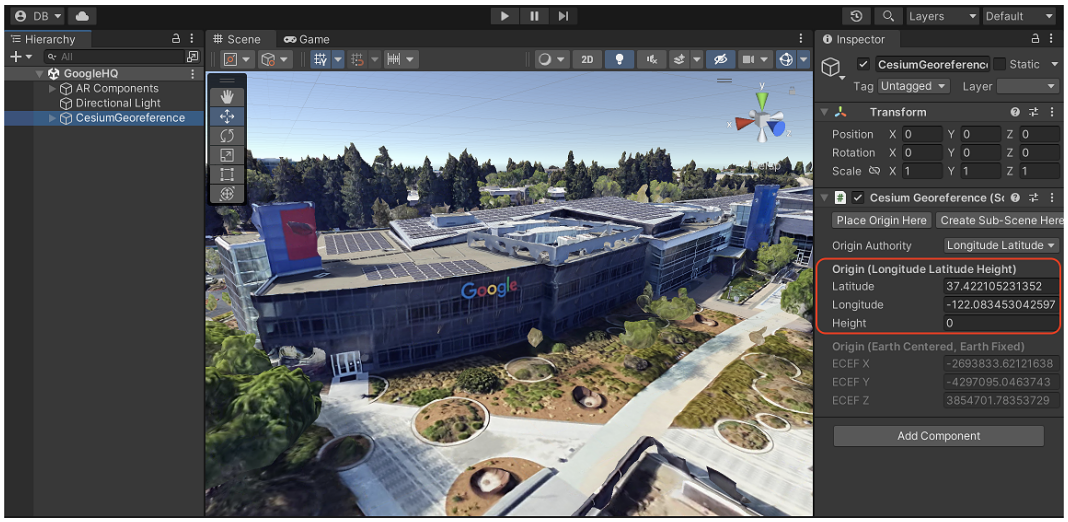


Figure 2.13. Google's Geospatial Creator in Unity [73].

2.8.2 Applications using Geographic Information System data

There are just a few applications that visualize GIS data in AR, though none of them use the Geospatial API. This section discusses some of them.

AuGeo

One of such applications is AuGeo, developed by Esri Online LLC[74]. AuGeo enables users to seamlessly integrate their GIS data into the real world through augmented reality. By leveraging the power of geospatial information, AuGeo provides a dynamic and interactive experience for users.

With AuGeo, users can bring their GIS data, specifically point data, to the real world using augmented reality technology. This means that users can visualize their geospatial points in their physical surroundings, allowing for a more intuitive and immersive understanding of the data.

Although AuGeo focuses specifically on points, its functionality extends beyond mere visualization. Users can interact with the augmented reality representations of the points, access additional information, and perform analyses right in the AR environment. This empowers users to make informed decisions and gain insights directly in the context of their physical surroundings.

It should be noted that AuGeo received its latest update in May 2021.

- **Augview** Augview[75] is an augmented reality (AR) application specialized in geospatial data visualization and asset management.

  At its core, Augview allows users to overlay geospatial data onto the real-world view captured by a device's camera. This enables users to view and interact with GIS data, such as infrastructure assets, utility networks, and spatial layers, directly in their physical surroundings.

  One of the key features of Augview is its ability to integrate with existing GIS systems. By connecting to GIS databases, Augview ensures that the displayed data are up-to-date and synchronized with the central GIS repository. This real-time connectivity ensures that field workers have access to the latest information and can make informed decisions on site.

  Users can view asset information, perform inspections, and access relevant documentation through the AR interface. The application supports functionalities such as spatial querying, measurements, and asset tracking, empowering field workers to efficiently manage and maintain assets in the field. The application can be seen in figure 2.14

  However, there was no new update since May 2018.

Figure 2.14. Augview application [75].

- **SiteVision**

  SiteVision by Trimble[76] is an augmented reality (AR) application specifically designed for the construction industry. It combines AR technology with Trimble's SiteVision Integrated Positioning System to provide construction professionals with accurate and real-time visualization of project data on their devices.

  Using Trimble's high-precision positioning technology, SiteVision achieves centimeter-level accuracy in location tracking. The system consists of a handheld device, such as a smartphone or tablet, connected to Trimble's SiteVision hardware unit via Bluetooth. Such a setting can be seen in figure 2.15. The hardware unit includes high-accuracy GNSS (Global Navigation Satellite System)

receivers and an inertial measurement unit (IMU) to capture precise positioning and orientation data.

To begin, construction project data, such as 3D models, design plans, and geospatial data, are prepared and converted into compatible format. These data include building designs, utility infrastructure, terrain models, and other construction-related information.

On site, the handheld device running the SiteVision app connects to the SiteVision hardware unit via Bluetooth. This enables the hardware unit to receive real-time positioning data from the GNSS receivers and the IMU, providing accurate location and orientation information[76].

With the SiteVision app, users can visualize and analyze project data directly in the field. The app overlays 3D models, design plans, and other project information onto the live video feed captured by the device's camera. This precise alignment of virtual data with the physical environment allows users to view an augmented representation in real time of the construction project on their device.

Figure 2.15. SiteVision Positioning system [76].

vGIS

vGIS[77] (Virtual Geographic Information System) is an AR platform designed to improve geospatial visualization and analysis in various industries, including utilities, telecommunications, construction and urban planning. By combining AR technology with geospatial data integration, vGIS provides users with accurate, real-time visualization of spatial information on their devices.

Before using vGIS, users begin by preparing their geospatial data, including maps, infrastructure assets, and spatial layers, in a compatible format. These data can

encompass underground utilities, pipelines, buildings, or any other relevant geospatial information.

To achieve centimeter-level accuracy, vGIS leverages high-precision positioning systems. By connecting the vGIS app to compatible positioning hardware, such as GNSS (Global Navigation Satellite System) receivers, the platform captures precise location and orientation data. This integration ensures the accurate alignment of virtual GIS data with the physical world.

Through the vGIS app, geospatial data is seamlessly overlaid onto the live video feed captured by the device's camera. This can be seen in figure 2.16. As users view their surroundings through their device, they witness the real-time integration of virtual GIS data into the physical environment. This augmented reality visualization allows for a seamless blending of spatial information with real-world surroundings.

vGIS empowers users to effectively visualize and analyze geospatial data in the field. By superimposing 3D models, infrastructure assets, or other spatial information onto the real-world view, users gain a better understanding of the site. They can detect clashes, assess asset conditions, and perform measurements or spatial analyses directly in the augmented reality environment[77].

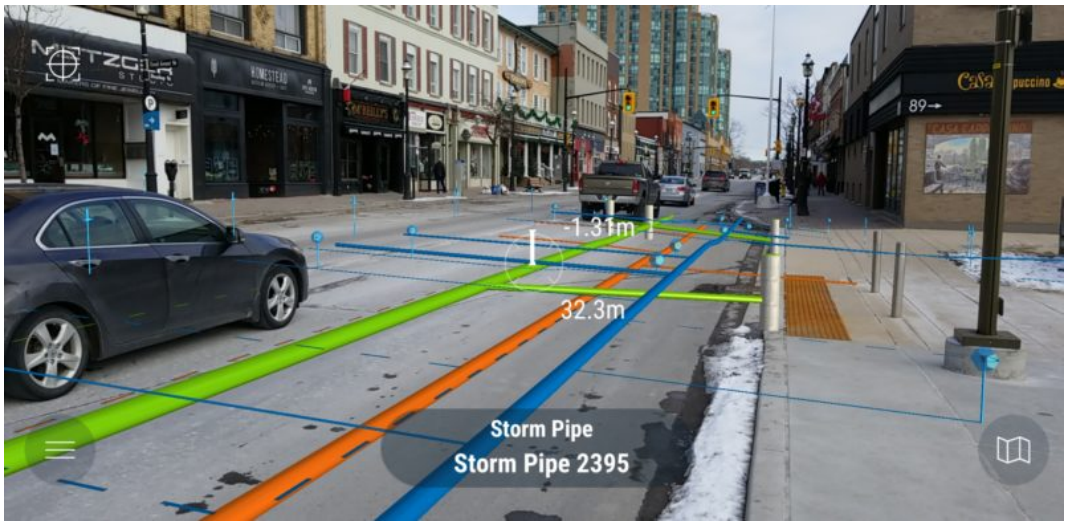


Figure 2.16. vGis visualization [77].

2.8.3 Summary

In summary, existing GIS data visualization applications, including vGIS, SiteVision, and others, currently require additional hardware to maintain the accuracy of the position of the device. However, the Geospatial API has the potential to eliminate the need for such additional hardware. Although there are no applications utilizing the API in this manner, it presents an excellent opportunity to explore and test the limits of the API's capabilities.

Chapter 3
Design

In this chapter, we will talk about the design of the application. First, we will discuss the reasoning behind choosing one of the data sets and then go further into designing the application and its core elements.

3.1 Why underground utilities?

There are many different data sets at the Prague geoportal. I considered these points when choosing the data set that would be used for this work:

- **Is it suitable for AR?**
  The first thing to consider is whether the data set is really suitable for AR visualization. Is there anything to visualize?
- **Is it useful?**
  The other deciding parameter is whether visualization of such data would provide the user with useful information if I omit extreme use cases.
- **Is it not already available?**
  The last deciding factor is whether the data set is not already easily accessible by different means, for example, Google Maps. If the visualization brings something new.

At this point, I could divide the remaining datasets into two groups. Those that would require some kind of navigation for them to be useful and those that would not need that and would instead visualize data around the user.

With the consideration of this work goal, which is to visualize some data, I have decided that navigation to some place might not be the best option. Therefore, I was left with only a few options. The remaining data sets were:

- **Parking zones**
  The application would visualize what type of parking the user's car is currently standing on.
- **Underground utilities**
  The application would visualize what utilities are going underneath the user and around him.
- **Land use plans**
  The application would visualize how the ground around the user is divided into lands.

In the end, I decided to choose underground utility lines because it felt the most interesting and if it turned out to be precise enough, it could even be the most useful one.

3.2 Application Design

As mentioned before an AR application must satisfy the three key characteristics. It should combine the real and virtual world, be interactive in real time, and finally position the virtual object in the 3D world instead of being just an UI. With these three characteristics in mind, these are the features that the application should have.

- **Markerless AR**
  The application should utilize markerless AR technology to seamlessly blend virtual objects representing underground utilities with the real world environment. This allows users to view and interact with the utilities without the need for physical markers or QR codes.
- **Accuracy**
  The application should aim to maximize the accuracy of the visualized objects. Especially in this case as it is working with real-world data.
- **Data Integration**
  The application has to use the data provided from the Prague geoportal and use them to generate the objects at the right place.
- **User-Friendly Interface**
  The application should have an intuitive and user-friendly interface to facilitate ease of use. The UI should provide clear instructions on how to interact with the AR elements and access additional features. The user should be able to see how accurate the visualization currently is.
- **Interactive Information Display**
  Users should be able to interact with the visualized utility lines in real time. They should be able to tap on a utility line to obtain detailed information such as the type of utility, the depth, and other associated data. This information can be displayed as a pop-up or in a sidebar panel.
- **Customization options**
  The application should provide customization options to enhance the user experience. Users should be able to filter the types of utility lines they want to see, adjust the color or representation of utility lines, and customize other visual settings. Additionally, options for adjusting the size of the pipes or enabling/disabling occlusion culling could further enhance customization.

3.3 UI design

This application UI should include these components:

- **Calibration screen**
  When the application starts, it is necessary to calibrate the position of the user. There should be a screen dedicated to calibration, which will stay there until the device's position is accurate enough. There can also be a menu access button, as the user may want to set some settings before completing the calibration. A sketch can be seen on the left in Figure3.1.
- **Main screen**
  Once the calibration is completed, the user should get to the main screen of the application. There should not be many UI elements on this screen, as it is essential to give the user as much free space as possible so that they can see and interact with

the AR. The main screen should have a button that allows the user to enter the menu. A sketch can be seen in the middle in Figure3.1.

- **Menu** After clicking on the menu button, a menu should open. This menu can cover the whole screen, as there is way too much information in it to make the AR interactable while being in the menu. The menu contains buttons that either open a different page in the menu or set the settings for a utility line. The user should be able to set the visibility of the line and its color. In addition to this, there should also be a close button to return to the main screen and a button to access the settings. A sketch can be seen on the right in Figure3.1.
- **Settings** Here should be the main settings of the application. Details of the settings available to the user should be specified during the implementation.
- **Color Change Screen** When the user wants to change the color of a line, there should be a screen where they can decide what color they want to use. If it is done by sliders, it would be better to make the sliders correspond to the HSV model instead of the RGB model, as it is more intuitive.

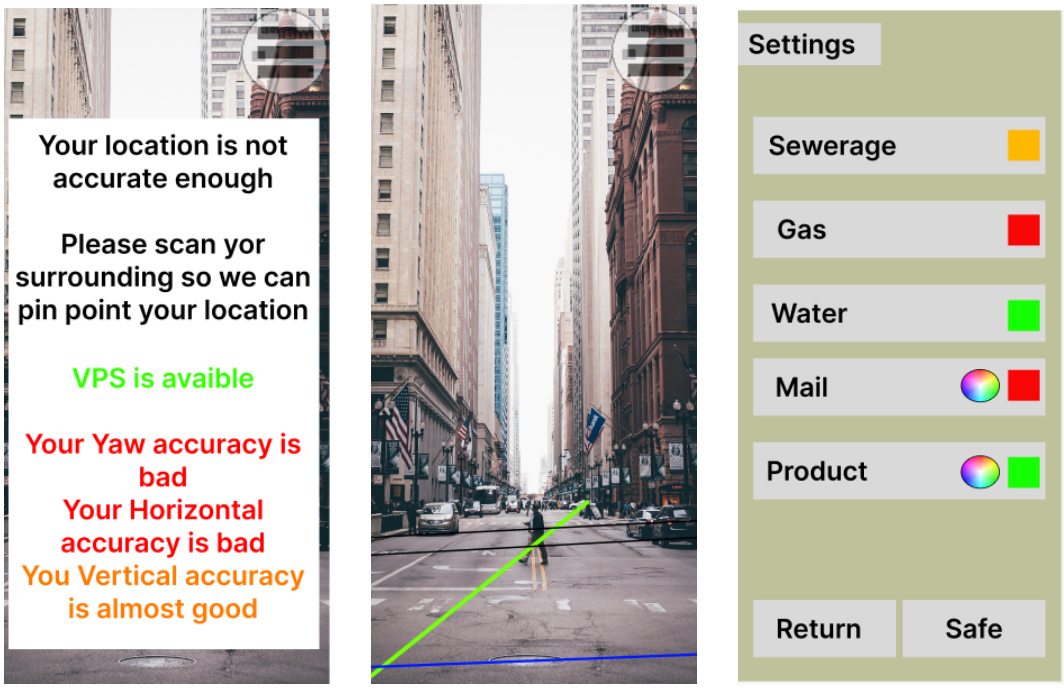


Figure 3.1. A design of the calibration, main, and menu screen.

Chapter 4
Implementation

In this chapter, we will talk about the details of the implementation and the problems and their solutions that I have encountered during the development of the application. I started with just the data files in WGS84 and from that I have built a whole application to visualize their contents. This chapter will follow the process of getting these files to appear on a user's phone, starting with the parsing of the data, continuing with the creation of a custom search file, and finishing with the visualization in the AR.

4.1 Parsing the input data

The first step in visualizing the data was the ability to read them. Therefore, this section will describe the parsing process and how to access individual records quickly.

Parsing the .shp file

I had no options when it came to choosing the file format because, at that time, the only format available for the utility lines for the entire Prague was the shapefile.

To parse the file correctly and access the data stored inside, the application follows the predefined format, which can be found in the shapefile documentation[6]. First, the application parses the main header, which can be seen in Figure 4.1, to retrieve basic information about the contents of the file. The header is followed by records that, in this case, are of type PolylineZ. To parse them, the application follows the format shown in Figure 4.2.

Position	Field	Value	Type	Number	Byte Order
Byte 0	Shape Type	13	Integer	1	Little
Byte 4	Box	Box	Double	4	Little
Byte 36	NumParts	NumParts	Integer	1	Little
Byte 40	NumPoints	NumPoints	Integer	1	Little
Byte 44	Parts	Parts	Integer	NumParts	Little
Byte X	Points	Points	Point	NumPoints	Little
Byte Y	Zmin	Zmin	Double	1	Little
Byte Y + 8	Zmax	Zmax	Double	1	Little
Byte Y + 16	Zarray	Zarray	Double	NumPoints	Little
Byte Z*	Mmin	Mmin	Double	1	Little
Byte Z+8*	Mmax	Mmax	Double	1	Little
Byte Z+16*	Marray	Marray	Double	NumPoints	Little

Note: X = 44 + (4 * NumParts), Y = X + (16 * NumPoints), Z = Y + 16 + (8 * NumPoints)
* optional

Figure 4.2. Description of the polylineZ record. [6]

Position	Field	Value	Type	Byte Order
Byte 0	File Code	9994	Integer	Big
Byte 4	Unused	0	Integer	Big
Byte 8	Unused	0	Integer	Big
Byte 12	Unused	0	Integer	Big
Byte 16	Unused	0	Integer	Big
Byte 20	Unused	0	Integer	Big
Byte 24	File Length	File Length	Integer	Big
Byte 28	Version	1000	Integer	Little
Byte 32	Shape Type	Shape Type	Integer	Little
Byte 36	Bounding Box	Xmin	Double	Little
Byte 44	Bounding Box	Ymin	Double	Little
Byte 52	Bounding Box	Xmax	Double	Little
Byte 60	Bounding Box	Ymax	Double	Little
Byte 68*	Bounding Box	Zmin	Double	Little
Byte 76*	Bounding Box	Zmax	Double	Little
Byte 84*	Bounding Box	Mmin	Double	Little
Byte 92*	Bounding Box	Mmax	Double	Little

* Unused, with value 0.0, if not Measured or Z type

Figure 4.1. Description of the main file header [6].

- **Accessing a single record with a known record number**

  The .shx file makes it possible to avoid searching record-by-record when a ceratin record needs to be retrieved from the .shp file. It has the same file header as the .shp file, but instead of long records, the header is followed by 8-byte-long elements, which contain the offset and length of the record in the .shp file. So, when the id of the record is known, it is possible to quickly count where to look for its offset and position in the .shx file, and then use these values to retrieve the record from the .shp file.

- **Parsing .dbf file**

  Once again, the application follows the preset format that can be found in the documentation [78]. Unlike the .shp file, the size of the individual records is known in advance and is included in the header. So when there is a need to find just one record, it can be done by calculating its offset and quickly accessing it.

4.2 Acceleration Structure

The dataset includes information about more than 2 million utility lines. However, the application needs to access only information about the utility lines that are close to the user. It would be impossible to go through all of the lines during run-time and check if they are in the users' vicinity. Instead, it is better to use some kind of acceleration structure that can help to find the right utility lines quickly.

The problem can be formulated as follows. Given the current position of the user, identify all utility lines within a specified distance from them.

Although the WGS84 coordinate system, which is not planar, is being used, it is possible to consider it to be planar, as the area is considerably small. On top of that,

the application visualizes all the utility lines that are beneath the user no matter how far they are, and thus this problem can be narrowed into 2D.

And once it is in 2D it is very easy to come up with a great acceleration structure that will allow finding the utility lines close to the user very quickly. A gird.

A bounding box can be constructed from the minimum and maximum longitudes and latitudes of the whole data set. The bounding box for the dataset used can be seen in Figure 4.3. It can be divided into areas, and thus create the grid. However, if it is divided into many small areas, many of them may not have any utility lines inside. Instead, it is better to first divide the plane into bigger zones, which if they contain any utility lines in them, can then be divided into smaller areas, thus creating a hierarchical grid. With the heirarchical grid ready, it is possible to take the boundaries and quickly calculate the index of the grid cell from it and, thanks to it, find where the user is. On the basis of the distance in which the utility lines should be visualized, the cell's neighbors can be taken into consideration, too. By doing this, the list of utility lines that are potentially close to the user can be retrieved quickly.

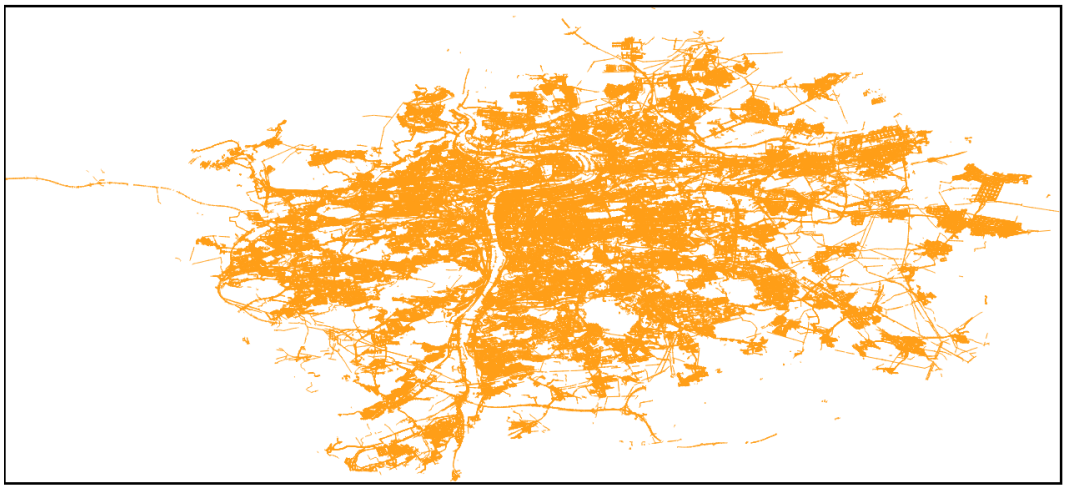


Figure 4.3. These are all the utility lines in the dataset, surrounded by a bounding box(black).

To create the first partition into zones, the algorithm described in Figure 4.4 can be followed. After sorting the utility lines into zones, the same process can be repeated for each of the zones that have at least one of the lines inside them. It could be repeated many times like this to partition the space more and more, but for the purposes of this application,partitioning it just twice is enough. The first partition divides the space into zones of approximately one square kilometer. The second partition further divides the zone into areas of approximately 20 meters squared.

It would be great to save the grid in some format that would allow a quick access to the ids of the lines based on the users location. It would be much faster and also more memory efficient than having to load the whole structure every time. That is why I proposed a new format that safes the grid in such a way that it is possible to quickly retrieve the required data. It has a header of 72 bytes, which contains all the necessary

Algorithm 1 Filling a grid with lines

```
List < int > [,]zones = InitializeList()
for each utilityLine ∈ utilityLines do
    for i = 0; i ¡ utilityLine.Points.Count - 2 do
        pointA = utilityLine.Points[i]; pointB = utilityLine.Points[i+1]
        indexesA = getIndexes(pointA); indexesB = getIndexes(pointB)
        for x = min(indexesA,B.x); x ¡= max(indexesA,B.x) do
            for y = min(indexeA,B.y); y ¡= max(indexesA,B.y) do
                if IsIn(pointA, pointB, x, y) then
                    zones[x,y].Add(utilityLine)
                end if
            end for
        end for
    end for
end for
data = RequestDataFromServer(idsToLoad)
ProcessData(data)
```

Figure 4.4. PseudoCode of how to sort the utility lines in the grid

data to calculate the index of the outer and inner grids based on users' location. With the indexes, it is possible to extract the ids of the utility lines only for the corresponding area. The proposed format is described in detail in Appendix B. It is used to get access to the data quickly and efficiently, just by reading from a file.

4.3 Server side

There are many different ways in which I could have set up the server side, but I have chosen to use ASP.NET, mainly because it also uses C#. When the client makes a request, the server retrieves the necessary data from the shapefile or the grid file. Before sending the data back to the client, it serializes them into a Json file. The server is designed to accept several different API calls to prevent sending redundant data. A brief description of the API request follows:

- **BinHeader**
  This is a GET request that on call returns the header of the grid file described in the previous section.
- **GetIds**
  GetIds is also a GET request. The request takes the indexes of zones and areas. The server uses the grid file to find the corresponding list of ids and sends the list back.
- **DBFfieldNames**
  This is a GET request, which on call returns the information about the fields in the .dbf file.
- **SHPPostData**
  This is a POST request, which takes a list of ids from the body of the request and returns a list which has all the points information for each of the requested ids.
- **DBFPostData**
  This is also a POST request, which takes a list of ids from the body of the request. It returns a list with the row of the .dbs file belonging to the requested ids.

Initially, all requests were designed as GET requests; however, the list of ids was sometimes too big to fit into the URL, so the two requests taking in the ids had to be changed to POST requests. The usage of the requests will be discussed later in the client description.

4.4 Setting up the project in Unity

When starting a new AR project in Unity, I had two options. Either I could have set up the whole project manually, or I could have used a template offered by Unity in the new project creation menu. The template adds all the necessary packages and sets up a basic scene. In contrast, if I started with an empty project, I would have to add these components manually. The packages that would have to be added are AR Foundation and the ARCore XR Plugin.

If I started with an empty project, I would also have to add these game objects to the scene, because they take care of the augmented reality.

- **AR Session**
  The AR Session controls the lifecycle of an AR experience, such as motion tracking, environmental understanding, and life estimation. It also checks if the user's device has support for AR and if an ARCore/ARKit needs to be installed or updated.
- **AR Session origin**
  It's purpose is to transform trackable features into their final position, orientation, and scale in the Unity scene. It should have the AR Camera as a child object, so when there is a necessity to start the AR experience in a specific place in the Unity scene, adjusting its position makes it possible.

To use the Geospatial API in my project, I had to add another package. This must have been done, regardless of whether I had started with a template or not. The package is called the ARCore Extension. At the time of writing, this package was not available from the Unity registry and had to be added from a GitHub URL [79]. With the package added to the project, I had to add the ARCore Extensions object to the scene. Furthermore, I had to enable the ARCore API in a new Google Cloud Platform project. After enabeling the ARCore API, I had been able to generate credentials which the application uses to authorize the application calls to the API. The process of generating the credentials and setting up the authorization is described on the Geospatial API official page[51].

4.5 Communication with the server

The goal was to make the amount of communication between the server as small as possible and to prevent sending redundant data. That is why the application keeps most of the received data in memory, within some set limit. The application uses two separate dictionaries. One for cells and the ids of the utility lines going through them, and the second for the utility lines themselves. The reason for separating the two is that the utility lines are going through multiple cells. If I were to keep the data on the lines for each cell, I would have a lot of unnecessary duplicit data. By keeping only the ids of the lines connected to the cells, the application can check if the line data are not already available before requesting them from the server again.

The first thing the application retrieves from the server is the grid file header and the list of fields from the dbf file. The received data is kept in the device memory for the

rest of the run-time. The next request is made when the user calibrates their position (this will be described later), and when it is known what data should be visualized. The indexes of the grid cell that is to be visualized can be counted by using the information in the grid file header. The header includes the information on the minimum and maximum longitude and latitude of the bounding box of the entire data and the size of the outer and inner grid cells. From this the indexes of the outer grid cell can be calculated. The indexes can then be used to calculate the minimum and maximum latitude and longitude of the cell. From there, the process can be repeated, but with the size of the inner cells instead. Thanks to this, it is possible to obtain all the indexes that are necessary to identify the cell in which the user is and for which data about the utility lines should be retrieved.

Instead of immediately requesting the ids of the utility lines from the server, the indexes are used to check if the ids are not already available in memory. If not, the application sends a request to the server to get them. If the ids are already present, they are loaded from the memory instead. Once the ids are available, either from the memory or from the server, they are cross-referenced with the utility line data in the memory. If any data for the ids are not within the memory, they are added to a list of needed information. Finally, the list is used to make another request to the server. This approach minimizes data redundancy and optimizes communication with the server.

The pseudocode for this process can be seen in Figure 4.5

Currently, the application uses a simple rule to manage the data received in memory. When the amount of data reaches a given threshold, the lists are cleared. To increase efficiency, I suggest clearing only the data that have not been used recently.

Algorithm 1 Requesting data

```
1: GridIndexes = GetGridIndexesByCurrentPosition(latitude, longitude)
2: if IndexesLoaded(GridIndexes) then
3:     ids = GetIndexes(GridIndexes)
4: else
5:     ids = RequestIdsFromServer(GridIndexes)
6: end if
7: List idsToLoad = new List
8: for each id ∈ ids do
9:     if not IdLoaded(id) then
10:        idsToLoad.Add(id)
11:    end if
12: end for
13: data = RequestDataFromServer(idsToLoad)
14: ProcessData(data)
```

Figure 4.5. Pseudocode of how to obtain data of the utility lines in a given cell

4.6 Utility line visualizaiton

When all the data for the utility lines that pass through the users' surroundings are ready, it is time to visualize them.

When I implemented this part, I had to take into account the fact that I would be using the Geospatial API and the anchoring system it provides. It is essential to have the anchor close to the user, because the accuracy decreases with larger distances. If there is a one-kilometer long utility line and the user is not at its starting point, it cannot be anchored there because it would lead to high accuracy errors.

With this in mind, I came up with two possible solutions. One was to work with the entire utility line at once and continually re-anchor it when necessary. This would lead to making the utility line seamless, but the re-anchoring would cause the line to move and change position. The second option was to divide the line into smaller lines based on the gird cells. Each part would have its own anchor, and so there would be no need for re-anchoring. In perfect conditions, the lines would still seamlessly connect and would not suffer from the re-anchoring movement. In not perfect conditions, the lines might not connect and be slightly misaligned. The size of the misalignment would depend on the accuracy.

The first option would require me to implement a system in which I keep track of what utility lines are currently active and decide where to anchor them as the user moves. A viable solution would be to re-anchor them every time the user moves into a new grid.

The other option would require me to implement a way to divide the lines into smaller pieces belonging to the given cell. I could then visualize the pieces inside the cell and its neighboring cells. Each time the user moves to a different grid, I would remove the cells that are no longer needed and generate the new neighboring cells.

I decided to use the second option because I wanted to limit the movement of the lines as much as possible. Although I knew that I might have to deal with the lines not being seamlessly connected.

That is why the application loads not only the cell in which the user is currently in, but also the cells around him, so that the visualization seems natural. This is done by simply adding and subtracting one from the inner cell indexes, and controlling if it is necessary to adjust the outer cell indexes.

Once all the utility line data are loaded, the application takes cell after cell and finds out the points of the lines that are inside and where the line intersects the cell. It then generates a mesh for the lines based on the points and assigns them to the appropriate cell.

For the generation of utility lines, I have used a script called Unity Plumber created by Federico Casares[80]. I had to make some changes to fit the application needs, but the main functions remained the same.

It is important to note that to generate the mesh, the WGS 84 coordinates have to be projected into a planar coordinate system that works with meters.

I wanted to be as accurate as possible, so I first tried to use the Křovák's coordinate system as it should have given the most accurate results for the Czech Republic. At first, when I generated the utility lines inside Unity without using the Geospatial API, there were no visible problems, and everything seemed to work. However, when I started testing the visualization, the lines had huge gaps between the borders of the grid. The comparison between the current version and the version in which the Křovák's coordinate system was used can be seen in Figure 4.6.

At the time, I did not know if the Geospatial API was not accurate enough or if there was a mistake somewhere else.

After conducting some investigation, I discovered that the reason for this was that the Křovák's coordinate system was not aligned with respect to the north. This resulted

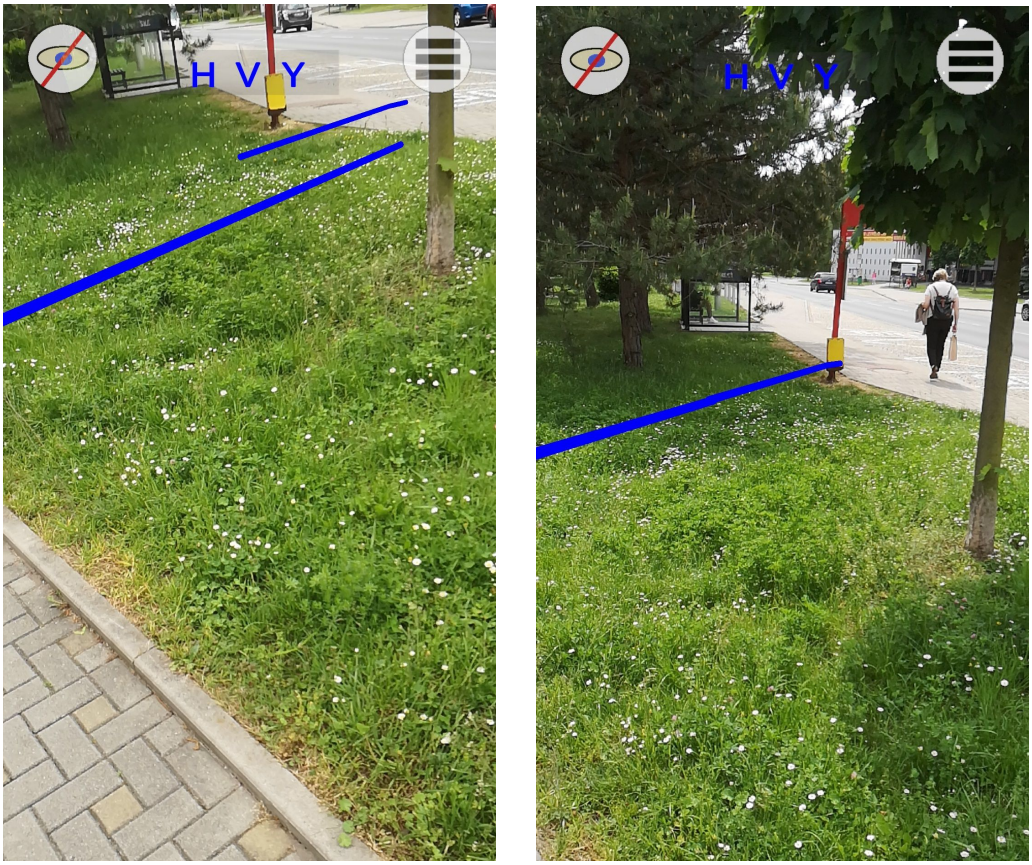


Figure 4.6. Visualization using Křovák(left) and UTM (right).

in all the generated meshes being slightly rotated in the positive direction. Further research led to a method for calculating the rotation and aligning it with respect to the north [81].

$$C = 0.008257\,Y + 2.373\,Y/X$$

Although this approach yielded improved results, they were not perfect. Therefore, I decided to explore an alternative coordinate system, the UTM coordinate system. While using UTM, I observed slightly different results, with the distances between points being slightly different compared to the Křovák's coordinate system. The difference was approximately 0.3 millimeters, which I deemed negligible. A comparison between the UTM and Křovák's coordinate systems, both with and without correction, is presented in Figure 4.7.

I also experimented with where the utility lines are anchored. At first, they were anchored at one of the points where they entered the grid cell. However, later that was changed to the middle point between the first and last points of the utility line inside the cell instead. This small change led to a much better alignment of the lines at the cells' boundaries.

4.7 Geospatial API and accuracy

To make the visualization accurate, the Geospatial API is used. First, the application checks if the device supports the API and if the user gave the application permission to

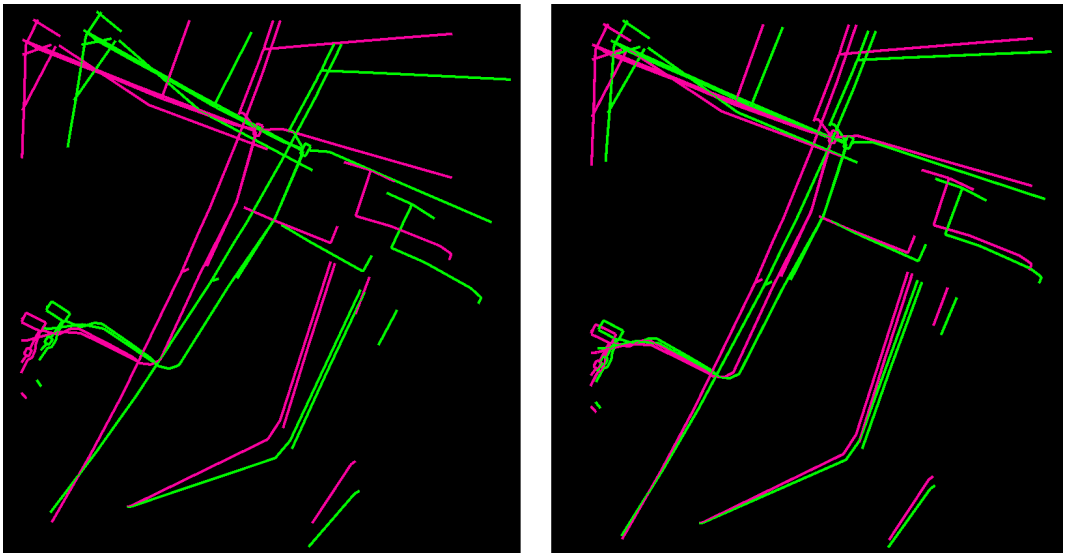


Figure 4.7. Comparison of UTM(pink) and Křovák(green) with(left) and without(right) angle correction.

use the camera and location. With permission received, the application starts tracking the device's position. The Geospatial API offers an option to check the horizontal, vertical, and yaw accuracy of the device. The accuracy is a number that describes the radius of the 68th percentile confidence level around the estimated value[82]. For example, if the estimated altitude is 150 meters and the vertical accuracy is 10, then there is a 68% chance that the true altitude is actually within 10 meters of the real altitude. It is essential to keep track of these values, and in case they become too high, the application informs the user that the visualization may be inaccurate. I decided to have two sets of values: one for the initialization, which requires higher accuracy, and slightly lower values for the rest of the application runtime. These values (see Table4.1) are set in a way that is possible to reach them, but also to keep the visualization quite accurate.

	Vertical	Horizontal	Yaw
Initialization	1	1	1.5
Run-time	1.5	1.5	3

Table 4.1. The required precisions.

Once the user finishes the calibration, that is, achieving the required accuracy values, the application starts visualizing the utility lines.

Initially, I wanted to use the terrain anchors that would position the objects at the appropriate height. However, they turned out to be unusable because they used the altitude above sea level. Unfortunately, the Geospatail API provides the altitude as a value above the WGS 84 ellipsoid. It is stated on the Geospatial API page [82] that the altitudes are "ruffly" the same. However, the difference in Prague is around forty-six meters, which is not even close to "ruffly" the same. So, I have decided to take a different approach.

Instead, the application takes the device's current altitude and subtracts a given value from it and visualizes the utility lines at that height using the WGS84 anchors. The user has the ability to set the value of how far under them the lines will be. In

addition to that, there is also a threshold value that checks the current altitude against the altitude from the time when the lines were anchored. If the difference between these two altitudes goes over the threshold, the lines are re-anchored to keep their distance from the user. This prevents the lines from sinking too far or from getting too close to the user. The threshold is also adjustable by the user. The effectiveness of the threshold can be seen in Figure 4.8.

Figure 4.8. Comparison of the result with(right) and without(left) the threshold.

4.8 Customization

There are many different types of utility lines in the data set, so I created a system to group some of the types together to limit customization in favor of the clarity of the application. The entire document with all types can be found on the Prague geoportal[83]. I take the code that represents the type of utility line and use it to create the categories. The first number of the code is used to obtain the general type of utility line. The next four numbers decide the subtype. The final number is ignored to avoid having too many categories. The user can costumize the categories. They can change their color or visibility. Right now, the user can also change the size of the utility lines for all of the lines at once, but it would be a great addition to the customization if this option is available for each of the categories. An example of current customization screens can be seen in Figure 4.9.

4.9 Utility line information

The user should be able to access the information of any of the utility lines. So, when they click on the lines, a pop-up box appears. This is done by a simple ray caster, which shoots a ray from the camera into the scene when the user taps it and retrieves

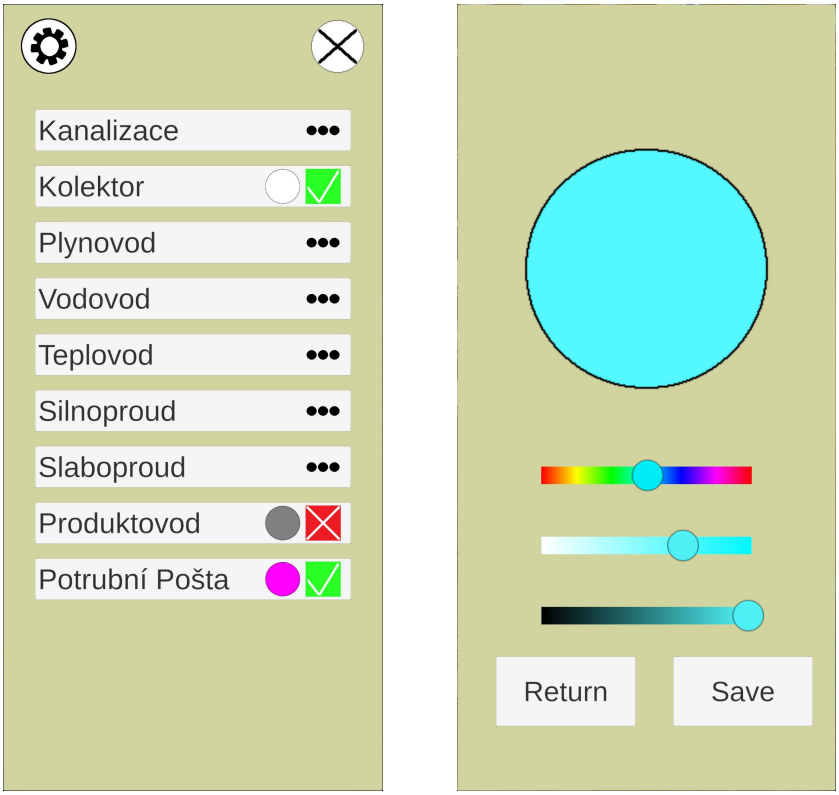


Figure 4.9. Menu and Color change screen.

the first hit object. Based on the user tests, which will be described later, I found that users would appreciate to know how deep underground a line is. The problem was that the application could not use the altitude of the device to calculate it because of the reasons described above. Therefore, instead, the Elevation API[84] is used. This API can be used to obtain the elevation of the location at which the user is currently at. To get the right height, the exact point on which the user tapped must be found. This is done by taking the raycast hit position, projecting it into the lines' local coordinate system, and then calculating the closest point on the line to the hitpoint. The point is then used to calculate the height by interpolation between the endpoints. The height is then subtracted from the elevation, and thus the value of how far below the ground the line should be is calculated.

Currently, the information about the pipe can be displayed in two modes. One shows the user all of the information (or just the depth) about the line in a box above it. This box has a set size and only rotates to face the camera. This can make the box unreadable if the user is too close or too far from it. So, it is possible to tap on the box. By tapping the box, the information will appear as a new UI element. When the user wants to close the interface, they can tap on it and it will disappear.

4.10 Occlusion and lighting

When it comes to occlusion, I used the features provided by the ARCore. The ARCore offers occlusion culling with three different modes, which vary in terms of performance and quality. I included all options in the application, and the user can switch between them in the settings. However, the user can also choose not to use occlusion culling

at all. Although it might make the visualization look less realistic, it allows the user to see the lines going through places that would otherwise be invisible. At the time of project development, the occlusion view distance had been limited to only 8.191 meters. That means that anything further than 8 meters is invisible. This is quite a contrast to the twenty-meter view distance that the mode without occlusion culling offers. On 10 May 2023 Google announced that the maximum range has increased to 65.535 meters; however, this has not been added to the project yet. It is also important to note that the occlusion culling is not perfect even when set to the `best` mode, especially when the line is running through long grass.

When it comes to lighting, I also used the features provided by ARCore. There are many different options when it comes to lighting estimation. I could choose to track ambient intensity, ambient color, ambient spherical harmonics, main light direction, or main light intensity.

I tried all the options, but the results were suboptimal. The main testing device was not able to track the direction and intensity of the main light very often, and when it did, the light intensity was so low that everything turned very dark. Due to that and the lack of time, I decided to make the application support only an ambient-intensity adjustment, where the ambient-light intensity is set to correspond to the estimated value.

Similarly to occlusion culling, users have the option to decide whether they want to use the ambient light intensity or if they want to use the default value.

4.11 Other encountered problems

Another problem that I encountered during the development was the fact that Unity works with floating-point numbers, which is quite limiting. Especially when I was working with a geographic coordinate system, which required high precision. For that reason, I limited the usage of floating-point numbers to a minimum. The only time the application uses floating-numbers is after the coordinates are projected into a planar system and are prepared to for the mesh creation.

Chapter 5
Testing

In this chapter, we will talk about the testing of the application. First, we will talk about the application testing. I will talk about its accuracy, performance, and usage of internet data. The second part will be about iterative user testing. We will discuss what I tested and how I incorporated the testing results into the development of the application.

5.1 Application functions testing

Accuracy is the most important factor of this application. If the accuracy is high enough, it could make the application usable for professional purposes, while if the accuracy is too low, it would be completely useless. I tested the accuracy in two different ways. The first test was to determine whether the visualization is in the correct place according to the position of the device. If the mobile thinks that it is somewhere where the utility line should be, then I can test if the line is actually there. The second test tries to evaluate whether the position of the device is accurate. That is, whether the mobile is where it thinks it is.

5.1.1 Visualization accuracy

To check whether the visualization is where it is supposed to be, I created a logging tool, where on a button click the current position of the device was saved. I then went into the Prague streets and followed some of the utility lines, logging my position on the way. After the logging was done, I took the data and used Qgis[85], a free open source desktop geographic information system application, to visualize it. Then, I used its functions to calculate the distance of the measured positions from the real utility line. During the test, I had the device in my hands and it is likely that I was not perfectly aligned with the utility line, so some of the inaccuracy can be caused by that. This test was carried out three times for three different utility lines.

The results were great, especially when I take into account that the data gathering method was not precise. I never got further than 0.4 meters from the utility line. The highest measured distance was 0.37 meters and it was only twice that I got over 0.3 meters. One of the tests is visualized in Figure 5.1 and its values can be seen in Table 5.1.

5.1.2 Device accuracy

It is a bit more complicated to check how accurate the device position is without any way of telling what the real position should be. So, I decided to place the device in a fixed position using a tripod. I chose a place where the application was able to get through the initialization (calibration) phase without the necessity to move and look around. Each time the application loaded and passed the initialization, I logged the position and started the process again. In this way, I could compare the values between the individual measurements and see how far they are from each other. I estimated the

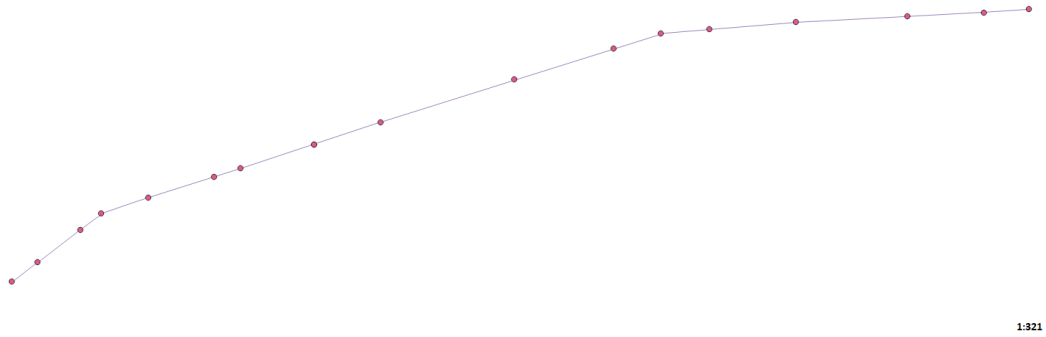


Figure 5.1. Points gather while compared to the line from the dataset.

Latitude	Longitude	Distance(m)
50.10511229	14.39057870	0.13509
50.10515986	14.39064109	0.05557
50.10523802	14.39074383	0.01015
50.10527621	14.39079419	0.08613
50.10531504	14.39090841	0.01564
50.10536511	14.39106702	0.03458
50.10538532	14.39113026	0.06857
50.10544284	14.39130773	0.04833
50.10544316	14.39130814	0.02993
50.10549559	14.39146876	0.10597
50.10559869	14.39179031	0.14475
50.10567308	14.39203060	0.06239
50.10570929	14.39214384	0.14363
50.10571979	14.39226171	0.04748
50.10573689	14.39247113	0.03902
50.10575136	14.39274016	0.05844
50.10576036	14.39292475	0.06794
50.10576829	14.39303307	0.07408

Table 5.1. Measured data, the distance is from the closest point on the utility line.

position as the average of all the logged values and then calculated the distance from the estimated position. I repeated this test three times in different places and evaluated the results. Surprisingly, the longest distance from the estimated position was 0.41 meters, which is much better than what I expected. The altitude differences were also not that large. The difference between the lowest and highest altitude measured at the same place was 0.56. Finally, the yaw angle was always within one degree. The visualization of one of the tests can be seen in Figure 5.3 and the corresponding values can be seen in Table 5.2.

The second way to test the accuracy of the application is to find a digging site, or places with utility line markings, and see if the utility lines visualize where they are supposed to be. I managed to find several such places and tested the application there. Screenshots of the application from these locations can be seen in Figure 5.4.

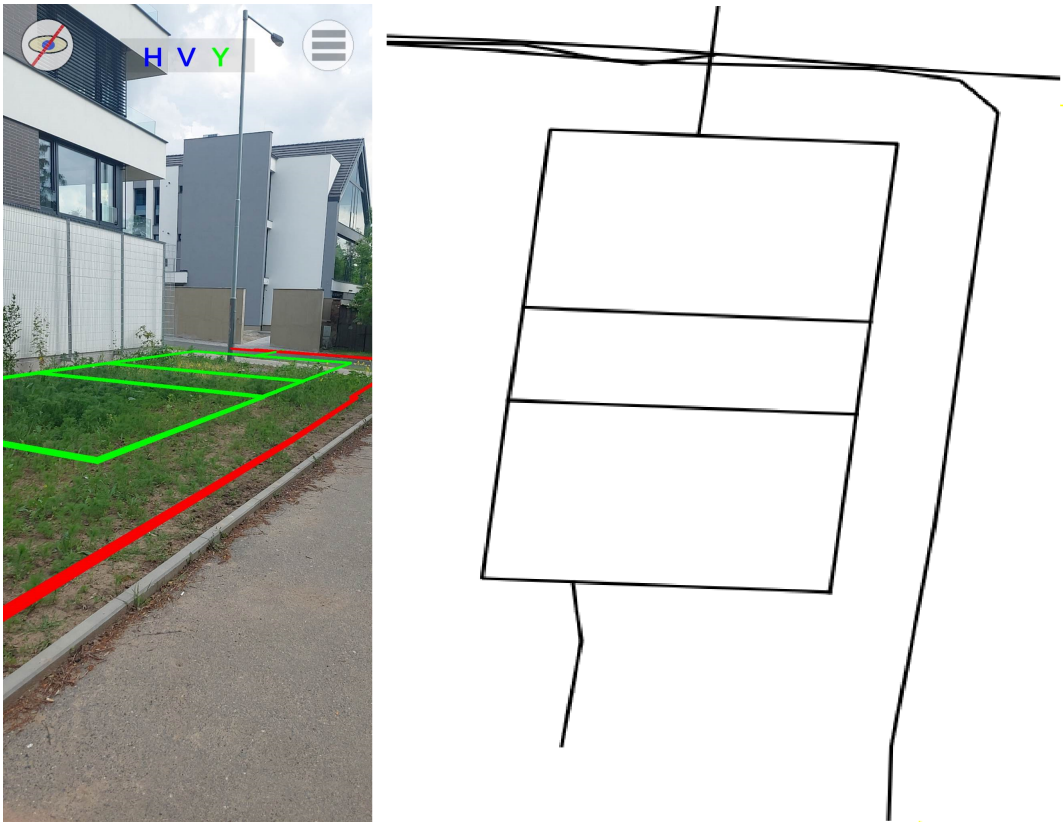


Figure 5.2. A comparison of the visualization with the visualization in QGis.

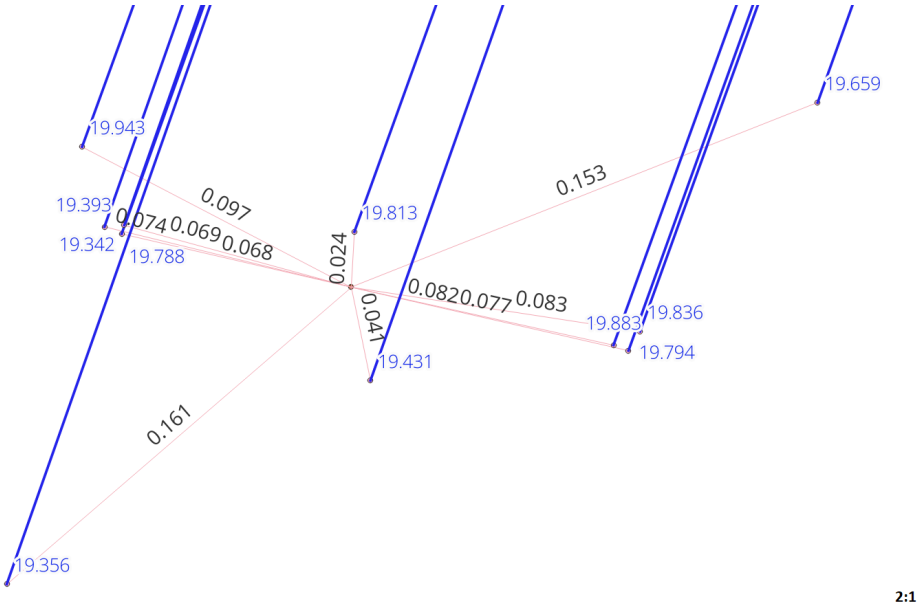


Figure 5.3. All of the positions measurements with their distance from their average value. With the blue lines and numbers that describe the yaw.

5.1.3 Coverage by Geospatial API VPS

It is great when the Geospatial API can be used to accurately pinpoint devices' position; however, it also binds the application to the places where the VPS is available. The VPS

Latitude	Longitude	Altitude	Yaw
50.16096486	14.74464028	231.796960	19.7880
50.16096537	14.74464300	231.747235	19.6590
50.16096440	14.74464226	231.772646	19.7939
50.16096448	14.74464230	231.763973	19.8355
50.16096443	14.74464220	231.754619	19.8829
50.16096487	14.74464119	231.838557	19.8125
50.16096520	14.74464012	231.863644	19.9434
50.16096489	14.74464029	231.905647	19.3416
50.16096489	14.74464021	231.905227	19.3925
50.16096429	14.74464125	231.866807	19.4305
50.16096349	14.74463983	231.839623	19.3556

Table 5.2. The data measured for the device accuracy test, depicted on Figure5.3.

uses the data from Google Street View, and one might expect it to work everywhere, where the streets are mapped. However, that is not the case. There are places, whole streets, which are possible to view in Google Street View, but they are not covered by VPS. I have not found many places where the VPS would not work in Prague(other than those described in the next paragraph), but that might be just because most of the tests were done in the city center. However, I managed to find many such places, in Čelákovice, a smaller city close to Prague. A map of these places can be seen in Figure 5.5.

I was unable to perform the calibration in places without any buildings nearby. It seems like the VPS is not using vegetation to pinpoint the users' location. So that limits the usage of the application even more. This also holds true for places where trees cover the buildings. If there is no building visible through the tree, the VPS will not work.

Lastly, it is important to note that the lighting matters. It is much easier to reach and maintain the necessary accuracy on sunny days and then on cloudy days.

5.1.4 Internet data usage

When it comes to using Internet data, the only part that the application can control is the communication with the server. The data used for communication with the server and data retrieval depend on the user's position and their movement. The more utility lines that need to be visualized, the higher the data usage. However, it is important to note that the amount of bytes sent and received is not that high. Each utility line costs 325 bytes for the data from the.dbf file and 244 bytes (8 bytes for each coordinate) for each point on the line. It is harder to estimate the data usage by Geospatial API, because I do not know what the communication with the API looks like. To monitor data usage, I used the built-in data monitoring on the device and checked how much data the application used over a period of time. I started with a one-minute interval and tested it in multiple different places. Then I set a longer time interval and tested it again. During the testing, I also tracked communication with the server so that I could subtract it from the tracked amount to find the geospatial API data usage. The measured data can be seen in Table 5.3. It seems like the initialization of the Geospatial API requires more internet data than during run-time.

Figure 5.4. Comparison of visualized utility lines with references from the real world.

5.1.5 The speed of visualization

I tested the visualization on multiple devices and in general did not encounter any problems with the speed of visualization. Frame rates per second (FPS) were consistently around 30 even when the occlusion mode was set to the best. However, on some devices, the application was unable to complete calibration, although they are mentioned in the official list of supported devices [86]. The application started and got to the calibration screen but was unable to calibrate. I then tried to run Pocket Garden, a project created to showcase how the Geospatial API works, on the device. However, I was not able to get through the calibration screen there either. This signals that the issue does not lie within the application but with the Geospatial API.

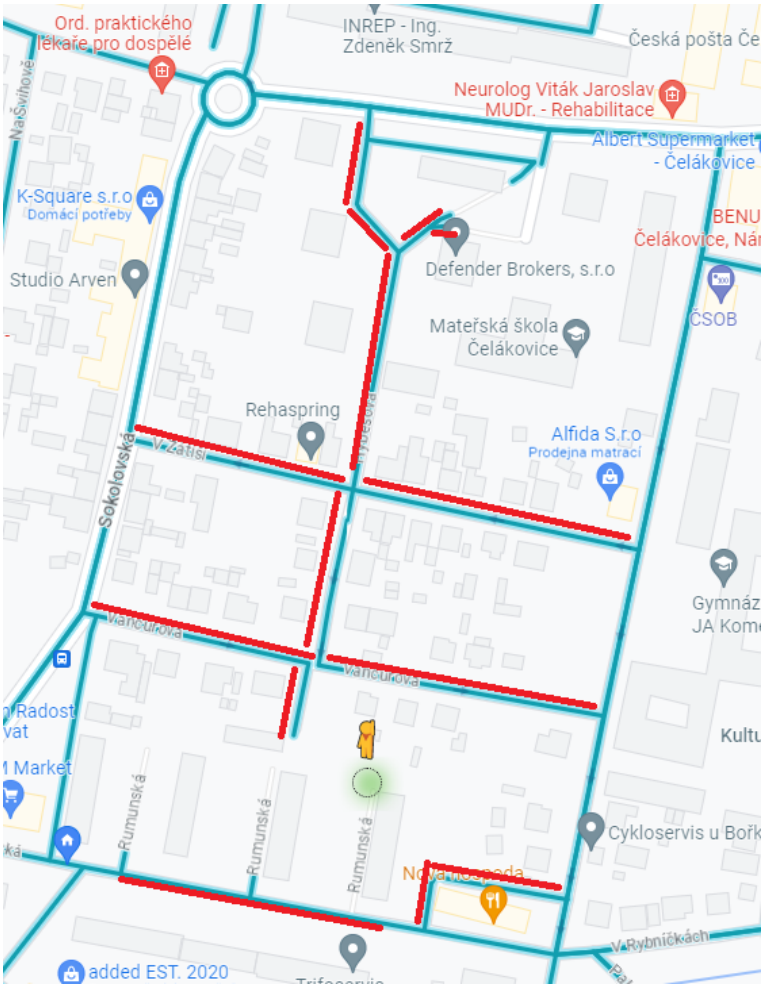


Figure 5.5. Blue indicates that Google Street View is available. Red indicates that VPS is not.

Time(min)	Geospatial API	Data retrieval	Overall
1	2.60	0.07	2.67
1	2.37	0.05	2.42
1	2.26	0.06	2.32
2	4.29	0.1	4.39
2	4.4	0.12	4.52
5	8.74	0.28	9.02
5	8.75	0.23	8.98
8	13.7	0.4	14.1
10	16.08	0.52	16.6

Table 5.3. Internet usage in megabytes based on the run time of the application.

5.2 User testing

An important part of application development is user testing. Users should feel comfortable using the application, and it should be as intuitive to use as possible. I decided to use the iterative testing approach, where I continue to test the application during

the whole development and adjust it based on the test results. In this section, I will go over the individual tests and how their results changed the final product.

5.2.1 The first test

I tested the application for the first time when I got the core functions and the basic UI done according to the design. It is important to note that at this time the application was not re-anchoring the utility lines when the height changed too much, so the utility lines sometimes just randomly flew into the distance or got too close to the user. I asked two different participants to perform these tasks:

- **Calibrate your position**

  The first task had to be the device calibration, as without it the application would not start. Both users were in a location with VPS coverage, so they had no problems with it.
- **Find a water line and identify what kind of line it is**

  I chose the water line because its color is set to blue and thus is easy to identify. The first participant managed to accomplish it very quickly because the closest line to them was a water line. The second participant had to walk a distance, but they also managed to find a water line without any problems.
- **Hide the water line**

  Both participants immediately entered the menu. However, when they got to the water line submenu, they did not know which of the subtypes the line before them belonged to. One participant decided to go back from the menu and find out by reading the information about the line again. The other decided to hide all of the water lines instead.
- **Make the water line green**

  This task was quite similar to the previous one. The second participant once again went through all of the subtypes and changed all of their colors. The first participant changed the color of only the one.

At this stage, these were all the functionalities of the application, so there was nothing more to test. Both participants managed to complete all tasks, although the second approach was a bit unorthodox. I asked both participants what they would like to improve and if there was anything they did not like. Both of them pointed out that the indicator of whether all, none, or some of the subtypes are hidden is confusing, and it would be much better to put there some kind of indicator that leads somewhere else. The second participant then pointed out that it would be better to see the color in the menu rather than after entering the color changing screen. In his words, “It was annoying to check whether I had already changed the color or not.”

The other thing both of them brought up was the “annoying” Lost tracking message that covered the whole screen. Especially when they had no idea that their accuracy was getting worse.

I agreed with all of their observations and thus redesigned parts of the application. I removed the type indicator and instead replaced it with three dots, which indicate that another menu is coming. IWe changed the color button to show the current color instead of the rainbow. I also implemented an indicator that shows the user whether their accuracy is getting closer to the minimal limit, and finally, I also changed the “annoying” lost tracking pop-up to be less intrusive. The changes of the menu can be seen in Figure 5.6.

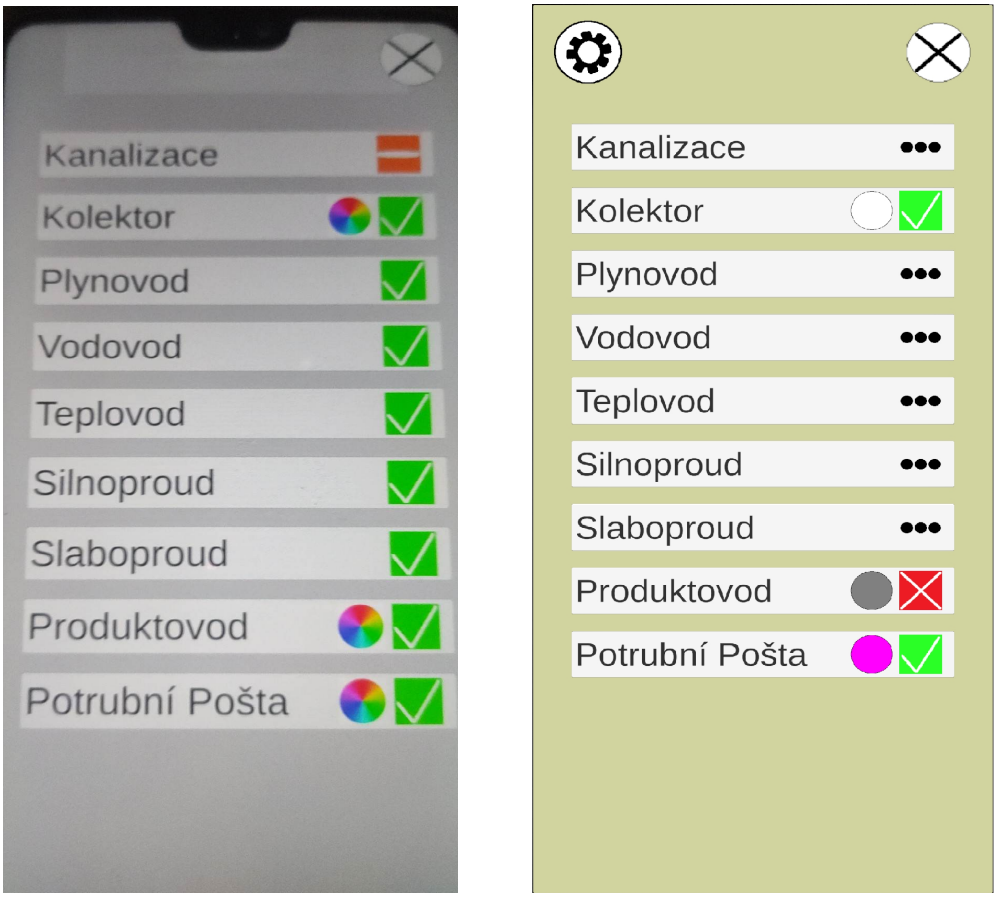


Figure 5.6. Showcase of how the menu changed after the first user testing. The old menu is on the left.

5.2.2 The second test

I changed the application based on the results of the first test. On top of that, I also added the altitude correction, so now the utility lines were more stable. I tested the application with two different participants. I kept the scenario the same as in the first test but just asked the users to find a gas line instead of a water line.

Both participants got through the tasks without any issues. Although, they did not encounter any problems during the testing. One of them mentioned that it was great to know at what height the utility line is, but it would be much more useful to see how deep it is beneath the ground. Also, if such an option was available, it would be great to be able to see information about multiple utility lines at once.

This led me to make use of the elevation API and calculate the depth of the utility lines. Furthermore, I added the pop-up signs that appear on the utility line when the user clicks on them. I was aware that there might be situations where the pop-up sign would be unreadable, and thus I kept the UI that was used to showcase the information about the utility line, and when the user clicks on the pop-up, it opens the UI. The pop-up signs can be seen in Figure 5.7.

5.2.3 The third test

The third test was performed when I added occlusion culling and ambient lightning estimation to the application. I also added the option to adjust the height that is being used for the utility lines' offset and the threshold for their re-anchoring. These settings

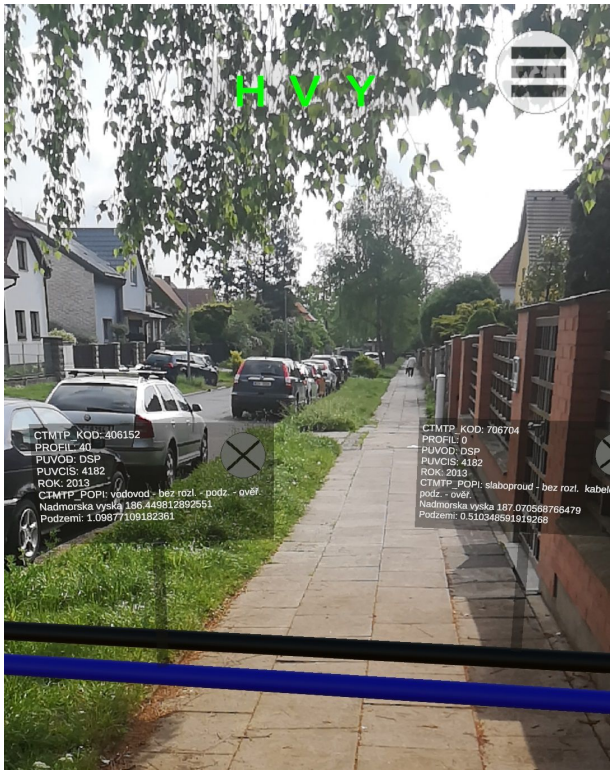


Figure 5.7. A showcase of the pop-up sign

are important for occlusion culling to work. Before the user would not be able to recognize, if the utility lines are visualized slightly below the ground, but the occlusion culling makes it very visible. With the wrong setting, the utility lines could become completely hidden by the ground. These two settings are quite hard to understand, so I told the participants about them before the test. The user can find these details in the manual.

The third test consisted of four new tasks in addition to the previous tests, and three participants tried to complete them.

- **Turn on occlusion culling**

  The users had to go to the settings menu and turn on occlusion culling. I left it up to them to decide which of the modes they want to try out. All the participants were able to do it without any problem.
- **Adjust the settings to suit your needs**

  One of the users did not have to adjust the settings at all. They were able to see the utility lines above the ground, even with the occlusion culling on. The other two had to change the height at which the lines were anchored, but thanks to my explanation before the test started, they did it without any problems.
- **Turn on the ambient lighting estimation**

  This task did not cause any problems either, because it is as simple as going to the setting menu and switching from one mode to another.
- **Change the size of the utility lines to 0.05**

  In this version, I also included the possibility to change the size of the utility lines. The task itself is again very easy, because it can be done by using a slider in the settings menu.

I wanted to find out whether the occlusion culling and the ambient light estimation enhance the realism of the application.

Occlusion culling received mixed reviews. On the one hand, the participant thought it was great to see the utility line hidden beneath a car; on the other hand, it was hard to keep track of a line that led under many of them. Also, when a line passed through tall grass, the occlusion culling made it look unnatural. On top of that, all of them complained about the shorter vision range. One of the participants suggested making it easier to turn on and off the occlusion because going to the settings menu each time is a chore.

The ambient light estimation was not received well at all. It made the colors less distinct, especially during one of the tests when it was cloudy. All of the participants quickly reverted to the default mode, where the colors are bright and easily distinguished. I would like to quote one of the participants "I have set up so many different colors, and with this mode on, they all look the same."

One user also suggested making a button that would close all of the pop-ups at once.

In general, the participants did not mind that the utility lines are not realistic. They preferred to see them through the obstacles so they could keep a better track of them, and they also preferred to keep the colors bright.

I kept the option to turn these features on in the application. I added the button to turn the occlusion culling on and off to the main screen and I also added a button that appears when there are some pop-ups in the world that get rid of them all.

5.2.4 The fourth test

The last test was performed with only one participant. This participant was testing the application during the first test, so it was interesting to see what they would think about the changes. I did not change much between this and the third test; I only added two buttons, one to turn the occlusion on and off and the second to remove all the pop-up signs.

The participant was surprised at how different the application now felt. Especially when the utility lines did not fly randomly to the distance. They got through all the tasks without an issue. They suggested making it possible for the pop-up signs to show only the depth of the lines. This would make it easier to see all the different depths when there are a lot of utility lines in the same place. They also suggested creating some mode where the lines would not be planar and thus it would be even easier to see.

I added the option to make the signs show only depth, which can be seen in Figure5.8, but I did not create a 3D mode. I agree that it would be great, but I did not have enough time to make it happen.

5.3 Summary

Application function testing is fundamental to the overall functionality and efficiency of the app. The focus of the tests was mainly on accuracy, performance, and internet data usage. In terms of accuracy, I evaluated both the visualization accuracy, how correctly the app displayed the position of utility lines based on the device's location, and the device accuracy, which evaluated if the device position corresponds to the real world. Performance testing looked into the speed of visualization across different devices, while internet data usage monitoring evaluated the app's communication efficiency with the server and the Geospatial API. I also looked at the coverage of the Visual Positioning System (VPS) to understand its impact on the application's accuracy and utility.

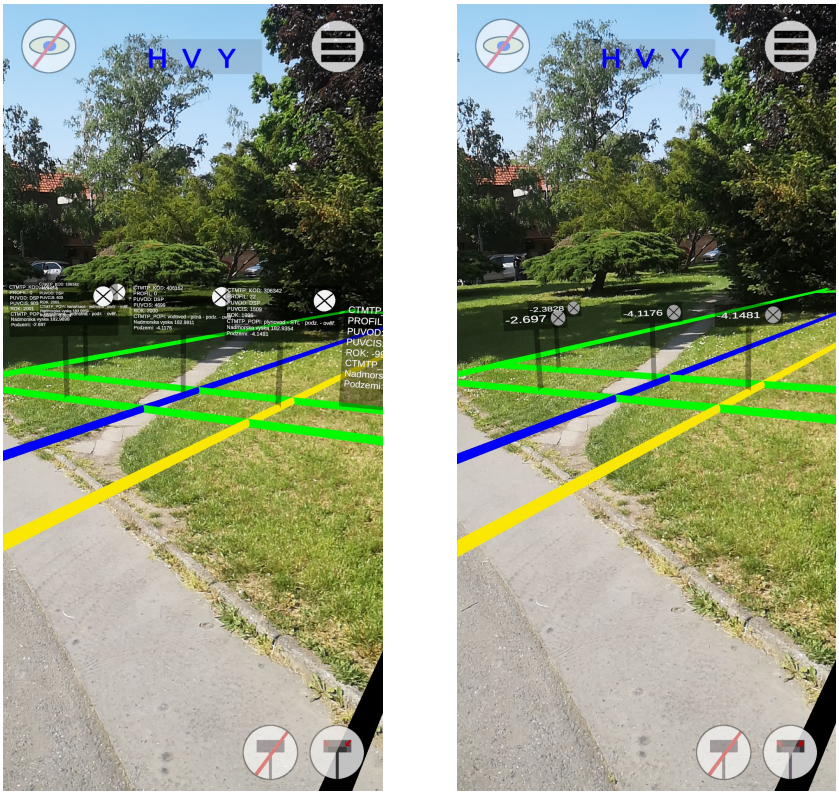


Figure 5.8. Comparision of the usage of big and small signs.

The iterative testing and development approach helped me adapt the application to user feedback, allowing me to get a more in-depth understanding of what users want from the application.

In terms of the application's core functionalities, users were generally pleased with the ability to identify, hide, and color-code different types of utility lines.

One notable advancement I made in response to user feedback was the implementation of the elevation API to calculate the depth of utility lines. This addition, along with the clickable pop-up signs, significantly improved the application's functionality and usability.

Despite some challenges and complications, such as the limitations of occlusion culling and the inadequacy of ambient light estimation, user testing was ultimately beneficial for the development of the application. It allowed me to continually refine and improve the application throughout its development cycle, resulting in a product that was better suited to the needs and preferences of its users.

Chapter 6
Conclusion

My task was to use some of the geospatial data available on Prague's Geoportal and visualize it with an AR application using the Geospatial API. I analyzed the data and their meaning, looked at the capabilities of ARCore and the Geospatial API, and searched for AR applications that either work with the geospatial data or use the API. Then I chose which geospatial I wanted to visualize and designed an application for it. With the design ready, I implemented the core functions before performing several rounds of user testing. I integrated the results of the tests and implemented new functions in between the rounds to obtain user opinions on the changes. I also tested the application's accuracy and performance and limitations.

The application I created is able to visualize the utility lines in Prague. It allows users to find out what utility lines are in their vicinity and how deep underground they are. It gives users the ability to color code the lines and choose which of them should be visualized. It includes the options to use occlusion culling and ambient light estimation to make the visualization more realistic.

Based on the test results, I do not think that this application is ready to be used by professionals. It can be used to give them a general idea of what utility lines are going through the area, but it is not able to precisely locate them. I think the 0.3-meter inaccuracy is too much and I cannot guarantee that it is not possible that even bigger errors would occur. I could try to set the accuracy limits a bit higher, and thus get a bit more accurate, but then the application might be hard to use.

When it comes to the application, there are still many ways it could be improved. It would be better to visualize the depth of the utility lines in some way, rather than using the information boxes to convey the information. Also, it would be interesting to look into the lighting a bit more. The UI could use designer to redesign it because right now it is very simple and not very appealing.

Even though I was not able to get the accuracy high enough to accurately visualize utility lines, I still think that the Geospatial API is a powerful tool and it provides the means to reach position accuracies in the city which were not possible before without an external device. It might not work everywhere and it might not work perfectly, but it opens the door to many different interesting applications that would not work before.

References

[1] *Geoportál hlavního města Prahy, Data.* https://www.geoportalpraha.cz/cs/data/otevrena-data/seznam. Accessed on 2023-01-08.

[2] *IPR.* http://en.iprpraha.cz/. Accessed on 2023-04-08.

[3] *Geoportál hlavního města Prahy.*

[4] *IPR Prague.* http://en.iprpraha.cz/clanek/1358/ipr-prague. Accessed on 2023-04-08.

[5] *Shapefile.* https://www.precisely.com/glossary/shapefile. Accessed on 2023-03-14.

[6] ESRI. *ESRI Shapefile Technical Description.* 1998. https://www.esri.com/content/dam/esrisites/sitecore-archive/Files/Pdfs/library/whitepapers/pdfs/shapefile.pdf. Accessed on 2023-03-19.

[7] *ESRI Shapefile.* https://www.loc.gov/preservation/digital/formats/fdd/fdd000280.shtml. Accessed on 2023-03-19.

[8] *GeoJson.* https://datatracker.ietf.org/doc/html/rfc7946. Accessed on 2023-04-08.

[9] *DXF files.* https://www.adobe.com/creativecloud/file-types/image/vector/dxf-file.html. Accessed on 2023-04-08.

[10] *CityGML.* https://www.ogc.org/standard/citygml. Accessed on 2023-04-08.

[11] *Introducing Coordinate Systems and Map Projections.* https://www.youtube.com/watch?v=PICwxT0fTHQ&ab_channel=EsriEvents. Accessed on 2023-04-10.

[12] *Geographic coordinate systems, datums.* https://desktop.arcgis.com/en/arcmap/latest/map/projections/datums.htm. Accessed on 2023-04-10.

[13] *Geographic coordinate system.* https://www.ibm.com/docs/en/informix-servers/12.10?topic=data-geographic-coordinate-system. Accessed on 2023-03-12.

[14] *What are geographic coordinate systems?* https://desktop.arcgis.com/en/arcmap/latest/map/projections/about-geographic-coordinate-systems.htm. Accessed on 2023-04-10.

[15] *Graticular Network.* https://desktop.arcgis.com/en/arcmap/latest/map/projections/GUID-0921FD4E-B619-491B-92C2-38B70E231948-web.gif. Accessed on 2023-04-10.

[16] *What are projected coordinate systems?* https://desktop.arcgis.com/en/arcmap/latest/map/projections/about-projected-coordinate-systems.htm. Accessed on 2023-04-10.

[17] *Projected coordinate systems.* https://www.ibm.com/docs/en/db2-warehouse?topic=SSCJDQ/com.ibm.db2.luw.spatial.topics.doc/doc/csb3022b.htm. Accessed on 2023-03-12.

[18] *GIS Concepts - projections image.* http://gisedu.colostate.edu/webcontent/nr505/2012_Projects/Team6/images/GIS_concepts/Figure2_proj.PNG.

[19] *WORLD GEODETIC SYSTEM 1984.* https://earth-info.nga.mil/index.php?dir=wgs84&action=wgs84. Accessed on 2023-03-12.

[20] Paul Bolstad. *GIS fundamentals: a first text on geographic information systems. 3rd ed.*. 2008.

[21] John P. Snyder. *Map Projections: A Working Manual*. 2012.

[22] James R. Smith. *Introduction to geodesy : the history and concepts of modern geodesy*. 1997.

[23] *UTM Image.* https://www.researchgate.net/profile/Divya-Rk-2/publication/342330834/figure/fig3/AS:904454035496961@1592650054738/Universal-Transverse-Mercator-UTM-coordinate-system-is-a-standard-set-of-map.jpg. Accessed on 2023-04-10.

[24] B. Veverka. *Krovák's projection and its use for the Czech Republic and the Slovak Republic.*. https://citeseerx.ist.psu.edu/document?repid=rep1type=pdf-doi=c97d46ee3f78af13b81935ccfa8f6a2ce1575e03: 2004. Accessed on 2023-04-10.

[25] *Křovákovo zobrazení.* https://czwiki.cz/Lexikon/K%C5%99ov%C3%A1kovo_zobrazen%C3%AD. Accessed on 2023-04-10.

[26] *Křovák's projection image.* https://player.slideplayer.cz/11/2954947/data/images/img12.png. Accessed on 2023-04-10.

[27] *GPS.* https://www.gps.gov/systems/gps/. Accessed on 2023-03-14.

[28] *About GLONASS.* https://glonass-iac.ru/en/about_glonass/. Accessed on 2023-04-06.

[29] *What is Galileo?* https://www.gsc-europa.eu/galileo/what-is-galileo. Accessed on 2023-04-06.

[30] *Beidou.* http://en.beidou.gov.cn/SYSTEMS/System/. Accessed on 2023-04-06.

[31] *Space Segment.* https://www.gps.gov/systems/gps/space/. Accessed on 2023-03-14.

[32] *How Do You Measure Your Location Using GPS?.* https://www.nist.gov/how-do-you-measure-it/how-do-you-measure-your-location-using-gps. Accessed on 2023-03-14.

[33] *Control Segment.*
https://www.gps.gov/systems/gps/control/. Accessed on 2023-03-14.

[34] *GPS places picture.*
https://www.gps.gov/systems/gps/control/map.png. Accessed on 2023-03-14.

[35] Lu Zhiping, Qu Yunying, and Qiao Shubo. *Geodesy : Introduction to geodetic datum and geodetic systems*. 2014.

[36] *Accuracy.*
https://www.gps.gov/systems/gps/performance/accuracy/. Accessed on 2023-03-14.

[37] *GPS equations pictures.*
https://qph.cf2.quoracdn.net/main-qimg-3a562785e28606779d96b4b044a03744, https://qph.cf2.quoracdn.net/main-qimg-8761f3247584eede4024dc4b373b3158. Accessed on 2023-03-14.

[38] Christopher Hegarty, and Elliott Kaplan. *Understanding GPS Principles and Applications, Second Edition*. 2005.

[39] *ARCore.*
https://developers.google.com/ar. Accessed on 2023-04-01.

[40] *ARCore, Fundamental Concepts.*
https://developers.google.com/ar/develop/fundamentals. Accessed on 2023-04-01.

[41] Ikeda S. Taketomi T., Uchiyama H.. *Visual SLAM algorithms: a survey from 2010 to 2016*. 2017.
https://doi.org/10.1186/s41074-017-0027-2. Accessed on 2023-03-17.

[42] Kaichang Di, Wenhui Wan, H. Zhao, Zhaoqin Liu, R. Wang, and F. Zhang. Progress and Applications of Visual SLAM. *Cehui Xuebao/Acta Geodaetica et Cartographica Sinica*. 2018, 47 770-779. DOI 10.11947/j.AGCS.2018.20170652.

[43] Weifeng Chen, Guangtao Shang, Aihong Ji, Chengjun Zhou, Xiyang Wang, Chonghui Xu, Zhenxiong Li, and Kai Hu. An Overview on Visual SLAM: From Tradition to Semantic. *Remote Sensing*. 2022, 14 (13),

[44] *ARCore, Environment.*
https://developers.google.com/ar/design/environment/definition. Accessed on 2023-04-01.

[45] *ARCore, Get the lighting right.*
https://developers.google.com/ar/develop/lighting-estimation. Accessed on 2023-04-01.

[46] *ARCore, Depth adds realism.*
https://developers.google.com/ar/develop/depth. Accessed on 2023-04-01.

[47] *ARCore, Hit-tests place virtual objects in the real world.*
https://developers.google.com/ar/develop/hit-test. Accessed on 2023-04-01.

[48] *Introducing Persistent Cloud Anchors from ARCore.*
https://www.youtube.com/watch?v=b4mgaIuCozk&ab_channel=GoogleAR%26VR. Accessed on 2023-04-12.

[49] *ARCore, Cloud Anchors allow different users to share AR experiences.*
https://developers.google.com/ar/develop/cloud-anchors. Accessed on 2023-03-17.

[50] *ARCore, Add dimension to images.* https://developers.google.com/ar/develop/augmented-images. Accessed on 2023-04-01.

[51] *ARCore Documentation for Geospatial API.* https://developers.google.com/ar/develop/geospatial. Accessed on 2023-03-17.

[52] *Geospatial Anchors.* https://developers.google.com/ar/develop/unity-arf/geospatial/anchors. Accessed on 2023-04-16.

[53] *VPS and the ARCore Geospatial API.* https://www.youtube.com/watch?v=pFn11hYZM2E&ab_channel=GoogleDevelopers. Accessed on 2023-03-17.

[54] *Unity.* https://unity.com/. Accessed on 2023-04-17.

[55] Borromeo Nicolas Alejandro. *Hands-On Unity 2021 Game Development.* 2021.

[56] Hocking Joseph. *Unity in Action, Third Edition.* 2022.

[57] *Oxford Learner's Dictionaries - Augmented Reality.* https://www.oxfordlearnersdictionaries.com/definition/english/augmented-reality. Accessed on 2023-04-17.

[58] *Practical augmented reality : a guide to the technologies, applications and human factors for AR and VR.*

[59] Ronald T. Azuma. *A Survey of Augmented Reality.* http://www.cs.unc.edu/~azuma/ARpresence.pdf. Accessed on 2023-04-17.

[60] *Augmented Reality In Healthcare.* https://medicalfuturist.com/augmented-reality-in-healthcare-will-be-revolutionary/. Accessed on 2023-04-12.

[61] Dieter Schmalstieg, and Tobias Hollerer. *Augmented Reality: Principles and Practice (Usability).* 2016. ISBN 978-0321883575.

[62] A. Montero, T. Zarraonandia, P. Diaz, and et al. *Designing and implementing interactive and realistic augmented reality experiences.* 2019. https://doi.org/10.1007/s10209-017-0584-2. Accessed on 2023-04-17.

[63] *ARCore, Realism.* https://developers.google.com/ar/design/content/realism. Accessed on 2023-04-13.

[64] *Realism in Augmented Reality.* https://www.interdigital.com/download/592efefd01a81b7b0b0007ed. Accessed on 2023-04-13.

[65] Francesco Osti, Gian Maria Santi, and Gianni Caligiana. Real Time Shadow Mapping for Augmented Reality Photorealistic Rendering. *Applied Sciences.* 2019, 9 (11),

[66] *Pocket Garden.* https://buck.co/work/google-gracie-pocket-garden. Accessed on 2023-04-17.

[67] *Geospatial Contest.* https://arcoregeospatialapi.devpost.com/project-gallery?page=1. Accessed on 2023-04-17.

[68] *Skinny Ape*.
https://skinnyape.gorillaz.com/. Accessed on 2023-04-17.

[69] *Skinny Ape video*.
https://www.youtube.com/watch?v=iFaKhtlBU7A&ab_channel=Gorillaz. Accessed on 2023-04-17.

[70] *Google Live View*.
https://support.google.com/maps/answer/9332056?hl=cs&co=GENIE.Platform%3DAndroid. Accessed on 2023-04-17.

[71] *Google Live View, Image*.
https://www.google.com/url?sa=i&url=https%3A%2F%2Fvrmag.cz%2Flive-view-google-maps-v-rozsirene-realite%2F&psig=AOvVaw1kCcxE6aYgWG7drnJfdozY&ust=1684747873550000&source=images&cd=vfe&ved=0CBEQjRxqFwoTCIDOoIWNhv8CFQAAAAAdAAAAABAE. Accessed on 2023-04-17.

[72] *Geospatial Creator Introduction*.
https://io.google/2023/program/b95fac83-6dfd-4138-928a-f2d60dde408d/. Accessed on 2023-05-20.

[73] *Geospatial Creator in Unity*.
https://developers.google.com/codelabs/arcore-unity-geospatial-creator?hl=zh-cn##2. Accessed on 2023-05-20.

[74] *AuGeo Application*.
https://play.google.com/store/apps/details?id=com.esri.augeo&hl=cs&gl=US.

[75] *AugView*.
https://www.augview.net/. Accessed on 2023-04-17.

[76] Trimble. *SiteVision*.
https://sitevision.trimble.com/. Accessed on 2023-04-17.

[77] *vGIS*.
https://www.vgis.io/. Accessed on 2023-04-17.

[78] dBase. *Data File Header Structure for the dBASE Version 7 Table File*.
https://www.dbase.com/Knowledgebase/INT/db7_file_fmt.htm. Accessed on 2023-03-19.

[79] *ARCore unity extensions*.
https://github.com/google-ar/arcore-unity-extensions.git. Accessed on 2023-03-17.

[80] Federico Casares. *Unity Plumber*.
https://github.com/federicocasares/unity-plumber.

[81] *Zobrazení užitá pro ČSR a ČR*.
http://old.gis.zcu.cz/studium/mk2/multimedialni_texty/index_soubory/hlavni_soubory/cechy.html##obr.%2014.8.

[82] *Geospatial Documentation for Geospatial Pose*.
https://developers.google.com/ar/reference/unity-arf/struct/Google/XR/ARCoreExtensions/GeospatialPose. Accessed on 2023-04-17.

[83] *Číselník typů prvků technické mapy*.
https://www.geoportalpraha.cz/assets/dokumenty/datove-sady/ciselnik_technicka_mapa.pdf.

[84] *Google Elevation API*.
https://developers.google.com/maps/documentation/elevation/start.

[85] *QGis*.
https://qgis.org/en/site/.

[86] *ARCore, supported devices*.
https://developers.google.com/ar/devices##google_play.

Appendix A
Glossary

API	Application Programming Interface
AR	Augmented Reality
FPS	Frames Per Second
GCS	Geographic Coordinate System
GIS	Geographic Information System
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
IMU	Inertial Measurement Unit
IPR	The Prague Institute of Planning and Development
QR	Quick Response
UI	User Interface
UTM	Universal Transverse Mercator
VPS	Visual Positioning System
vSLAM	Visual Simultaneous Localization and Mapping
WGS84	World Geodetic System established in 1984
XR	Extended Reality

Appendix B
Custom binary file for quick data retrieval from a grid

The file starts with a header that is 72 bytes long and consists of these values:

- **double meterLatitude** - this is an approximated value of how much latitude degrees correspond to one meter.
- **double meterLongitude** - this is an approximated value of how many longitude degrees correspond to one meter.
- **int zoneSizeInMeters** - the first division of the whole area; this says how many meters should one zone be.
- **int areaSizeInMeters** - the second division of the whole area; this says how many meters should one area be.
- **double fromLongitude** - the minimum longitude of the whole area.
- **double toLongitude** - the maximum longitude of the entire area.
- **double fromLatitude** - the minimum latitude of the entire area.
- **double toLatitude** - the maximum latitude of the entire area.
- **int zoneAmountLon** - number of longitude-wise zones.
- **int zoneAmountLat** - number of latitude-wise zones.
- **int areaAmountLon** - number of longitude-wise areas.
- **int areaAmountLat** - number of latitude-wise areas.

There are **zoneAmountLon** * **zoneAmountLat** elements following the header. Each element has 5 bytes representing these values:

- **bool somethingIsInZone** - tells if there is any utility line going through this zone
- **int offsetZ** - the offset for the information about area

To find the right element we can count the offset like this:

```
offset = (yourLonZoneIndex * zoneAmountLat + yourLatZoneIndex) * 5 + 72
```

where 5 is the size of one element and 72 is size of the header.

After all of the zone elements, there are blocks of **areaAmountLon** * **areaAmountLat** for each zone that had **somethingIsInZone** set to true.

Once again there are 5 bytes long elements this time corresponding to the areas:

- **bool somethingIsInArea** - tells if there is any utility line going through this zone.
- **int offsetA** - the offset for the information about the area.

To find the right area element we can count the offset like this:

```
offset=(yourLonAreaIndex * areaAmountLat + yourLatAreaIndex)*5 + offsetZ
```

where the 5 is the size of one element.

If **somethingIsInArea** was true, we can find the data about the cell at **offsetA** The data have this form:

- **int length** - number of ids that follows.
- **int id** - an id of the utility line that passes through this area. (There is a **length** amount of them).

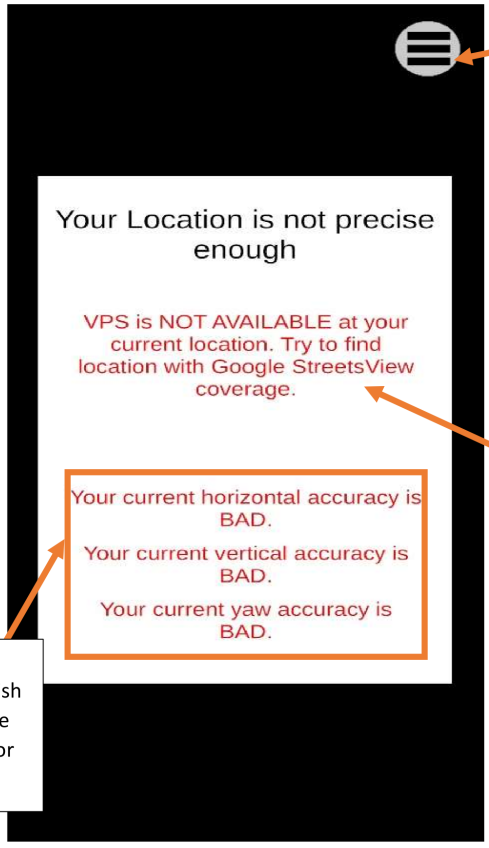
In case we do not know what our zone and area indexes are, we can get them like this:

```
yourLonZoneIndex = Math.Floor((YOURLONGITUDE - fromLongitude) /
    meterLongitude * zoneSizeInMeters)

yourLatZoneIndex = Math.Floor((YOURLATITUDE - fromLatitude) /
    meterLatitude * zoneSizeInMeters)

zoneFromLong = fromLongitude + (LongitudeIndex) *
    (meterLongitude * zoneSizeInMeters)

zoneFromLat = fromLatitude + (LatitudeIndex) *
    (meterLatitude * zoneSizeInMeters)

yourLonAreaIndex = Math.Floor((YOURLONGITUDE - zoneFromLong) /
    meterLongitude * areaSizeInMeters)

yourLatAreaIndex = Math.Floor((YOURLATITUDE - zoneFromLat) /
    meterLatitude * areSizeInMeters)
```

Appendix C
Manual

When you open the app, you are asked to let the application use your camera and get access to your location. The application needs these permissions to run. PipeIT is an AR application; an AR doesn't work without a camera. PipeIT is also a location-based application; without your location, it won't work. Furthermore, it should be noted that this app requires an internet connection and consumes data during its runtime as it downloads data about the pipes and communicates with the VSP server.

Here is a walkthrough of the app where the functionalities are explained. This is the calibration screen. It tells you what your current position accuracy is. To start the AR experience, you must get all three accuracies to green. If you believe that the VPS should be available and it says it isn't, try to restart the app, as sometimes your GPS position is so off that it thinks the VPS isn't available.

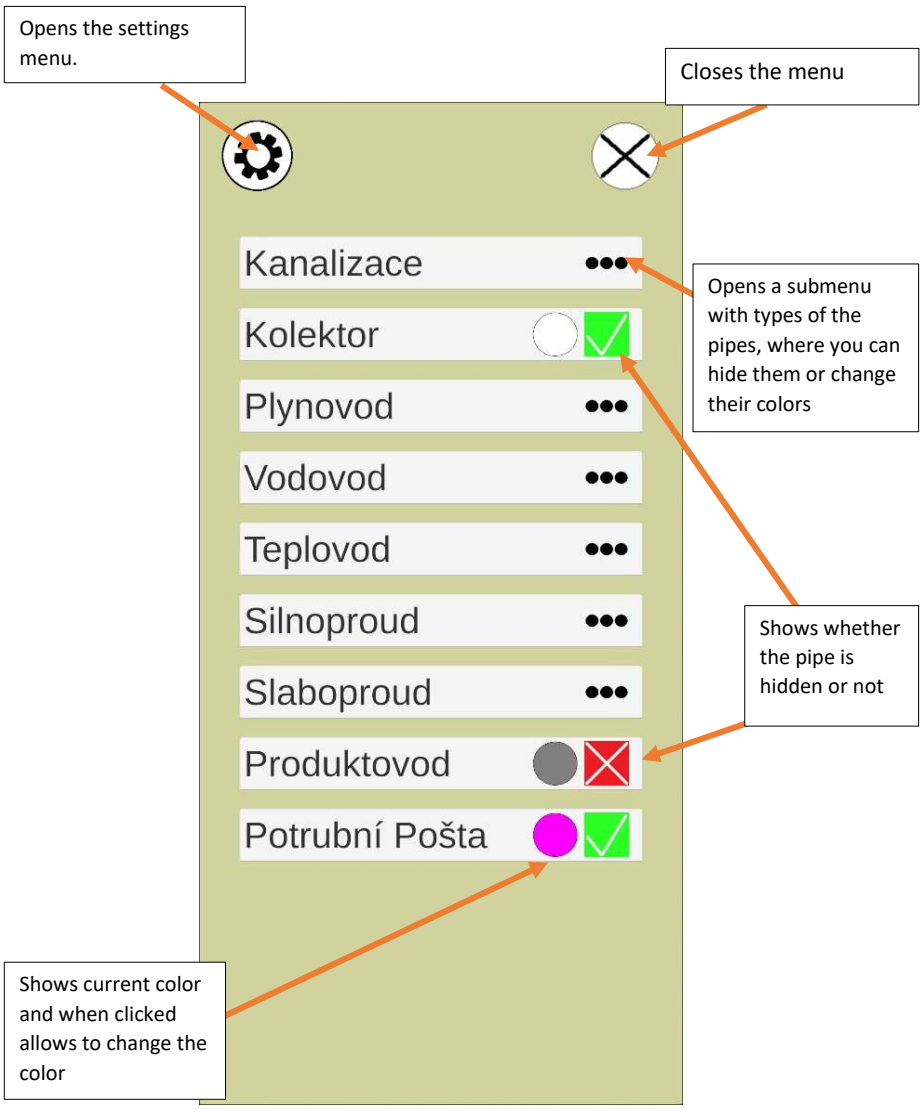


A menu button

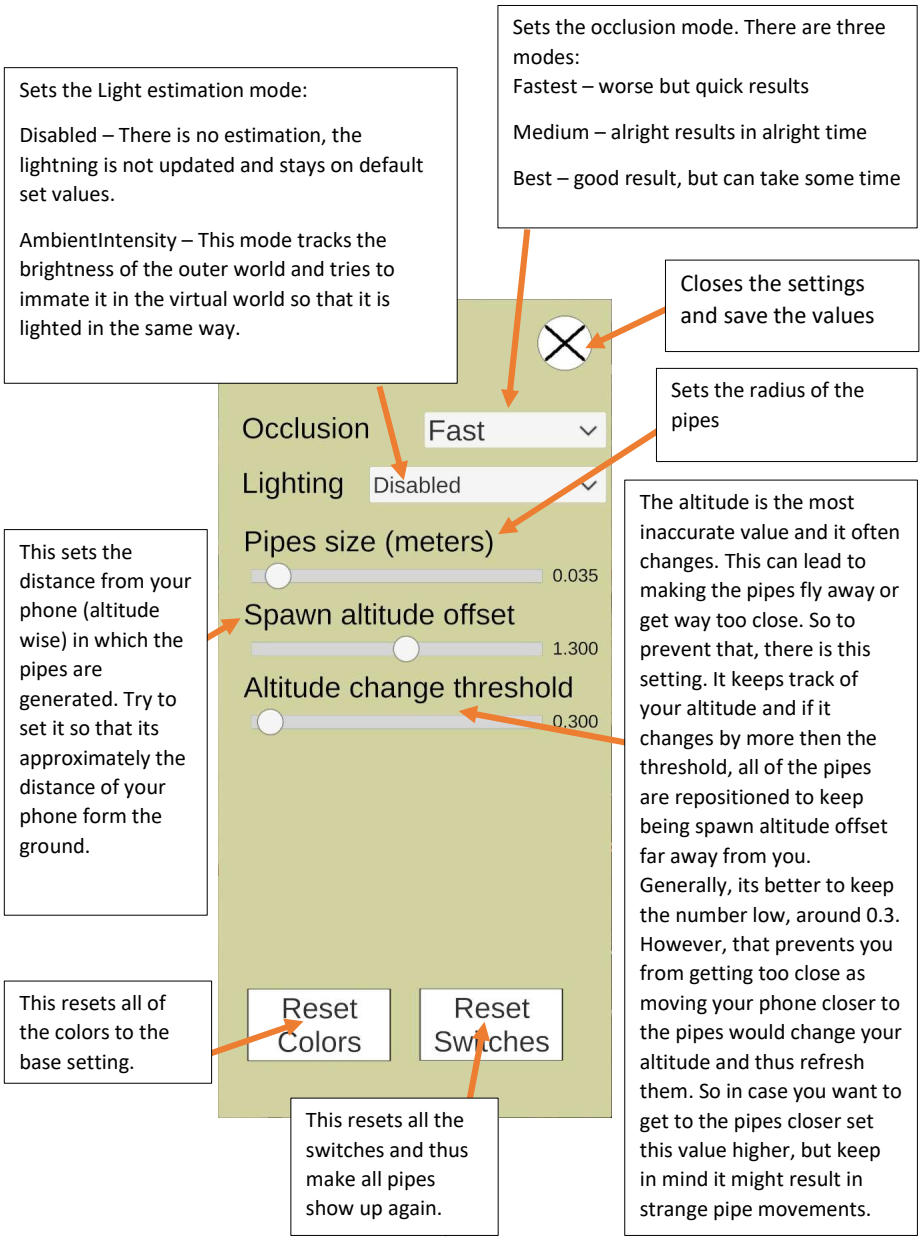
Tells you whether the VPS is available. However, beware that it doesn't have to be accurate when it says it is, as your location is still imprecise.

Tells you how is your current accuracy. To finish the initialization all three lines have to be green for at least one second.

Let's take a look at the menu first. When the menu button is clicked, the following screen opens. You can access the settings and change the color of the pipes or hide them.



Now to the settings menu, where you can adjust your app's settings.



Now let's see what happens when you are done with your calibration.

When you finish the calibration, the app starts processing your current position. It prepares the data about the pipes going around you. The amount of time this takes depends on the number of pipes and your internet connection speed. When the data are ready, the pipes will appear.

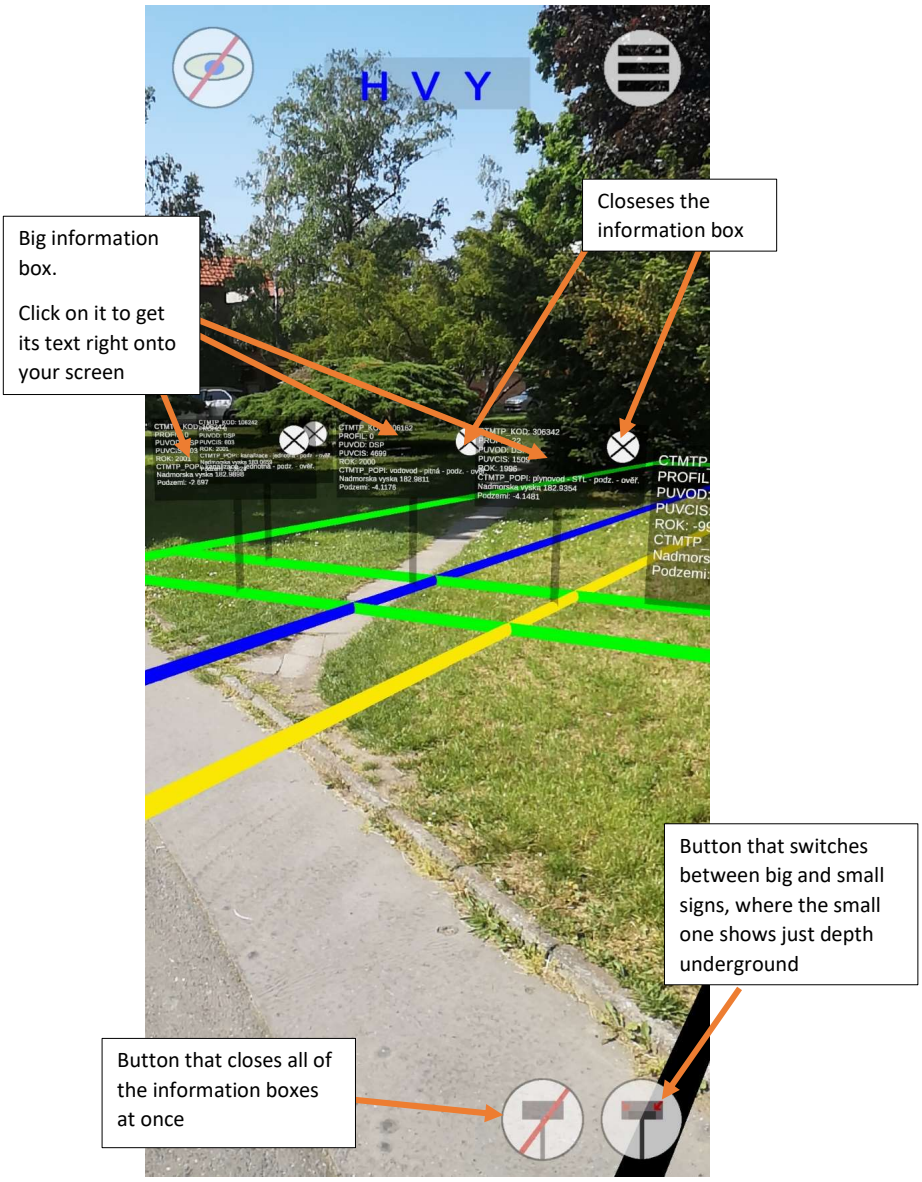
Keeps track of your position accuracy. As long as it stays blue/green you are alright. Blue is better than green.

You'll see this button if your device supports occlusion culling. And by pressing it you change it on and off.

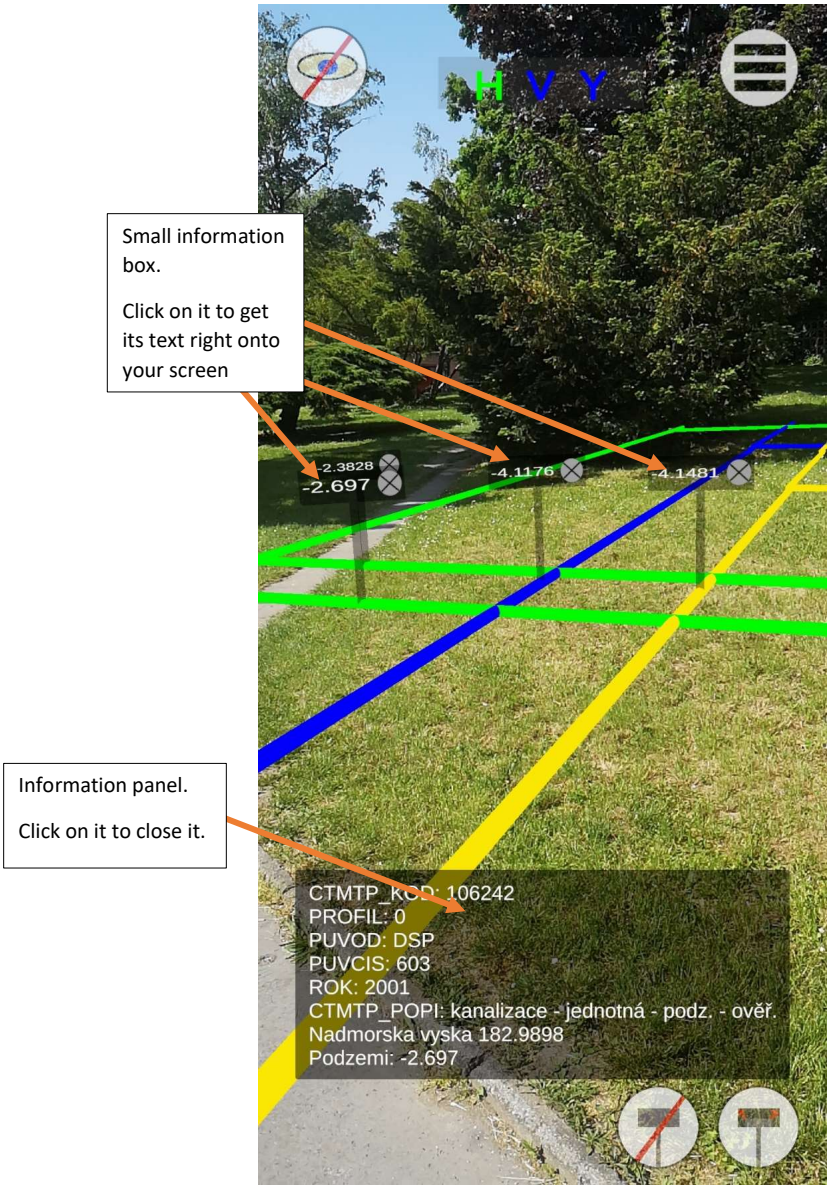
Beware that when the occlusion culling is on, your viewing distance lowers from 20 meters to 8 meters as that its current limit.

These are the pipes. That's why you are using this app. If you want to get more details about the pipe click on it.

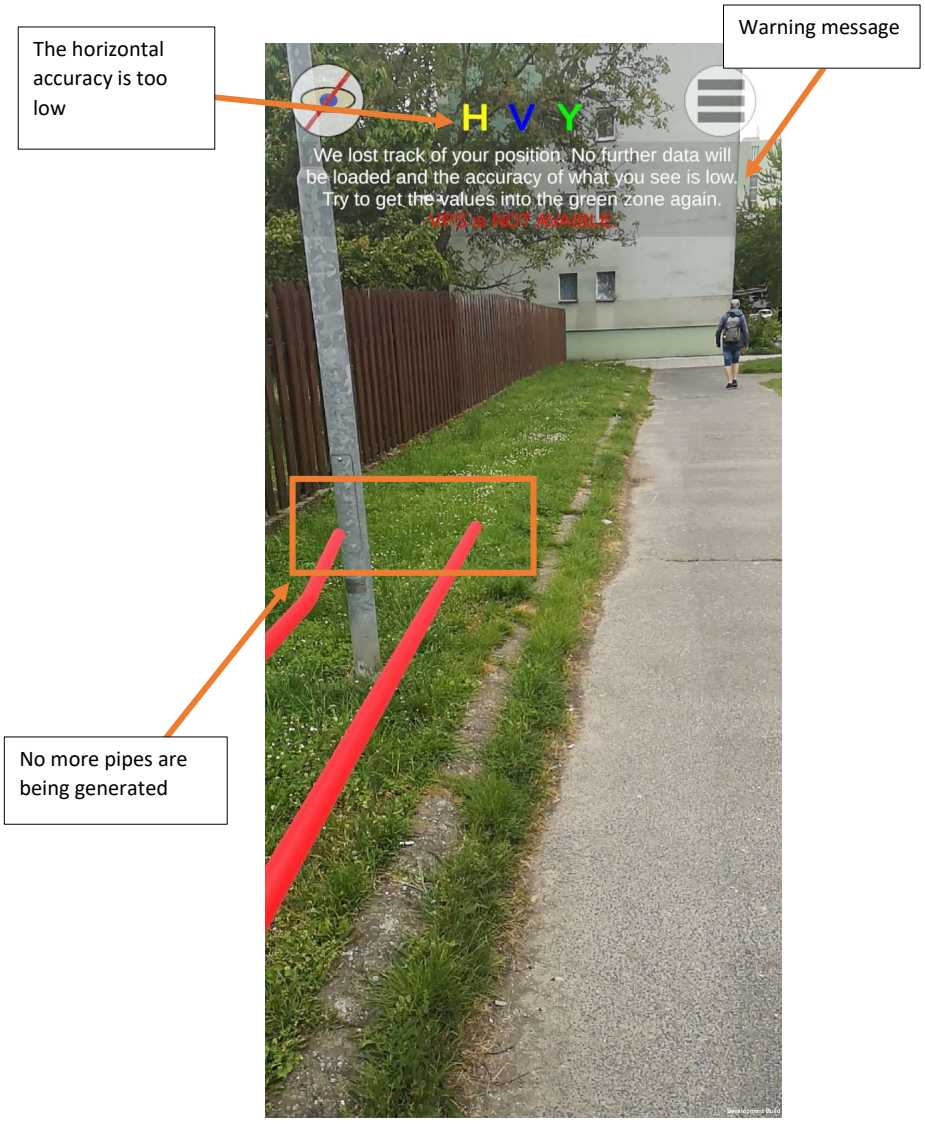
Let’s see what happens when you click on the pipe.
When you click on the pipe, the information box appears. It gives all the details that were available in the database. On top of that, it tells you what the pipe’s altitude is and how deep beneath the ground it is compared to you. You can create as many information boxes as you like.



You can always take a step back to read the sign when you are too close, but what if the pipe is too far from you and the information box becomes unreadable. In that case, CLICK on the sign. When you click on the sign, the information appears on your screen and stays there until you click the information box.



Finally, let's discuss what happens when your position accuracy gets too low. A warning message will appear when one of the three values gets from the green zone to the yellow zone. The application will keep showing you the pipes it has already loaded. However, it will not download more data or generate new pipes until you improve your accuracy.



And that's it. This should cover everything there is to this app. Good luck with your pipe hunting!