TU WIEN Informatics

Interactive Web-based 3D Solar Shadow Map

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Visual Computing

eingereicht von

Georg Molzer, BSc.
Matrikelnummer 0525148

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Mag.rer.nat. Dr.rer.nat. Georg Gartner
Mitwirkung: Dipl. Ing. Florian Ledermann

Wien, 20. Februar 2020

Georg Molzer	Georg Gartner

TU WIEN Informatics

Interactive Web-based 3D Solar Shadow Map

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Visual Computing

by

Georg Molzer, BSc.
Registration Number 0525148

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Mag.rer.nat. Dr.rer.nat. Georg Gartner
Assistance: Dipl. Ing. Florian Ledermann

Vienna, 20th February, 2020

Georg Molzer	Georg Gartner

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Georg Molzer, BSc.

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 20. Februar 2020

Georg Molzer

Danksagung

Ich bedanke mich bei meiner Mutter Silvia & meinem Vater Konrad. Dafür, dass sie mich in dieses Leben geholt haben und mir weise, liebevolle und kritische Lehrer und Vorbilder waren. Ich denke hier auch an meine mittlerweile verstorbenen Großeltern und deren Vorfahren. Ihnen allen will ich diese Diplomarbeit widmen.

Florian Ledermann bin ich dafür dankbar, dass ich bei ihm mit der ursprünglichen Idee auf offene Ohren stieß und er mich motiviert hat, den damaligen Stand der Arbeit auf der 29th International Cartographic Conference 2019 in Tokio einzureichen — mein erster Vortrag auf einer wissenschaftlichen Konferenz. Florian hat mir zu Beginn dieser Diplomarbeit in einer schwierigen Zeit mehr geholfen, als ihm wahrscheinlich bewusst ist. Sein kritisch-positives und kluges Wesen hat mein Leben damals sehr positiv beeinflusst. Hervorzuheben sind auch seine Geduld mit mir, seine Hilfsbereitschaft und seine beeindruckende Korrekturleistung nach Fertigstellung.

Georg Gartner möchte ich dafür danken, dass er diese Diplomarbeit ermöglicht hat und dass er mir bei der Beantragung eines Stipendiums eine große Hilfe war — ein Stipendium, dass es mir erstmalig im Leben ermöglich hat, mich einer einzigen Sache mit Fokus zu widmen. Eine sehr heilsame Erfahrung. Über Georg erfuhr ich auch von der 16th Conference on Location Based Services, wo ich nach erfolgreicher Einreichung ebenfalls meine Arbeit an Solar Shadow Maps präsentieren durfte.

In diesem Sinne möchte ich auch der österreichischen Studienbeihilfenbehörde danken, die mir durch ein Studienabschlussstipendium diesen letztendlich erfolgreichen Abschluss meines Diplomstudiums ermöglicht hat. Ich danke auch den zahlreichen Autorinnen und Autoren der Werke, die ich im Zuge dieser Arbeit studierte, insbesondere auch den Menschen hinter Three.js, einer webbasierten 3D-Engine, ohne welche die Entwicklung des begleitenden Prototypen so nicht möglich gewesen wäre.

Ich danke Gott und dem Universum für diese Bühne und Schule.
Und ich danke der Sonne, dass sie scheint.

Acknowledgements

I want to thank my Mother Silvia & my Father Konrad. For bringing me into this life and being the wise, loving, and critical teachers and role models, I had needed. Them in mind, I also think of my grandparents and ancestors who have all passed away in the meantime. I want to dedicate this diploma thesis to all of them.

I am thankful for Florian Ledermann, who I found with open ears when I told him about this idea of a diploma thesis, and who motivated me to submit my work to the 29th International Cartographic Conference 2019 in Tokyo — my very first presentation at a scientific conference. At the beginning of this thesis, during a difficult time in many regards, Florian helped me more than he is probably aware of. His critical-positive and bright nature has influenced my life in a very positive way back then. I also want to emphasize his patience with me, his helpfulness, and his impressive performance in correcting the final work.

Thanks go to Georg Gartner, who enabled this thesis in the first place and who was of tremendous help during my application for a student scholarship. A scholarship that, for the first time in my life, allowed me to focus on one single thing. This was a very healing experience. Through Georg, I also found out about the 16th Conference on Location Based Services, where I eventually presented my work on solar shadow maps as well.

With this in mind, I would like to thank the Austrian Study Grant Authority, for granting this very scholarship that enabled the eventual completion of my diploma studies.

I also want to thank the numerous authors of papers I read and researched during my work on this thesis, especially the people behind Three.js, a web-based 3D-engine, which was crucial to the development of the accompanying prototype implementation.

I thank God and the Universe for this stage and school.
And I thank the Sun for that it shines.

Kurzfassung

Die Sonne hat fundamentalen Einfluss auf die Erde — und damit auch auf die meisten Lebewesen, die sie beheimatet. Temperatur, Photovoltaik-Potential, Wachstumsraten von Pflanzen und auch physische wie psychische Gesundheit von Menschen stehen in direktem Zusammenhang mit ihrer Verfügbarkeit. Im Gegenzug lebt heutzutage der Großteil der Menschheit in Städten, deren Gebäude selbst immer höher werden, was direkten Zugang zur Sonne tendenziell schwieriger macht.

Im Zuge dieser Diplomarbeit wird daher argumentiert, dass es vorteilhaft wäre, hätten Menschen ein Werkzeug, mit dem sie Sonnenschatten besser verstehen könnten — sowohl inner- als auch außerhalb von Städten. Ein solches Werkzeug sollte in der Lage sein, beliebige dreidimensionale schattenspendende Strukturen, wie Gebäude, Terrain und Vegetation, sowie die konkrete Sonnenposition zu berücksichtigen, um entsprechende Sonnenschattenszenarien präzise visualisieren zu können. Sinngemäß wird die Frage gestellt, ob eine benutzerfreundliche, webbasierte Applikation, die mittels Echtzeitvisualisierung eine interaktive Erforschung von Sonnenschatten ermöglicht, technisch realisierbar ist.

Literatur zu dem Thema ist facettenreich und erstreckt sich über historische "Solar Shadow Maps" hin zu technischen Themen, wie z.B. Schattenvisualisierung in der Computergrafik, Datenintegration oder sonnenbezogener Astronomie. Existierende Applikationen zeigen zwar Potential, schaffen es aber nicht, voran erwähnte Eigenschaften in sich zu vereinen. Daher wird eine Methodik mit dem Ziel der Entwicklung eines funktionalen Prototyps definiert, der die Mängel bestehender Anwendungen kompensieren soll. Diese Methodik inkludiert unter anderem grundlegende Designaspekte, notwendige Algorithmen, sowie Überlegungen zu Datenintegration und Art der Visualisierung.

Der implementierte Prototyp wird anschließend in Bezug auf Interaktivität sowie Qualität getestet: Seine Performance wird dazu auf verschiedenen Endgeräten evaluiert und generierte Visualisierungen werden mit ihren jeweiligen realen Pendants verglichen. Es kann letztendlich gezeigt werden, dass eine interaktive, präzise, dreidimensionale und webbasierte Solar Shadow Map tatsächlich realisierbar ist; einsetzbar auf dem gesamten Planeten — und potentiell darüber hinaus.

Abstract

The Sun impacts Earth and most lifeforms it is inhabited by. Factors like overall temperature, photovoltaic potential, plant's growth rates, and even health and mental conditions in humans, are directly correlated to its presence. Nowadays, the majority of people live in cities, consisting of ever taller building structures, occluding more and more sunlight. Thus, humans are getting increasingly restricted from direct access to the Sun.

This thesis claims that a tool, enabling humans to gain a better understanding of solar shadows in cities and around the world, would be beneficial. Therefore, it provides motivational arguments from various scopes and disciplines. Such a tool should be able to consider relevant three-dimensional occluding structures such as buildings, terrain, and vegetation, as well as the actual Sun position, and visualize respective shadows for arbitrary points in time, providing predictability of solar shadows. Eventually, the thesis raises the major question, whether such a tool could be implemented as a user-friendly, web-accessible application, which — through real-time visualization — fosters interactive exploration of the Sun and its shadows.

Related literature is multifaceted, covering the history of "solar shadow map" attempts, as well as engineering aspects, like appropriate methods of shadow visualization, data integration, or Sun-related astronomy. There are also existing approaches, whereas some of them were already investigated in 2014. While they show potential, none of them integrated all desired features, as they were defined above, within one unified application. Therefore, a methodology towards a capable prototype implementation is framed, covering fundamental design aspects, as well as more detailed thoughts on data, 2D versus 3D visualization, and required algorithms.

Based on this methodology, a prototype application is implemented and reviewed for its fulfillment of the aforementioned requirements on interactivity and quality: A performance test on various hardware is undertaken, and generated visualizations are compared to their real-life counterparts. It is eventually shown that an interactive, web-based 3D solar shadow map is, in fact, something that can be built and could work on the whole planet Earth — and potentially even beyond.

Contents

CHAPTER 1

Introduction

"There is strong shadow where there is much light."

— Johann Wolfgang von Goethe

The Sun is a prerequisite for life on Earth. Its gravity pulls the Earth into the so-called habitable zone, that very orbital distance that is neither too cold nor too hot to sustain life. The Sun provides us with radiation to warm, illuminate, and nurture. Sunlight is the basis for plants' metabolism, and plants themselves are food for other species, including us humans. And it is not just about temperature and food: Energy production via photovoltaics, health and mental condition in humans (Mead, 2008), real estate, or tourism — areas where sunlight has an impact, are manifold. Sunlight does, however, not only have benefits. In a time of global warming, some regions on Earth heat up beyond comfortable or even life-sustaining levels, and there are health conditions that require severe restrictions on Sun exposure. Given the importance of sunlight, a convenient and intuitive way to understand and predict its behavior might be justified.

On Earth, there is not always sunshine. Any non-transparent physical object potentially occludes sunlight, and among them is Earth itself: Hence, one half of it is always in the shade. That half changes continuously due to the Earth's rotation around its axis and its orbit around the Sun. But even the lit hemisphere, oriented towards the Sun, has shaded areas — occluded by structures, such as mountains, trees, or buildings. Depending on the Sun's angle over the horizon and its azimuth, cast shadows vary in area and shape. The closer the Sun is to the zenith (e.g., at noon in summer), the less the shadows cast by vertical structures are and vice versa (e.g., early afternoon in winter). Seasons have an effect as well: It is typical not to see much of the Sun during winter in European cities — a circumstance that was a significant initial motivation for this thesis.

1.1 Motivation

"The motivation for this work manifested in early 2014 during a dark winter in Vienna, Austria. I was busy working in Vienna's central districts and for two consecutive weeks,

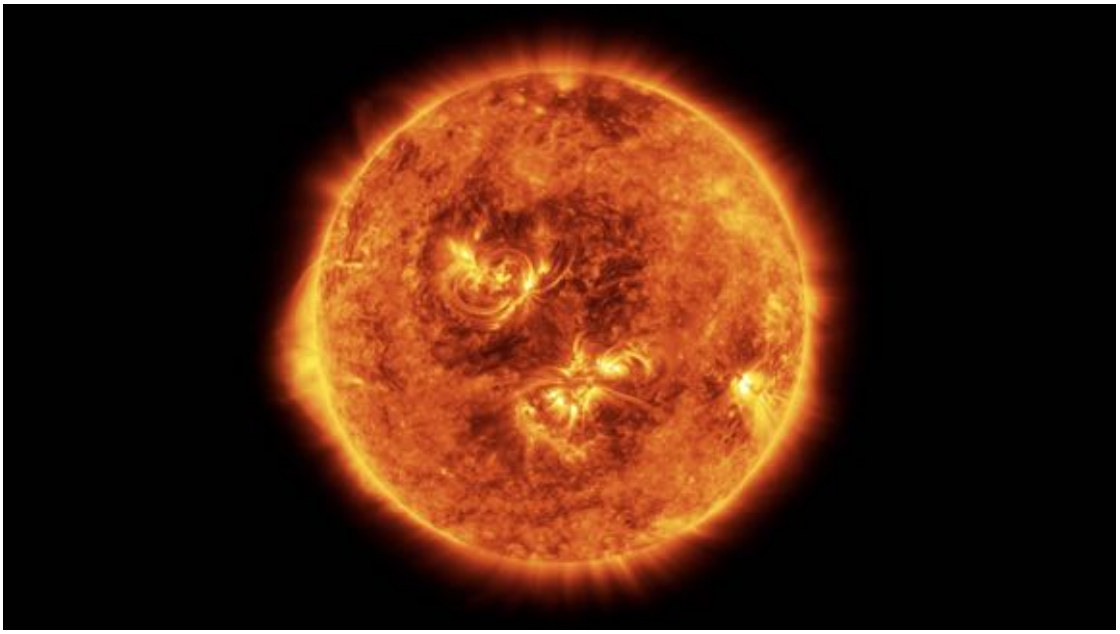


Figure 1.1: The Sun. Image source: NASA

I could not see the Sun's disk in the sky. Most of the time, buildings blocked it, or it was just too foggy and cloudy to be visible. It did not even matter whether I was at home, at work, or on my way between these two spots. I felt tired and weak, slightly depressed and unhealthy, and wanted to know where in the closer area I could go to spend half an hour in the Sun. A map that visualizes solar shadows at a given time so I could see where I could go. From my understanding back then, the required three-dimensional data should be available to enable necessary calculations and visualizations to help me find the Sun. Initial research back then was, unfortunately, underwhelming as I did not find any convincing tool."
— Georg Molzer

1.1.1 First attempts on end-user applications in early 2014

Long before this thesis was even considered, a search for suitable solutions began. All in all, the solar shadow visualization situation in 2014 was sobering, however, demonstrating the existence of a few engineers and cartographers, who cared about the problem:

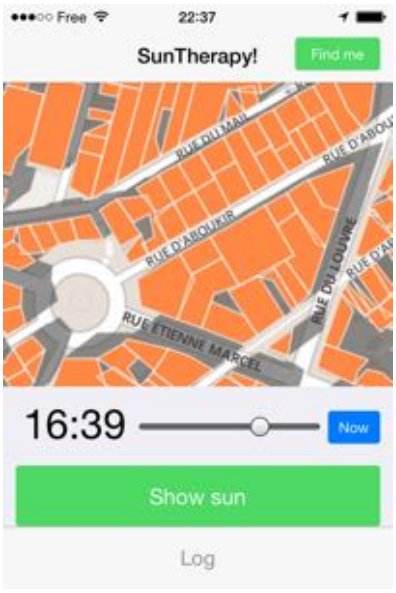
- **SunTherapy:** Reverse-engineering a provided screenshot (see Figure 1.2a) and adding the little info still available in a Twitter-post (twitter.com, 2014), one can assume that the basis for time-dependently visualized shadows is extruded ground plans of buildings, based on *OpenStreetMap (OSM)* data. There is no hint on considering terrain or more complex roof structures as shadow casters. A time

slider on the bottom probably allows to change the time, hence the Sun's position and, therefore, the shadow arrangement.

- **Beer in the sun:** "A data mapping project of the sun in relation to pubs & bars. This free website & app will allow people to find a pub in the sunshine." (kickstarter.com, 2013). The founders tried to collect funds via a Kickstarter-campaign to implement a web-based application that should help find pubs and bars in the Sun. Based on current location and arbitrary time — limited to summer months — the service was supposed to list the closest locations as well as empirical periods when each of these places has a sunny spot to sit (see Figure 1.2b). Starting with the city of London, they planned to manually collect data of available restaurants in a "painstaking data aggregation process" (ibid.).
  What is extraordinary about the endeavor is the manual data collection process. Instead of modeling the three-dimensional environment and the Sun's position, the problem is reduced to its very essence of the given use-case, i.e., manually collect: 1) name of the bar/pub/restaurant, 2) time at which the location becomes sunlit and 3) time at which the site lies in shadow again. This data would then be migrated to a database and made available via various web, and native app front ends for their users. In the fundraising campaign, it is mentioned that the primary cost factor is human-driven data acquisition.

- **Pints in the sun:** Another UK-based attempt about finding a place to have a drink in the Sun (pintsinthesun.co.uk, 2014). Ground plans of buildings were — similar to *SunTherapy* — while Foursquare (`www.foursquare.com`) provides locations of pubs. The web-based 3D engine Three.js (`www.threejs.org`) was then used to render a 3D model, which is a big step from the aforementioned *Beer in the sun* project: Instead of manually gathering Sun data of specific locations, this attempt creates the shadow occluders (i.e., *OSM* sourced extruded buildings) in a virtual 3D scene, which in turn enables shadow rendering from various sunlight positions, based on arbitrary time and day of the year. Information about sunlit places and those in the shade is liberated from specific, hand-picked ones and instead available to any area covered by 3D data. As a side effect, the ponderous and expensive process of manual data collection, is rendered obsolete.
  As of writing this thesis, the web application lost its functionality. Back in 2014, it was possible to pan and zoom a 2D map interactively; therefore, define a section of the plan. By manual update, that section is recreated as a 3D scene in a semi-interactive panel (i.e., it was not possible to change the framing, perspective, or zoom) with a light source that, however, could interactively be changed by dragging a time-slider on the bottom. The light source would then update according to data provided by SunCalc (SunCalc, 2016), and solar shadows would update within the otherwise static map section.

Evaluating the investigated tools, *SunTherapy* had several drawbacks, especially in the geological and architectural context of Vienna: The lack of terrain inclusion means



(a) SunTherapy app screenshot. Image source: twitter.com (2014)

(b) Example of London pubs' and bars' sunny hours in summer. Image source: kickstarter.com (2013)

Figure 1.2: Different attempts on communicating solar shadow scenarios back in early 2014.

that hills and mountains are not considered as occluders. Mainly the west/north-west of Vienna in the summer, and the west/south-west in winter are affected by earlier sunsets due to mountains in the west. Furthermore, the city is hilly — their absence in a visualization would at least negatively influence shadow precision.
Buildings, on the other hand, are reduced to extruded floor plans, which results in flat roofs only. Churches or houses with pitched roofs are inherently misrepresented, up to being completely unrecognizable. For example, the floor plan of *Stephansdom* (a landmark church in the center of Vienna) has only remote similarity with its shape around the tower and roof. The *Eiffel tower* in Paris would be a similar example, even more, due to its light-permeable lattice structure. That difference in three-dimensional shape also affects cast shadows: The more the original shape diverges from its extruded floor plan, the stronger. Regarding the user interface, it seems to lack options to set a specific date (i.e., day of the year) as well as to navigate to a custom address.

Beer in the sun did not reach the crowdfunding goal and was probably never realized. It tells how variable potential approaches to the comprehensibility of solar shadows can be. On the possible spectrum of encompassing Earth's surface, they chose to be on the other end and focus on hand-picked locations that were researched by humans and manually added to the system. Deliberately leaving out seasonal changes of the Sun's azimuth and altitude, their users would have still been able to expect high-quality recommendations: In contrast to an *OSM* or similar public data-driven approach, pitched roofs, a large

tree in a private backyard and other obstacles are considered by a human evaluating on-site. The dramatically reduced scaling of manual data acquisition, on the other hand, inherently limits coverage.

Pints in the sun, the seemingly nominal and logical successor that was actually functional, went the simulation route instead of manual data collection. The drawbacks cover still extruded floor plans of buildings, no terrain and no vegetation (e.g., trees) as shadow occluders. On top of that, the shadow visualization did not allow interaction with the scene: One could not change zoom or perspective; hence, investigate shadows on facades or other vertical structures. Since map-clipping and shadow visualization were decoupled into a two-step process, interactive exploration of a shadow scenario was needlessly hindered.

Back then, conclusively the question emerged if this can be done better. If beyond personal use, there is even a necessity to improve the situation on *interactive solar shadow maps*. Conversations about the topic were held, winters in Vienna passed, and use-cases were identified. There seemed to be motivating resonance for the topic not to be forgotten.

1.1.2 Relevance of the Sun from a historical geoscientific perspective

As stated at the beginning of this chapter, the Sun impacts numerous areas and lifeforms, and this is obviously not just a recent development: In his master's thesis from 1975, titled "Solar Shadow Maps", Duffield (1975) provides a fascinating collection of backgrounds, engineering attempts (which will be covered in Chapter 2), as well as potential applications for solar shadow maps from a geoscientist's perspective. While the technology surrounding these areas changed dramatically within the last four decades, their inherent relevance has not. Duffield (1975) categorizes five systems significantly influenced by the Sun and gives examples of affected phenomena, arguing that they could benefit from solar shadow maps. In the following, those fields are further divided, and for those, which underwent significant technological progress in the meantime, today's perspective is provided.

1. **Microclimatic systems**
   Among all further systems, this one is the broadest, covering Earth's surface temperature in general, as well as water evaporation, energy balances regarding heat and radiation, air temperature, and humidity in detail. Local temperature differences cause pressure imbalances, thus, affecting air currents and wind velocity (Duffield, 1975).

2. **Hydrologic systems**
   Sun influences evaporation, which in turn affects water balances, soil moisture, and surface runoff; absence of solar soil allows *permafrost* (Duffield, 1975), that is "ground with a temperature remaining at or below 0 °C for at least two consecutive years" (Biskaborn et al., 2019, p.2), which, between 2007 and 2016, is warming up and, therefore, disappearing globally (ibid.). Furthermore, snowmelt is affected,

which in turn influences — besides general water balance as well — glaciers (Duffield, 1975).

3. **Geological systems**
   Geomorphic processes, such as mechanical and chemical weathering of rocks, are influenced by solar radiation. Due to the Sun's impact on soil moisture levels, also the soil's chemistry changes — influencing, which types of sedimentary rocks form in the first place (Duffield, 1975). Sunlight also correlates with erosion rates, indirectly affecting the steepness of slopes (ibid.) — which in turn has a circular connection to insolation itself.

4. **Biological systems**
   Besides direct effect on plants and animal survival and behavior, biological systems are obviously affected through the aforementioned influence of the Sun on microclimatic, hydrologic, and geological systems (Duffield, 1975). Among others, plants' temperature, transpiration, respiration, and especially photosynthesis are a function of insolation. Seed germination and survival, growth, and size of forest trees, as well as their ring thickness, are influenced (ibid.). The survival of some plant species, in turn, allows other plants and animals to survive within the shade they provide (ibid.): It is a linked, species-spanning system. In regards to cultivated plants, agriculture and forestry are reliant on sunlight — affecting water requirements, plant survival, growth rates, and yield (ibid.).

   Also human and animal survival is dependent on the Sun: Not only are both nurtured — directly or indirectly — by plants, they also need specific temperature ranges to sustain their organisms in accordance with their thermoregulation. While the Earth is within the Sun's habitable zone, it is still possible for terrestrial organisms to freeze to death or die from overheating.

5. **Industrial systems**

   a) Before humanity started using fossil fuel as energy storage, pretty much everything was based on the Sun as an — either direct or indirect — **energy source**. While still being in the midst of this period, Duffield (1975), however, predicts that "as fuel supplies dwindle, solar radiation patterns will regain their importance to industrial cultures" — a forecast that, 45 years later, can be generally confirmed (see Figure 1.3).
   Duffield (1975) argues that solar shadow maps could guide the selection of construction sites for solar power plants. In contrast to his approach, these maps, however, would have been benefitting from averaging **solar potential** over the whole year — instead of providing just momentarily insights for specific days: While a potential site might provide good energy production during summer, it could be occluded in winter, and since structures of the size of a solar power plant are inherently stationary, annual energy potential is the relevant factor.

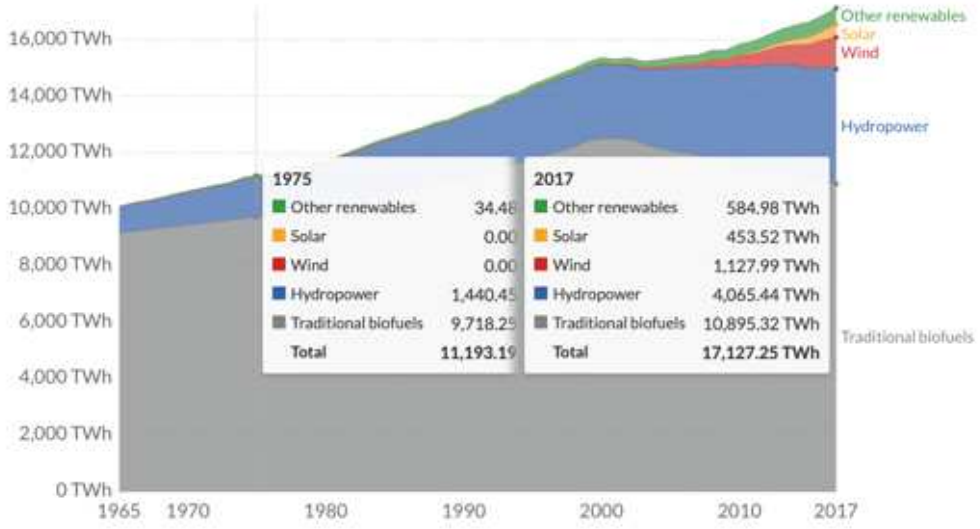


Figure 1.3: Global renewable energy consumption per year. Back in 1975, solar power was practically nonexistent. It took until the early 2000s for it to gain traction, as traditional biofuels (fuelwood, forestry products, animal and agricultural wastes) started to decline. Source of the chart: ourworldindata.org (2019), based on data by bp.com (2018)

As solar power became more relevant in the 2000s (again, Figure 1.3), research in this field intensified: Freitas et al. (2014) provide a state of the art review of various 2D and 3D solar potential visualizations within urban contexts that, besides terrain, also consider building structures and vegetation. Catita et al. (2014) expand the often communicated photovoltaic potential of city roofs, which are inherently more horizontal structures, with that of vertical facades, to support civil officials in selecting efficient locations for photovoltaics. Therefore, required 3D representation of insolation on buildings is generated from a Light detection and ranging (LiDAR) sourced Digital surface model (DSM), which is combined with an astronomical as well as a radiation model. Results are then stored inside a 3D Geographic Information System (GIS) database — a source for further analysis and applications. Wieland and Wendel (2015) compute solar radiation based on 3D City Geography Markup Language (CityGML) building models and apply their method to 13,000 buildings within the city of Karlsruhe, Germany, while seemingly neglecting terrain in the process. Murshed et al. (2018) want to enable city planners to gain further insight into urban photovoltaic potential by providing flexibility in terms of time and space: Their numerical model can produce results for arbitrary spatial (i.e., surface vs. building vs. district) as well as temporal extent (i.e., day, month, year). Bremer et al. (2016) argue that effective planning of solar panel construction and passive energy measures require regional as well as household scales to be equally considered. The data generated by long and

short-range shadows, however, requires the introduction of a multi-resolution approach capable of handling both coarse as well as fine domains within the same system. A solution to this challenge is presented. To increase accessibility of generated data, results are compatible with 3D standards like CityGML or COLLAborative Design Activity (COLLADA).

b) According to Knowles (1974), investigated cliff dwellings in Tsegi Canyon, Arizona, had their openings oriented towards the south: Consequently, during summer, the caves' overhang provided shade, while due to the lower altitude of the Sun in winter, its rays could pass into the homes, warming them. Duffield (1975) lists many examples in regards to **urban planning and architecture** from the 1950ies to the 1970ies: Strategies to take advantage of the Sun (and its lack) in the design and location of new buildings seemed to have gained popularity in the 1950ies. Knowles (1974) even envisioned "three-dimensional mega-structure cities designed with consideration for Sun path geometry." (Duffield, 1975).

c) Moreover, Duffield (1975) describes how human settlement in the Alps developed from centuries of trial and error in regards to respecting insolation, into a structured approach, oriented towards the Sun and shadow — affecting forests, pasture, farmland and housing (ibid.). Again, from nowadays point of view, his perspective on industrial and ecological systems, which seemed very much woven into 1970ies society, is telling: *"They have evolved and spread across the landscape through a series of many small individual decisions and actions. Frequently, habitual industrial patterns have been thrust into new environments for which they were not designed (for instance, large glass windows in buildings of hot deserts), and have been sustained there, only by use of heavy fossil fuel subsidies. We can afford such subsidies less and less."* (Duffield, 1975, p.52-53). Knowles (1974) proclaims that human survival is only possible if the industry takes inspiration from biological systems, and it would benefit by, as Duffield phrases it, "taking advantage of naturally occurring matter and energy gradients and flows." (Duffield, 1975, p.53) He concludes that ***"it will be to our advantage to make such adjustments consciously, now that we know the natural patterns better, rather than unconsciously through inefficient, time-consuming trial and error (as in the past). Perhaps solar shadow maps will be one tool to assist us in our conscious industrial evolution."*** (Duffield, 1975, p.53)

6. **Human systems**

a) The Sun already had a prehistoric impact on cultures, being the driver to build monuments like *Stonehenge* in Wiltshire, England, or the *Pyramid of the Sun* in Teotihuacan, which both where constructed in alignment to specific Sun positions (Duffield, 1975). Stonehenge, for example, provides structures that are, among others, oriented towards the rising Sun on the winter solstice,

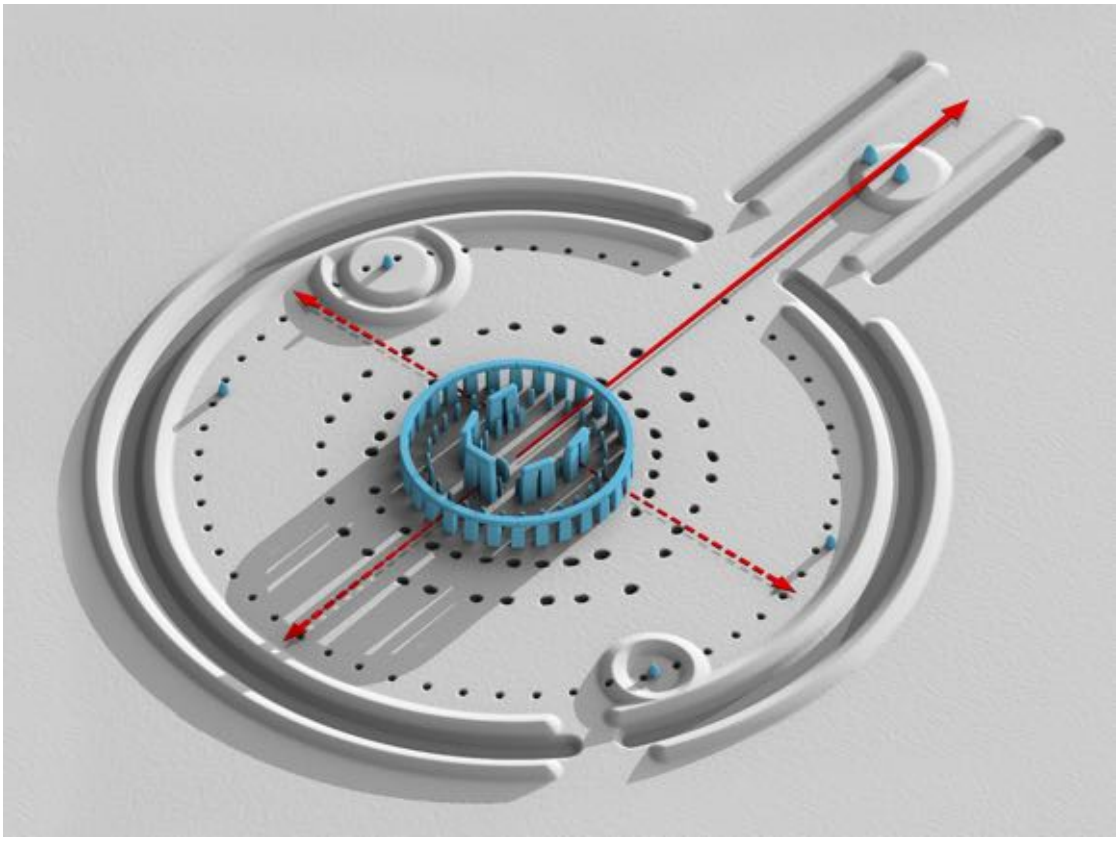


Figure 1.4: Rendering of the formerly complete site of Stonehenge, as it probably looked like ages ago. The red arrows depict the cardinal directions, whereas the solid arrow is directed towards the north. Stone structures are in blue while mounds are in white. Image source: wikipedia.org (2019b)

as well as the setting Sun on the summer solstice (wikipedia.org, 2019b). Figure 1.4 also visualizes its alignment with the cardinal directions. These rhythms, in turn, are the expression of seasonal changes directly affecting those cultures, which often ended up being **ritualized** (Duffield, 1975) — whereas some of these rituals persist to the present day, albeit in altered form.

b) Already considered in the introduction (see *Use-cases*, 1.1.5), Duffield (1975) mentions the Sun's impact on **recreation**: From sunbathing on Puerto Rico's beaches, seemingly taken seriously by the island's officials, to playgrounds and parks at higher latitudes (ibid.). Snow — the requirement for many winter sports activities — is, among other factors like air temperature, cloudiness, and wind speed, most sensitive to solar radiation (Hendrick et al., 1971). They investigated how the Sun influences snowmelt in northern New England and its impact on a local ski resort: *"Open south-facing slopes usually lose their snow cover in early or mid-March while high, north-facing, coniferous-*

forested slopes retain snow into mid- or late May. Individuals involved in activities dependent on snow cover, or the melting of the snow cover, become quite sensitive to the time and space variations of snowmelt. For example, a Vermont ski resort operator with trails on the south facing of the mountain now finds it necessary to operate the most extensive (and expensive) snow-making system in the state." (Hendrick et al., 1971, p.418). Production of artificial snow is a notable industry: Back in 2007, there were around 3100 "snow guns" (machines that produce artificial snow) operational in Austria alone (Krutzler, 2013) — this number soared to 25,000–30,000 in 2019 (Schnauder and Laufer, 2019). With these machines, 70 % of Austrian slopes (equalling 16,500 hectares), can be artificially snowed (Schnauder and Laufer, 2019; wko.at, 2018). As of a parliamentary request in February 2013, Austria's environment minister measured the energy required to operate the necessary infrastructure at 14 MWh per hectare and season — resulting in roughly 250 GWh per year (Krutzler, 2013). This approximates to the energy consumption of 62,500 European households within a year (odyssee-mure.eu, 2020), or the energy produced by Vienna's most powerful run-of-river plant over a three months time span (verbund.com, 2019).

1.1.3 Solar shadow in urban areas

According to Knowles (1974), who, as mentioned above, researched ancient cliff dwellings and their entrances, humans have a history in aligning their housing with the Sun. Nowadays, cityscapes, however, filled with large vertical building structures, often omit access to the Sun: Through taller buildings, the provided housing area, hence, profit increase — all while keeping land usage constant; a favorable approach, especially in areas of expensive land prices. The drawback, obviously, is more shadowing around those buildings. The concentration of people living in cities is increasing over the years: 2007 defined the tipping point when most of humanity ended up living in cities (50,1 %), compared to 36 % 40 years earlier (data.worldbank.org, 2019). This trend is linearly progressing, and as of 2018, 55.3 % of Earth's population resides in cities (ibid.). The more people live in cities or even *megacities*, the more are potentially affected by a lack of sunlight.

There are cities with far worse shadow situations than Vienna during winter: New York City, for example, characterized by its distinct skyscrapers, has streets that are completely blocked from direct sunlight over the whole year (see Figure 1.5). Not even during the summer solstice, the day of the year with the longest duration of the Sun being above the horizon and the Sun additionally reaching its highest position in the sky, a single direct ray hits the surface of the street. *"Sunlight and shadow shape the character and rhythm of New York's public spaces. They have the power to control the flow of foot traffic on our city streets and decide which plazas hum with activity and commerce and which stay barren and desolate."* (Quoctrung B., 2016)

Due to situations like these, some countries took legal action in an attempt to protect

(a) Looking towards the sky at Cedar Street, NYC. Image © Chang W. Lee/The New York Times

(b) Location of Cedar Street, encircled by tall buildings

Figure 1.5: 3 Cedar Street, blocked off the Sun by tall buildings, especially on its eastern end. As a result, it is one of the "shadiest streets in New York City" (Quoctrung B., 2016), receiving no direct sunlight even "on the winter solstice, the summer solstice or the autumnal equinox." (ibid.) Image sources: Quoctrung B. (2016)

their citizen's right for sunlight: Japan has *nisshōken* ("sunshine rights") "not simply to permit applications of solar energy but as a fundamental element of the 'civil minimum' as well as good health." (McKean, 1981, p.112). In England, the law of *Ancient Lights* exists: It is a property law enforcing that windows used as a light source "for 20 years or more could not be obstructed by the erection of an edifice or by any other act by an adjacent landowner." (Encyclopædia Britannica, 2018). It originated back in 1663. Ironically, "the doctrine did not acquire wide acceptance by courts in the United States" (ibid.).

1.1.4 Urbanization, health and the "right to light"

In their *Nature* article "Protect our right to light", Zielinska-Dabkowska and Xavia (2019) elaborate on the health, well-being, and sustainability impact of shadowing in urban areas. In 2018, 143 buildings taller than 200 meters were constructed, 88 of them in China alone (Zielinska-Dabkowska and Xavia, 2019), as shown in Figure 1.6. London is set to gain 510 more towers with at least 20 stories over the next decade (Kollewe, 2018). Taller buildings mean more shade and less Sun means decreased UV radiation on ground levels. Wai et al. (2015) measured and modeled urban scenarios in sub-tropical cities with high-rise buildings, concluding that in extreme scenarios (i.e., narrow roads, high buildings) only 10 % to 18 % of the UV exposure rate — compared to un-obstructed areas — remains (ibid.). Less UV radiation, on the other hand, has health impacts: Boubekri (2004) argues for *"daylighting legislation because of health"* (p.51): Studies found that psychiatric patients in sunny rooms had an average stay of 16.8 days, compared to those in dull rooms who stayed for 19.5 days (Beauchemin and Hays, 1996). Boubekri (2004)

lists studies investigating Vitamin D synthesis within the human body and how lack of UV radiation causes Vitamin D deficiencies, which in turn correlate with various issues like bone frailty or high blood pressure (ibid.). Sunlight — or the lack of it — further influences the human endocrine system, and is probably related to depressions, seasonal affective disorder (SAD), and drowsiness, and even seems to have an impact on our metabolism (ibid.).

According to Zielinska-Dabkowska and Xavia, most parts of the world lack sunlight-based regulation on tall buildings. Often, landowners have the right to build *above and below* their land, without (but physical and economic) limits. Besides the aforementioned *nisshōken* in Japan and *Ancient Lights* in England, San Francisco (California, USA) passed *"Proposition K, the Sunlight Ordinance"* in 1984 in "response to a growing concern about shadow impacts of buildings on the city's open spaces" (quoted from a description of it, available via Proposition K (2020)) — guiding San Francisco's Planning Commission on shadow impact of new (high) buildings. Therefore, it defines an absolute cumulative limit (ACL) for parks, that limits the amount of potentially sunlit public surface that can be taken away from a new building, averaged over a year (ibid.). If a building serves public interest, its ACL impact can be higher, and vice-versa (ibid.). Besides the area of shadows, also their duration is considered: Shadows close to the occluder move slower than those far away. Proposition K's recommendations are based on shadow analysis over a time span from June 21st to December 21st (summer to winter solstice), one hour after sunrise and one hour prior sunset, and there are different recommendations depending on sizes of public parks and shadowing (Proposition K, 2020).
Zielinska-Dabkowska and Xavia's work also mentions Zurich (Switzerland), which limits time-integrated total shadow casting of new tall structures to a maximum of two hours per day during winter (refer to ABV Zürich (2019)) — as well as the *"European Standard on daylight in buildings"* (DIN EN 17037) — which Germany was the first country to adopt in 2019 (Zielinska-Dabkowska and Xavia, 2019).

1.1.5 Contemporary use-cases

Next to Sun-affected systems and geoscientific application scenarios, assuming the availability of required data (i.e., three-dimensional data concerning light-occluding structures like buildings, terrain, and vegetation, as well as the Sun's precise position), a well designed interactive solar shadow map could assist in further present-day use-cases:

- **Photography:** Photographers could improve the predictability of shootings, particularly outdoors, and gain a better understanding of "blue" and "golden" hours through visualizations of affected regions.
- **Gastronomy:** It would allow people to find restaurants, bars, or pubs with outside seating, which is explicitly in the Sun for a given time — or not, if the aim is to avoid it (e.g., during hot summer days). Then again, operators would be able to market such areas.

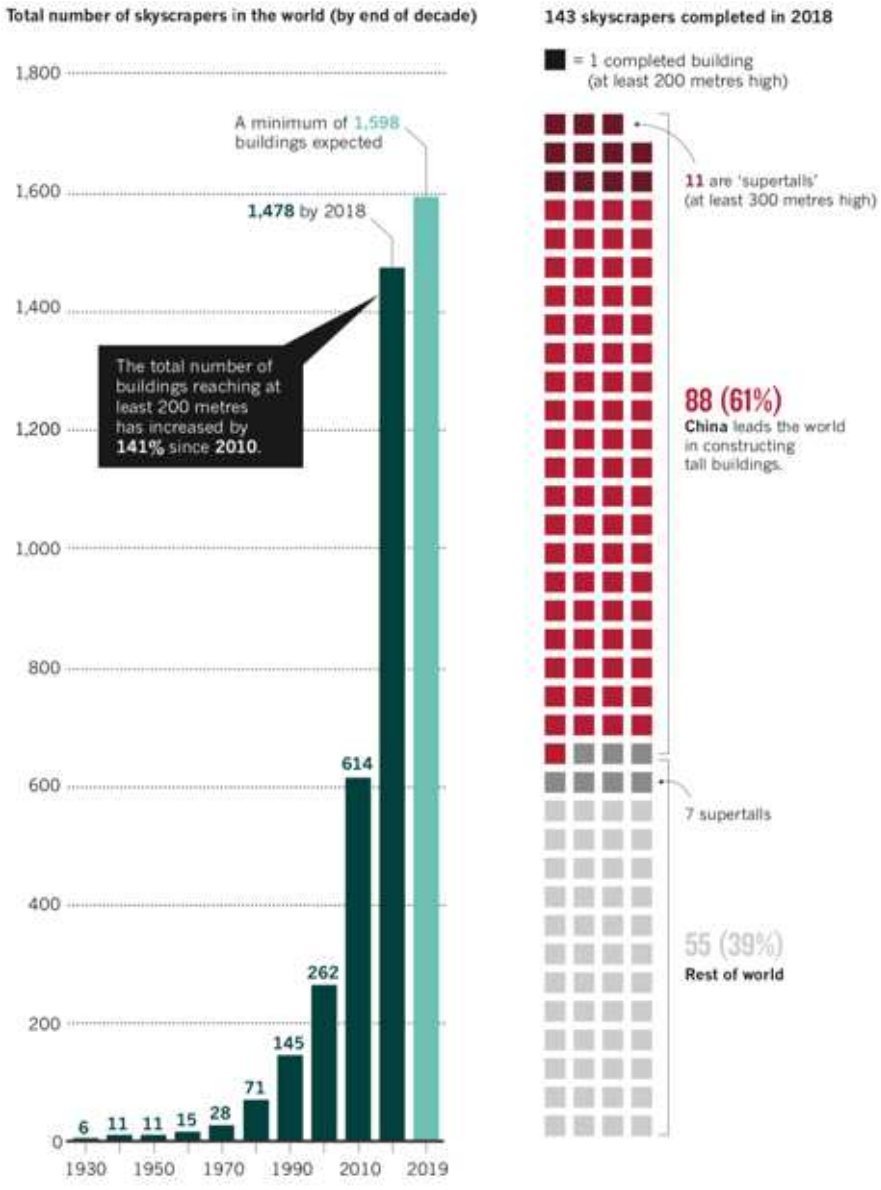


Figure 1.6: Skyscraper construction (at least 200 m high) in China and the rest of the world. Source of the chart: Zielinska-Dabkowska and Xavia (2019) based on data by the Council on Tall Buildings and Urban Habitat (ctbuh.org)

- **Solar cars and parking:** Currently, it is convenient to park cars in the shade during hot days to save time and energy to cool the interior down again. In the future, when solar cars might be more mainstream, this will shift to the opposite. Autonomous solar cars could even drive themselves into a sunny parking spot. In both scenarios, a solar shadow map would provide the required locational information.

- **Decentralized energy production:** There are many different providers for maps on photovoltaic energy potential on roofs. However, facades or balcony fronts are usually not taken into account since they cannot inherently be visualized on a two-dimensional map.

- **Urban gardening & facade greening:** Gardening in urban areas can be challenging due to frequent sunlight occlusion caused by tall buildings in the neighborhood. An insight in how the light transitions during the day, and among the seasons, helps to choose the right terrace or balcony for better growth. Comparable aspects apply to facade greening.

- **Real estate:** People in the market for a new apartment or house would understand how the light situation changes through the day and the seasons: Crucial information for such a long-term decision. Real estate agents, on the other hand, could transparently communicate how the light behaves, and emphasize exceptional objects.

- **Tourism, leisure industry & sports:** These sectors would be enabled to promote areas and activities based on their Sun exposure. Furthermore, athletes would be able to plan outdoor tours with increased safety: For instance, for ski or snowboard freeriding in the backcountry, knowledge about Sun exposed slopes enables more accurate avalanche hazard estimation.

1.1.6 Considerations & challenges

How could a tool that helps humans understand the Sun potentially look like? What information would it provide, and how should it be used by humans?

Interactivity

The term "map" appeared in the texts above. A map — in a classic sense — brings up the notion of a printed or drawn piece of paper, material that is inherently limited to show static content. It could be argued that it is sufficient to take that map content and augment it with additional shadows. The remaining relevant question, however, is: *At which time.*

Just as the Sun, solar shadows move and change *constantly.* There are no quantifiable steps, hence, a static approach of showing them is practically meaningless. *Interactivity* in regards to time, therefore, is a crucial feature of a solar shadow map, as it must

be possible to *adjust* it and retrieve an accordingly updated representation within a reasonable time frame; short enough to understand the link between the own action and the updated result. Two criteria, for a visualization to be considered interactive, are described by wikipedia.org (Interact. Vis.): 1) At least some aspect of the visualization needs to be controlled by a human, while 2) human interaction needs to be reflected by the application within reasonable time: "In general, interactive visualization is considered a soft real-time task" (ibid.). "Soft real-time" is one of the possible quality differentiations in regards to real-time computing, meaning that the usefulness of a system negatively correlates with a delay in presented results (wikipedia.org, Real-time). The others are *firm* and *hard* real-time, where the miss of an agreed-on deadline can even mean total failure of a system (ibid.). For an interactive solar shadow map, this means that the visualization is updated at *interactive frame rates*: *"Framerates measure the frequency with which an image (a frame) can be generated by a visualization system. A framerate of 50 frames per second (frame/s) is considered good while 0.1 frame/s would be considered poor."* (wikipedia.org, Interact. Vis.). Within this broadly formulated range, the own benchmark is, however, set to at least 24 Frames per second (FPS).

Assuming there were only, for example, four different positions for the Sun in the sky during every day of the year, and between them, there are hard cuts: It would be enough to pre-render (*offline*) four variations of 2D map tiles that cover a specific area. Obviously, this scenario is, in fact, not the case: A pre-rendered map like this would need an extreme amount of tiles, i.e., one set for every different Sun position. And given its unquantifiable nature, the number of sets would converge towards infinity — an extremely data-intense task with little practicability.
Online rendering, however, calculates and visualizes shadows interactively, depending on shadow-relevant variables. It, therefore, enables interactive exploration — in complete temporal and spatial freedom. Shadow affecting factors are (also refer to Figure 1.7):

- **Light source:** The Sun's position in a scaled-down 3D space.
- **Occluders:** Any non-transparent object, hindering light from passing through, hence, casting shadows. This could be buildings, trees, or terrain in the shape of mountains or hills. Clouds and other weather phenomena are obvious occluders as well. However, since weather is highly dynamic and a vast research area on its own, it is intentionally not integrated into this thesis and its accompanying prototype implementation. It shall also be mentioned, that atmospheric impact on sunlight (e.g., its refraction of low altitude sunlight) is not considered for this work.
- **Receivers:** Any object potentially receiving light. For instance, in a map's context, the ground, a building's roof or facade — naturally including any occluder as well.

Terrain is an example of a single structure that is occluder and receiver at the same time: A mountain casts a shadow on the neighboring valley (*self-shadowing*). In general, any concave structure, lit from a specific angle, is prone to self-shadowing.

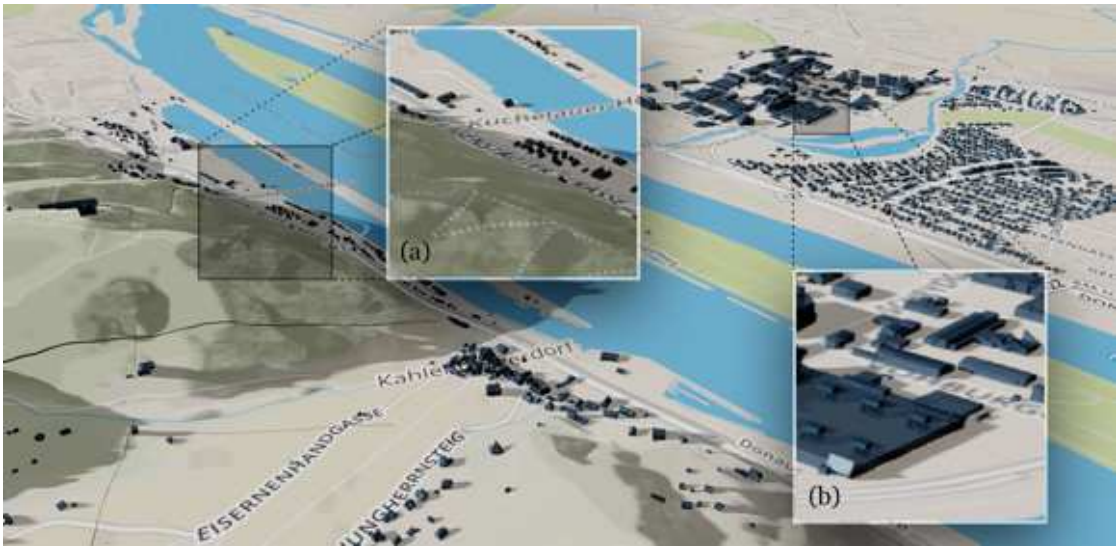


Figure 1.7: Simulated solar shadow scenario at the hilly northern edge of Vienna, including a *1) light source* (= Sun, not directly visible), shedding light on a summer afternoon. Sunlight is occluded by *2) buildings* and *3) terrain*: (a) shows terrain casting shadows on buildings as well as on itself (= *self-shadowing*). (b) depicts a building with vertical structures that cast shadows on itself while the construction also casts shadows on the underlying terrain. Image source: Rendering captured from the prototype that was developed in the course of this research project. © Basemap: Mapbox.com and OSM Contributors, © Elevation data: Mapzen, © 3D building models: Stadt Wien — data.wien.gv.at[a]

[a]If not stated otherwise, all "Prototype screenshots" share these copyrights for underlying data.

It can be argued that, since the data needed to calculate comprehensive solar shadows needs to be three-dimensional, the map might be as well. This does not necessarily mean that the map should allow arbitrary rotations, as this adds a potentially unnecessary layer of complexity to users. It is rather about the circumstance, that reasonable shadow rendering techniques require the full 3D scene (including all shadow occluders and receivers) to work — and this scene might as well just be rendered in the process. It effectively means that shadow visualization comes with little additional computing cost if the map itself is already in 3D. As a side-effect, whenever the use-case benefits from it, the scene can also be tilted: Some of the use-cases mentioned in 1.1.5 benefit from an ability to leave classic top-down perspective and look at walls, windows, and other vertical structures to, as another example, grasp the light situation of a new house.

Accessibility

It is desirable to provide an interactive map to the broadest possible audience. Hence, the implementation of a web application seems the obvious thing to do: The hardware and operating system abstraction, modern browsers deliver, reduce potential incompatibilities

in advance.

Web applications provide fast access, and they don't need to be installed. Their deployment is more or less instantaneous, enabling fast updates and fixes. Not being dependent on *"app store"* infrastructure increases the frequency of update cycles and general freedom.
One of the major advantages, however, is the potential reach: With the same codebase and application, end-user devices spanning from smartphones to tablets to desktop computers are covered in unison. Modern web-programming paradigms, like *responsive design*, provide optimal presentation whatever the actual device might look like.

Data

By downloading only required map data, an initial map clipping is visualized without loading any data that would not be visible in the first place. From there, the user might change the zoom level or pan on the map while required data (i.e., the basemap, 3D buildings and terrain) are downloaded in the background and visualized as soon as the transmission is finished. This allows the app itself to stay light-weight and load accordingly fast. A logical drawback of this approach is the required internet connection and potentially slow updates when there is low bandwidth.

The processed data is heterogeneous and needs to be merged to be visualized together:

- *Basemap*: A two-dimensional visual layer providing geographical and topographical context by displaying features like roads, rivers, trees or buildings. It is usually served as a bitmap or a vector tile. In a solar shadow map scenario, the basemap is mandatory to provide locational awareness.

- *Terrain*: As the basemap itself is two-dimensional, another data source is needed as a base for mountains and valleys. It is conveniently served as tiles that are oriented and aligned according to their respective basemap tiles.

- *Buildings*: 3D structures. A simple version would be extruded 2D floor plans (as used by some applications introduced in 1.1.1). More detailed buildings containing, for example, arbitrary roof structures, need appropriate and processible 3D file formats.

- *Vegetation*: Data regarding plants or trees that are relevant enough to occlude light can either be provided via 3D meshes (similar to buildings) or as metadata. The 3D meshes would then be created procedurally.

- *Clouds*: As mentioned before, clouds, weather and atmospheric influence will not be considered for now. Regarding clouds, there are, in fact, promising data providers for worldwide cloud cover. This all is, nevertheless, part of potential future work.

It is essential to highlight that these data are usually provided in: a) different coordinate systems, b) different file formats, and c) file sizes that might be unacceptable for online (i.e., downloaded) usage. A system to integrate those sources needs to be aware of the fact and pre-process the data in a way that sustains the best possible user experience.

Performance

Web browsers abstract away base hardware: *Graphics processing unit (GPU)* features, for example, are reduced to a subset (= lowest common denominator) of their potential to cover features across different device types. The advantage lies in increased compatibility, the disadvantage in the missed possibility to fully use specific hardware capabilities, hence, reduction of visual quality and performance. Limited access to memory requires limitations on user interaction (e.g., limited viewing angles to reduce 3D map regions to be loaded and rendered), and visualization algorithms need to be adapted and/or optimized. All this is, however, a sacrifice that is willingly accepted to simplify accessibility and broaden distribution potential.

Location awareness

By including mobile devices and their sensors, a solar shadow map would, furthermore, become location-aware, meaning that by accessing *Assisted GPS (A-GPS)* sensors, users would be able to locate themselves on the map — turning the application into a *location based system*.
Positioning on laptops and desktop computers without A-GPS is abstracted over IP-based lookups that usually provide less precise but still functional data.

Location awareness allows users to understand the shadow situation in current surroundings right away. As mentioned in the introductory quote, it would be possible to find the closest sunny spot for a 30-minutes break — without a complicated entry of the current address. The driver of a solar car would be enabled to find the closest parking spot that is sunlit over the next three hours.

It also adds the basis for guided or assisted routing: In a solar shadow context, it would be possible to find a shady path through the city on a hot summer day. A driver of a solar car, again, could drive on roads that resupply some of the consumed energy.

Time-integration

Most Sun-affected usage scenarios do not happen within a temporal instant, but span over a period of time. A period in which the Sun (to be precise: The Earth in relation to the Sun) moves, affecting the direction, area, and shape of cast shadows (see Figure 1.8). It is of potential interest *how* these shadows change: For instance, given a solar car that should charge in the Sun for three hours, it is crucial to know which parking spot actually provides most sunlight during that time frame. Another example would be a person that wants to lay in the shade for a few hours on a hot summer afternoon,

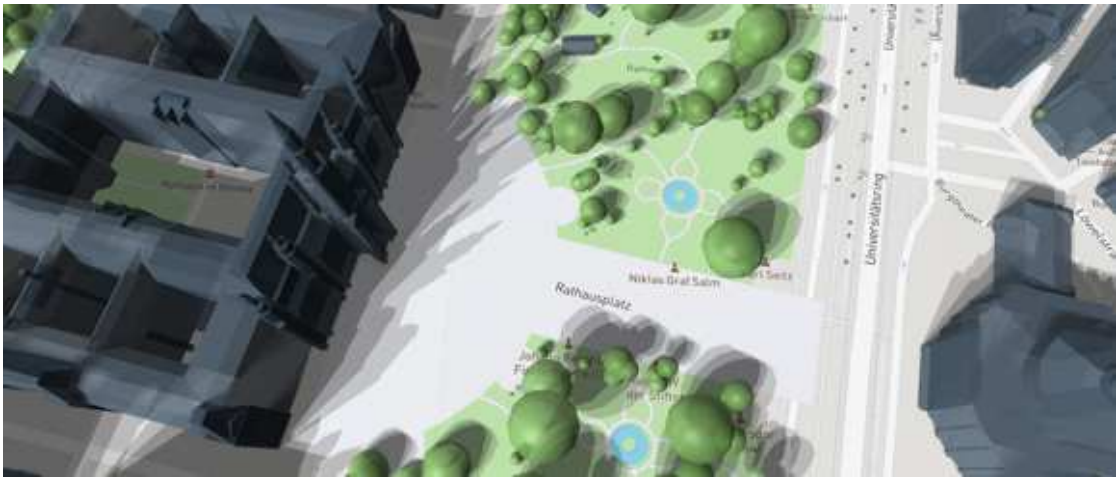


Figure 1.8: Visualization of shadow motion in front of Vienna's town hall on August 8th, 2019 between 14:06 and 15:06 using three stops: $t_0 = 0$ min, $t_1 = 30$ min and $t_2 = 60$ min. Areas that are constantly sunlit during this time frame are apparent. Conveniently, shadow-regions that are always occluded by sunlight appear darker than those only in the shade during either t_0, t_1 or t_2. Image source: Prototype screenshot

looking for the best spot to do that — undisturbed by the Sun.
By integrating over time and, thus, visually accumulating shadows, it is possible to distill time-spanning information into a single picture. This work will investigate a possible solution.

1.2 Problem statement & research questions

Due to the relevance of the Sun on (human) life on Earth, it can be argued that every human should be able to effortlessly and interactively explore and understand solar shadows in cities and around the world in an accessible, practical and visually appealing manner.

The problem space approached by this thesis is, therefore, defined by the lack of a toolkit *enabling* such exploration of solar shadows. This fact leads to insufficient predictability of solar shadows which, in turn, negatively affects scenarios mentioned in Section 1.1.5, "Use-cases", such as within the real estate market, tourism, leisure, or solar cars and their parking — or even just *living* per se.

To improve this situation, the attempt of building the prototype of an "Interactive Web-based 3D Solar Shadow Map" is made. Thus, literature and applications of related approaches shall be researched and evaluated, and technical challenges identified, analyzed, and documented. Conclusively, the implemented prototype shall be tested against the problem space, and its results qualitatively assessed.

The aforementioned problem statement may also be described by the following research questions:

1. **Is it technologically possible to interactively visualize solar shadows caused by terrain, vegetation, and buildings with sufficient precision in a web-based context? Therefore allowing an accessible, visually appealing map user experience?**

   a) **If so, how can the required heterogeneous data be a) retrieved, b) processed, and c) combined in a way that enables this**?

   b) **And how close do simulated urban solar shadow scenarios resemble their real-life counterparts?**

1.3 Aim of the work & methodological approach

The overall aim of this thesis is to provide comprehensible answers to the previously defined research questions. This shall be achieved by evaluating the problem space and its related literature and implementing a technical prototype, which is built upon the knowledge gained in the process. It is, therefore, irrelevant whether the primary research question can be proven right or wrong, as long as the steps leading to its answering are reasonably argued.

The approach looks as follows:

- **Research**
  - Review and evaluation of the state of the art in regards to related research and actual applications (see Chapter 2, *State of the art*)
  - Based on that, decide which higher-level technology should be used to build upon towards a solar shadow map. If there is no suitable higher level adaptable and extensible technology available, consider implementing a prototype from the ground up, maybe supported by an existing low-level 3D engine
  - Limit solar shadow map coverage to a specific region (e.g., the city of Vienna)
  - Research data sources for shadow occluders in this region
  - Research astronomy regarding Sun position for a specific time
- **Implementation & integration** (accompanied by continuous testing, research and documentation):
  - Start implementation of the web-based prototype
  - Integrate data sources
  - Implement solar lighting model

  - Design & implement a User interface (UI)
  - Optimize performance
  - Consider (less performant) mobile devices, adapt/optimize the UI

- **Evaluation**
  - Do the results support the research questions?
    - If not, why?
    - If so, where lies the potential for future improvements?
  - How do the simulated solar shadow scenes qualitatively compare to reality?

1.4 Structure of the work

This thesis is divided into four main chapters:

State of the Art

Chapter 2 covers related literature and analyzes existing approaches, similar or supportive to ours. Furthermore, map- and 3D-engines which — at least in theory — would be capable, or at least adaptable in regards to shadow visualization, are researched and evaluated.

Methodology

The methodological approach, as laid out in 1.3, will be elaborated in detail in Chapter 3. Decisions regarding design and implementation will be argued and described.

Implementation & Reflection

In Chapter 4, the actual implementation of a solar shadow map prototype will be described and reflected on: What is arguably good — where are shortcomings? How could interfaces for *open data* and the data itself, provided by governments and/or city officials, be improved to be beneficial for the presented work? Is the prototype's performance good enough to allow interactive use? And are, at the same time, generated visualizations qualitatively resembling respective real-life scenarios?

Summary & Future Work

The final Chapter 5 will provide, among a digest of the whole work, a motivated outlook covering possible extensions of the solar shadow map, describe potential upcoming use-cases, and suggest further related research areas.

CHAPTER 2

State of the Art

"Understanding has to do with the ability to change perspective."

— Roger Antonsen

The Sun's impact on life on Earth is multidisciplinary, so is related literature. Topics cover a broad range from the historical analog generation of shadow maps, computational shadow rendering methods, and 3D data, as well as the Sun itself. The second section is about existing approaches, among them the highly relevant work on "Shadow accrual maps". However, besides the already preliminarily introduced application attempts (see 1.1.1), even six years later, the number of related tools seems to be relatively sparse.

2.1 Literature

This section begins with Duffield's historic attempt on a seemingly cumbersome — however, still innovative for the time being — technique for topographic *mapping of shadows*. From there, the section will discuss engineering aspects, addressing the advantages and disadvantages of various shadow visualization methods and whether they are appropriate for web application scenarios. Followed by required 3D data, as well as Level of Detail (LOD). The literature section is then closed by a glimpse into relevant astronomy, such as properties of the Sun and its light, to provide a better understanding of how the Sun's dimension and position towards Earth affect solar shadows here on our planet.

2.1.1 History of solar shadow mapping

Besides providing valuable insight into potential application scenarios (refer to Section 1.1.2), Duffield's thesis in geosciences (Duffield, 1975) sheds valuable light on the historical background of solar shadow mapping. He seemed to have coined the term *shadow mapping* as one of the first people in the context of creating maps augmented by solar shadows. This context must not be mixed up with the computer graphics technique termed *shadow mapping* as well (refer to Section 2.1.2) — a pleasant coincidence that

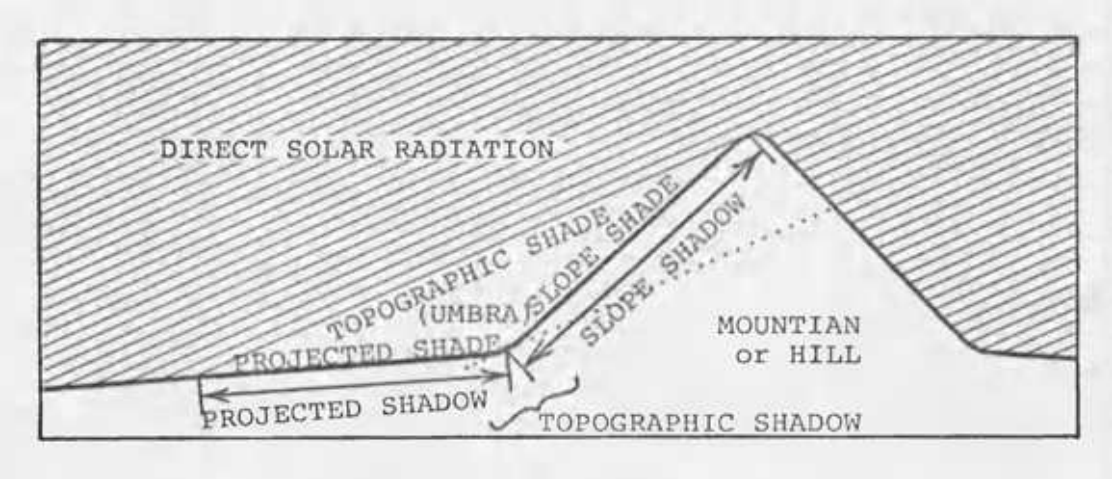


Figure 2.1: Distinction between slope shade and projected shade (Duffield, 1975)

both fields are eventually merged in this thesis.
He remarks that back then, investigations on *insolation* usually ignored topography (i.e., terrain), and were merely considering the Sun's position which affects the angle between incoming sunrays and slopes. If, nevertheless, topography was added to the calculation, shadow effects could be differentiated into *slope shade* and *projected shade* — both combined defined as *topographic shade* (ibid.), as shown in Figure 2.1: *Slope shade* occurs when the surface itself is not illuminated by the Sun, which is a result of the angle between its surface normal and the sunlight direction being greater than 90°. *Projected shade*, on the other hand, is the result of a surface that, given its angle towards the Sun, would be lit, but incoming sunlight is occluded by another surface (*self-shadowing*).

In foggy or cloudy atmospheres, sunlight gets scattered and becomes too diffuse to cause clear shadows on Earth's surface; thus, their simulation becomes less relevant. Duffield (1975), therefore, asserts that "Topographic shading is especially significant where the atmosphere is clear (arid and semiarid lands, alpine regions) or absent (the Moon), where slopes are steep (mountains, craters, escarpments), in middle and high latitudes, and in winter (low Sun angles)." The Moon provides a vivid example of the even stronger impact sunlight has, given there is no interference with an atmosphere: Surface temperature varies dramatically between −173 °C in the shade and 127 °C in the Sun (solarsystem.nasa.gov, 2019).

Projected shade is more complicated to calculate than slope shade: Instead of just considering the angle of a single isolated surface towards the Sun for slope shade, surrounding surfaces need to be taken into account. The radius involving those surfaces becomes relevant, as well as their geometric extension, in particular, their height. Overall, the need for *computation* emerges — Duffield's opinion on computers, however, is that "Computer techniques are convenient if computer time is available, but they require time-consuming (and somewhat tedious) digitization of topographic data." (Duffield,

1975, p.10)– a quote that needs to be put into perspective, as back in 1975, there was no Earth-covering Digital elevation model (DEM) (including Digital terrain model (DTM) and DSM); potential computation required preceding cumbersome digitization of relief data and computers themselves were not nearly as ubiquitous, as today.

Insolation

In his paper *"The Computation of direct insolation on a slope"*, (as referred to in Duffield's work), Ohmura (1968) presents a strategy to integrate insolation over time (refer to 1.1.6, Time-integration) — while also respecting atmospheric transmissivity. For this purpose, tables consisting of total direct insolation within a day for slopes with angles between 0° and 90° as well as azimuths ranging from -90° to +90°, measured from the east (= 0°), were pre-computed.[1] The tables were limited for slopes located at the 45th parallel north (45°N latitude). Within 10° steps in both axis of possible slope angles, time-integrated insolations are listed — covering variants for the two solstices and the equinox as well as different atmospheric transmissivities ranging from 0.50 to 0.85 in steps of 0.05 (see Figure 2.2).

To eventually visualize direct solar insolation, these tables are subsequently used in conjunction with a topographic map (see Figure 2.3a), which provides the necessary slope properties to retrieve daily insolation for arbitrary points on a regular 2D grid. Based on these values, maps with contour lines are created, illustrating the impact of terrain on solar radiation (see Figure 2.3b). In a subsequent paper, Garnier and Ohmura (1969) go in-depth with the method and apply it on the tropical Island of Barbados. Figure 2.4 shows a computer-generated plot where every grid point, represented by a letter *A* up to *U*, encodes a specific Langley (Ly) range, whereas the higher up the letter in the alphabet, the stronger the insolation: A = 125-149 Ly, B = 150-174 Ly, et cetera, in steps of 24 Ly. *Isarithms* (= contour lines) are then drawn *by hand* to improve readability of the visualization further.

These techniques, while still requiring manual finishing touches, are fascinating, even more, given their age: Provided there was access to a computer (which was not a given back then) and already digitized topographic data available, solar radiation visualizations gained informativeness since they were now able to respect specific dates as well as previously ignored aspects like atmospheric transmissivity. It must, however, be emphasized that the presented work completely ignores *shadows* (whether from slopes themselves, buildings, or vegetation) and, therefore, only focusses on insolation based on the angle between a slope and the Sun at a given time.

Attempts on shadowing

Williams et al., who present a computer program with the aim of taking shadowing into account, note that: *"Determination of whether a point is in shadow at any particular*

[1] It is not understandable why slopes oriented towards the west (azimuth < -90° or > +90°) seem to be ignored.

Table of Direct Insolation for Slopes of Different Angle and Azimuth
in Latitude 45°N under Selected Atmospheric Transmissivities

TRANSMISSIVITY= 0.60 SUMMER SOLSTICE

ANGLE OF SLOPE	AZIMUTH OF SLOPE -90	-80	-70	-60	-50	-40	-30	-20	-10	0	10	20	30	40	50	60	70	80	90
0.0	496.8	496.8	496.8	496.8	496.8	496.8	496.8	496.8	496.8	496.8	496.8	496.8	496.8	496.8	496.8	496.8	496.8	496.8	496.8
10.0	469.5	469.8	470.7	472.2	474.1	476.6	479.4	482.5	485.8	489.3	492.7	496.0	499.1	502.0	504.4	506.3	507.8	508.7	509.0
20.0	428.0	428.6	430.3	433.2	437.1	441.9	447.5	453.9	460.9	468.0	474.9	481.5	487.6	492.9	497.4	501.0	503.6	505.2	505.7
30.0	373.4	374.3	376.9	381.1	386.7	394.2	403.7	414.5	425.8	436.8	447.2	456.6	464.8	471.7	477.4	481.8	484.9	486.7	487.4
40.0	307.6	308.7	312.0	317.3	325.1	337.8	353.2	369.4	385.1	399.6	412.6	423.6	433.0	440.3	445.8	449.9	452.6	454.1	454.6
50.0	232.3	233.7	237.6	244.0	259.2	280.2	301.9	322.8	341.8	358.6	372.6	384.4	393.4	399.9	404.4	407.1	408.4	408.6	408.8
60.0	150.0	151.5	156.0	173.3	200.8	227.9	253.6	277.1	297.7	315.2	329.2	340.1	347.7	352.3	354.4	354.5	353.5	352.0	351.4
70.0	66.7	73.2	94.2	124.1	154.2	183.1	209.6	233.3	253.8	270.4	283.2	292.2	297.3	299.0	297.8	294.6	290.2	286.3	284.7
80.0	35.1	42.6	63.1	90.1	117.9	144.8	169.9	192.0	210.7	225.3	236.0	242.3	244.3	242.5	237.3	230.0	221.3	214.0	211.3
90.0	21.1	27.2	43.5	65.5	89.0	112.3	134.1	153.3	169.2	181.1	189.0	192.2	190.9	185.3	176.9	165.9	150.2	139.1	134.7

TRANSMISSIVITY= 0.80 WINTER SOLSTICE

ANGLE OF SLOPE	AZIMUTH OF SLOPE -90	-80	-70	-60	-50	-40	-30	-20	-10	0	10	20	30	40	50	60	70	80	90
0.0	109.4	109.4	109.4	109.4	109.4	109.4	109.4	109.4	109.4	109.4	109.4	109.4	109.4	109.4	109.4	109.4	109.4	109.4	109.4
10.0	52.6	53.5	56.1	60.3	65.9	72.8	80.6	89.3	98.4	107.9	117.4	126.7	135.4	143.3	150.1	155.6	159.7	162.2	163.0
20.0	3.1	4.7	9.8	18.2	28.8	41.2	55.3	70.8	87.5	105.0	122.8	140.5	157.4	172.8	186.3	197.2	205.2	210.1	211.8
30.0	0.0	0.0	0.0	0.0	7.8	21.1	37.7	57.1	78.6	101.9	126.3	150.9	174.8	197.2	216.8	232.7	244.5	251.6	254.1
40.0	0.0	0.0	0.0	0.0	1.5	11.1	26.9	47.3	71.4	98.4	127.4	157.4	187.2	215.6	240.7	261.2	276.3	285.5	288.6
50.0	0.0	0.0	0.0	0.0	0.3	6.5	20.2	40.0	65.0	94.1	126.1	159.9	194.4	227.5	257.3	281.7	299.7	310.7	314.5
60.0	0.0	0.0	0.0	0.0	0.1	4.3	15.8	34.4	59.0	88.6	122.2	158.3	195.7	232.5	266.1	293.7	314.0	326.5	330.7
70.0	0.0	0.0	0.0	0.0	0.1	3.0	12.5	29.4	52.8	81.9	115.6	152.6	191.6	230.7	266.8	296.8	318.8	332.3	336.9
80.0	0.0	0.0	0.0	0.0	0.0	2.1	10.0	24.9	46.4	73.9	106.4	142.9	181.9	221.8	259.4	290.8	313.9	328.1	332.9
90.0	0.0	0.0	0.0	0.0	0.0	1.5	7.9	20.6	39.6	64.7	95.0	129.5	167.3	206.4	244.2	276.0	299.5	313.9	318.7

TRANSMISSIVITY= 0.75 EQUINOXES

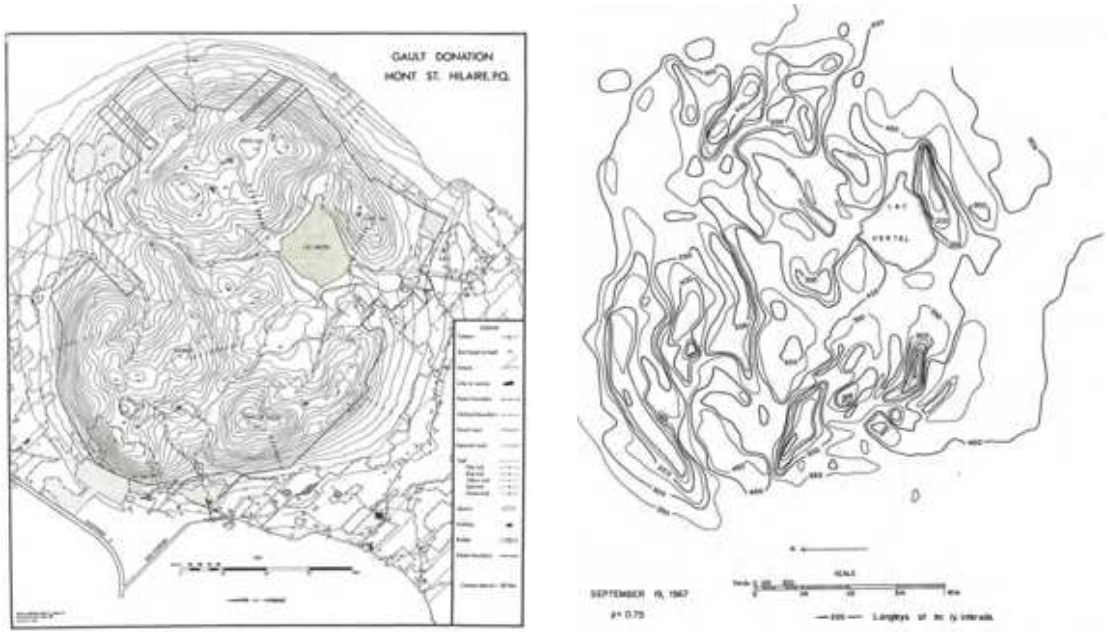
ANGLE OF SLOPE	AZIMUTH OF SLOPE -90	-80	-70	-60	-50	-40	-30	-20	-10	0	10	20	30	40	50	60	70	80	90
0.0	369.6	369.6	369.6	369.6	369.6	369.6	369.6	369.6	369.6	369.6	369.6	369.6	369.6	369.6	369.6	369.6	369.6	369.6	369.6
10.0	299.8	300.7	303.6	308.4	314.8	322.8	332.1	342.3	353.1	364.2	375.4	386.1	396.2	405.3	413.2	419.6	424.3	427.2	428.2
20.0	220.8	222.8	228.7	238.6	252.3	269.0	287.9	308.3	329.6	351.2	372.4	393.0	412.1	429.5	444.5	457.0	466.2	471.9	473.8
30.0	135.2	138.3	149.1	166.9	189.7	215.7	244.0	273.7	304.0	334.0	363.6	391.8	418.2	442.1	463.2	480.7	493.9	502.2	505.0
40.0	45.4	54.4	77.1	105.1	136.3	170.0	205.2	241.5	277.9	314.0	349.3	383.0	414.6	443.4	468.8	490.3	506.8	517.3	520.8
50.0	0.0	9.3	33.6	63.8	97.5	134.0	172.4	212.0	251.9	291.4	330.0	366.9	401.5	433.2	461.4	485.5	504.1	516.6	520.9
60.0	0.0	2.4	16.4	40.7	71.6	106.8	145.1	184.8	225.6	266.2	305.7	343.5	379.2	412.0	441.2	466.5	486.8	500.2	505.1
70.0	0.0	1.0	9.3	27.6	53.7	85.5	121.4	159.5	198.9	238.2	276.8	313.8	348.2	380.4	409.1	434.0	454.5	468.7	473.9
80.0	0.0	0.5	6.0	19.5	40.9	68.4	100.2	135.3	171.3	208.1	243.9	278.2	310.2	339.7	366.2	389.4	409.0	423.0	428.4
90.0	0.0	0.3	3.9	13.9	30.8	53.8	81.4	111.7	144.0	176.1	208.2	238.2	266.3	291.5	314.1	334.2	351.4	364.5	369.8

Figure 2.2: Extract of pre-computed tables covering daily insolation for hills of various orientation (Ohmura, 1968)

time is of critical importance in mountainous terrain and an algorithm for a solution to this problem was of major concern. No specific formulation of the problem has been noted in the literature." (Williams et al., 1972, p.528) They propose to consider terrain elevations high enough to occlude sunlight for a given region by interpolating grid point elevations along the line towards the Sun's azimuth and comparing it with the Sun's altitude. If the latter is too low, that region is in the shade and vice versa. Generated images neglect visualization of shadows per se: Shaded areas are only taken into account to reduce overall insolation, visualized using contour lines. As a result, the hand traced maps look similar to Ohmura's (Figure 2.3b).

Architecture & "Sun Machines"

Shadows cast by buildings have been neglected so far. Thus, Duffield (1975) describes architects' attempts to understand sunlight within their profession: So-called *Sun Machines* helped to simulate light situations on architectural models for arbitrary Sun positions.



(a) Contour map of Mont St. Hilaire as the source for slope properties

(b) Total solar insolation on September 19th, 1967. Atmospheric transmissivity p = 0.75

Figure 2.3: Mont St. Hilaire, Quebec, is used as an example to demonstrate solar insolation visualization using contour lines (b) based on a topographic map (a) (Ohmura, 1968)

Duffield (1975) differentiates them into the following three types:

1. Machines in which the model is fixed while the light moves.
2. Devices where the light is fixed and the model can move.
3. And those, where both, light as well as the model, can move (also called *Heliodons*, see Figure 2.5a)

A design by Knowles (1974) is remarkable insofar as it exchanges the light source of a *Heliodon* with a camera to later manually specify which surfaces are lit, as well as their degree of illumination. This approach shares interesting similarities to Williams's *Shadow Mapping* (also refer to Section 2.1.2).

Machines that support a movable light source (*1. and 3.*) prohibit the emulation of the Sun's roughly parallel rays (from the Earth as a reference point), which in turn are the reason for nearly parallel shadows on Earth: Compared to the Sun-Earth distance, light sources on these machines are just too close to the model for their rays to be even remotely parallel (see Figure 2.5b and refer to Section 2.1.5). Thus, the logical strategy to achieve Sun-like lighting — including nearly parallel shadows — is to rely on the Sun

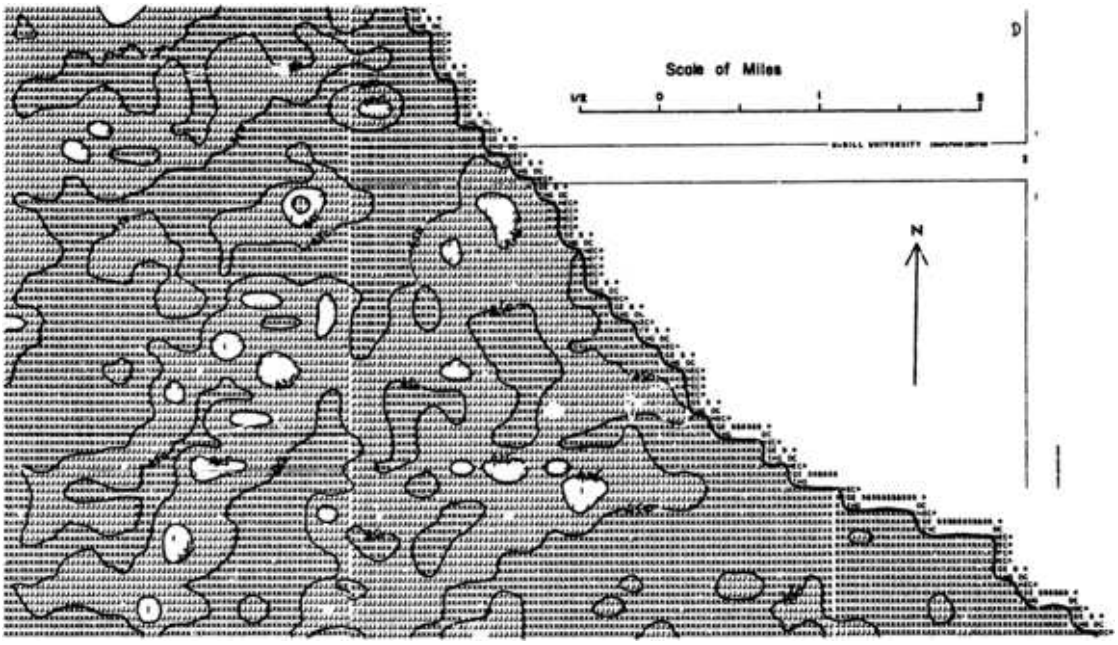


Figure 2.4: Computer generated plot of a part of Barbados. Every grid point encodes a range of solar insolation, providing the basis for hand-drawn isarithms (= contour lines) (Garnier and Ohmura, 1969)

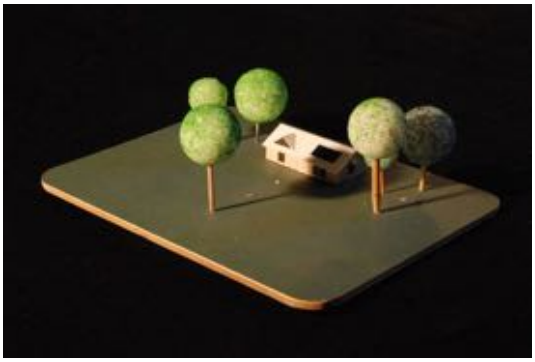
itself. Since the Sun cannot be moved arbitrarily (compare to item *2.* in the list above), the model has to: It is, together with a *sundial*, fixed on a platform that allows for precise alignment according to a desired date and time.

Duffield's shadow mapping technique

To visualize shadow situations for terrain, Wise (1969) chose an approach related to *Sun machines*: However, instead of architectural visualization using building structures, he exchanged them with topographic plastic relief maps. He illuminated them with from desired angles and photographed the scenery from above. But since his method required an artificial light source, it shared the disadvantage of diverging light rays, and, therefore, shadows.

Duffield, eventually, tried to compensate this shortcoming by combining the work of Wise (1969) with a sundial-augmented platform, enabling the machine to use the Sun as the light source (see Figure 2.6). He describes his machine as "a powerful analog computing method" (Duffield, 1975) and refers to relief maps as storage devices for topographic information — ideal for "rapid shadow processing and retrieval merely, by parallel illumination from the desired angle" (ibid.). From a modern perspective, his descriptions are inspiring: The physical "hardware" he describes, could just be abstracted away by virtual objects, putting the whole machine into a computer graphics context.

(a) A heliodon. Photo: © betanit.com



(b) Exemplary lighting simulation achieved by a heliodon. Photo: Norbert Lechner (2019)

Figure 2.5: The heliodon shown in (a) works as follows: Manual rotation of the outer ring-structure sets the model's latitude, while a given month can be simulated by switching between each rings' light sources (thus, enabling a fast switch and comparison between light scenarios in different months). Conclusively, by rotating the active ring, time is simulated as the light follows the Sun's natural path

In fact, computer-based shadow rendering approaches (as introduced in Section 2.1.2), are not too far off from Duffield's very physical device within their intrinsic operating principle. He, furthermore, concludes that: *"Tens of thousands of intricate angles, distances, and elevations solve themselves instantly, as in the real landscape, creating shadow patterns without recourse to brute-force digital computation. The new technique represents utmost simplicity and efficiency."*[2] (Duffield, 1975, p.15)

Operation of Duffield's machine, data sources, and accuracy

The machine works by setting up the scene accordingly first, which means rotating the platform containing the relief map in a way that the current sunlight inclination aligns with the desired date and time of the visualization. The process continues by taking a photograph of the map from above, whereas "from above" is defined by the inverse direction of the platform's normal. Its negative is later projected onto the flat basemap of the same region where shadow boundaries are, eventually, manually traced to achieve a visualizations as shown in Figure 2.7.

[2] Which nicely resonates with philosophies about the physical universe itself being nothing else, but an enormous quantum computer anyways...

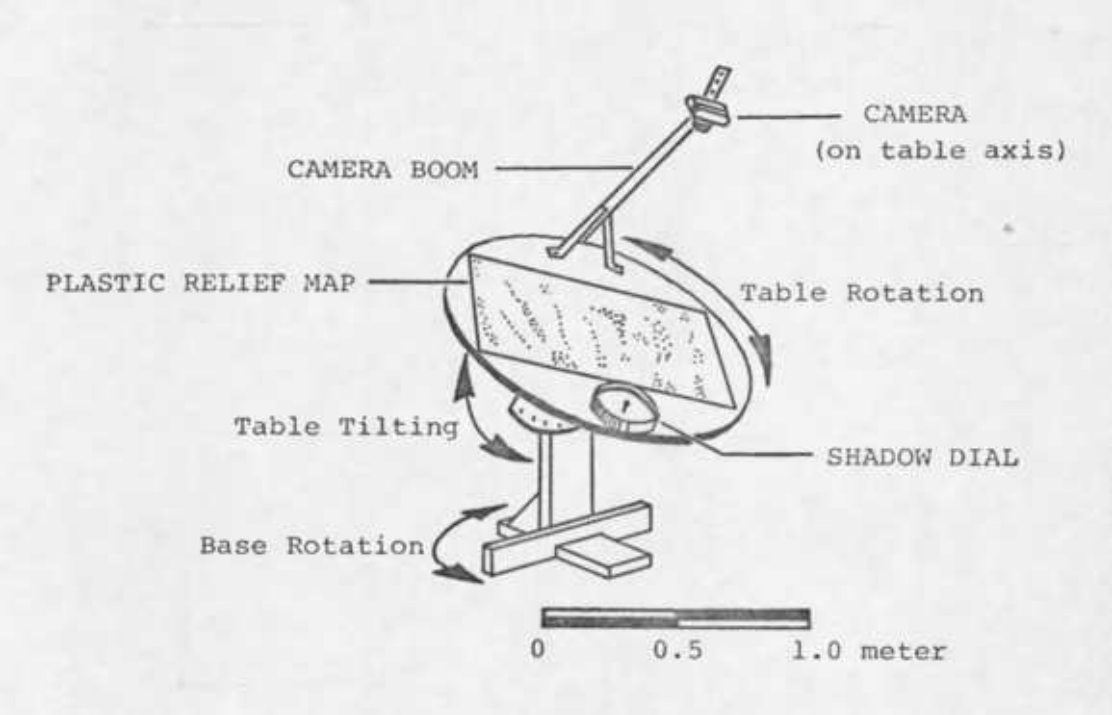
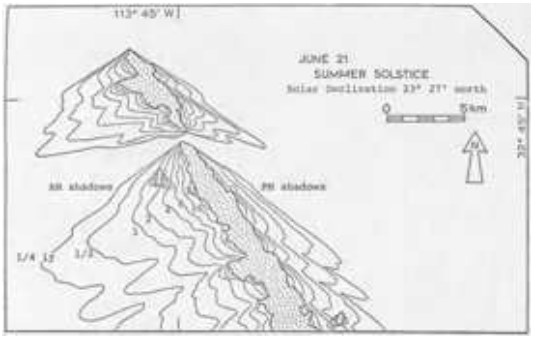


Figure 2.6: The solar shadow mapping machine designed by Duffield (1975)

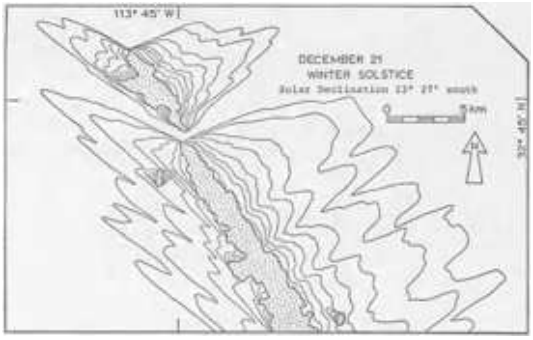
Regarding accessibility of relief maps, which are the machine's primary data source, Duffield mentions that most of the United States of America's topographic information has been archived via plastic relief maps at 1:250,000 scale by the U.S. Army Map Service — "and much of the rest of the world at various scales" (Duffield, 1975, p.15). The visualization also must not just be limited to topography, as he argues that even shrubs, trees or clouds could be mapped, provided there was "a scale model of the object" (Duffield, 1975, p.21).

Throughout the shadow mapping process, there are various stages where inaccuracies occur:

- Since relief maps are downsized models of the reality, a degree of inaccuracy is inherent. Duffield (1975) states that despite repeated requests to manufacturers, no feedback regarding tolerances was supplied. Furthermore, storage and handling wear the models over time (ibid.).
- Model mounting and platform orientation towards the desired Sun inclination is by nature always an approximation. Duffield (1975), however, limits the cumulative angle error in this regard to a maximum of 1°.
- Finally, during the shadow photography, its projection onto the basemap as well as the manual tracing of the boundaries.

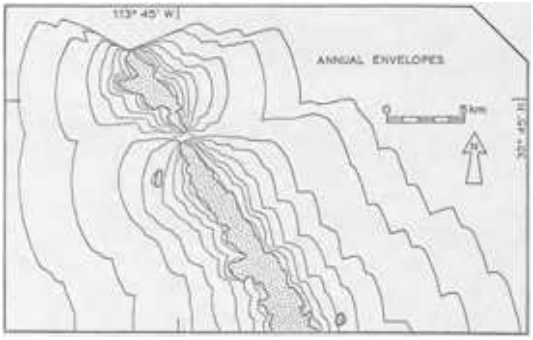


(a) Summer solstice, June 21st. Outermost shadow boundaries depict 1/4 Ly, next inner ones 1/2 Ly, et cetera



(b) In contrast to (a), this visualization shows shadow boundaries of the winter solstice on December 21st

(c) The juxtaposition between model illumination (Duffield's approach) and a graphical reconstruction of the shadow situation shows inaccuracies.



(d) Shadow boundaries for specific Ly-values, as described in (a), are enveloped over their motion from north to south over the year. A strategy related to *time-integration* (see Section 1.1.6)

Figure 2.7: A selection of solar shadow maps created with Duffield's apparatus for Mohawk Mountains, Yuma County, Arizona — 32°45' N 113°45' W (Duffield, 1975)

Error analysis for all aspects besides model mounting and platform orientation was omitted. Figure 2.7c, however, shows a graphical juxtaposition between Duffield's visualization and a "precise but extremely time-consuming graphical construction from a topographic map" (Duffield, 1975, p.29), using a technique related to one presented by Garnett (1935). Duffield (1975) concludes that his technique produces a good approximation of reality, whereas visible systematic deviations for the morning shadows were caused by imprecise model orientation (ibid.). The relief map tends to smoothen the topography, and "major peaks and saddles are slightly displaced" (ibid.).

Discussion

It is fascinating to see the urges and applications that led to Duffield's apparatus and all the other work presented in this section. The parallels to the present in regards to quality of data, data-sources, as well as the quality, flexibility and extensibility of the visualization and its possibilities, are remarkable. The idea of Knowles (1974), to mount a camera instead of a light source onto his "Sun Machine", in order to understand the impact of light on a physically modeled 3D scene, is nothing but a beautiful foresight to the work of Williams (1978) who invented the computer graphics algorithm of "shadow mapping" by rendering a virtual 3D scene from the light's perspective — just four years later. This peek into history shows the analogies between reality itself, a rarely computerized world back then, and modern, computer-driven simulation.

Applications of solar shadow maps around the 1970ies commendably explored and documented by Duffield, show the broad range of use-cases and scenarios that benefit from a better understanding of Sun and shade. Most, if not all of them, are still valid in the present day. The fact that Sun became a much more relevant factor in the last decades (not only in regards to energy consumption, see Figure 1.3, but also considering the urgent issue of climate change), provides reason and motivation to investigate the issue and enhance it with state of the art technology.

Even though Duffield's device is cheap and straightforward to build — "for less than $20" (Duffield, 1975, p.17) — one cannot ignore that the creation, access, archiving, and sharing of relief maps are cumbersome and limiting. On top, the necessity to manually trace shadow boundaries does not scale well either. It must, conclusively, be noted that this perception obviously originates from around 45 years later, where technological progress led to most of the planet's surface being accessible over the internet for almost no cost, while portable computers can process that data in realtime right in our hands (as it will be demonstrated later in this thesis).

2.1.2 Shadow rendering

Instead of applying physical models onto manual apparatus, such as those introduced at the beginning of this chapter (refer to 2.1.1), shadow visualizations may as well be entirely computer-generated. As mentioned before, computers aided the creation of solar shadow maps already in the 1970ies (Garnier and Ohmura, 1969; Ohmura, 1968; Williams et al., 1972). They allowed more complex calculations and considerations of factors that, beforehand, had to be neglected, such as atmospherical properties. Even though computation time was a rare good (Duffield, 1975), their potential has probably been recognizable by that time, and ever-cheaper technology during the following years paved the way for further utilization. Outside the scope of geosciences, pioneers of computer graphics started to realize that the lack of shadows in computer-generated scenes hurts comprehensibility and realism of rendered images (Crow, 1977) and so began research in a mathematics and computation backed field.

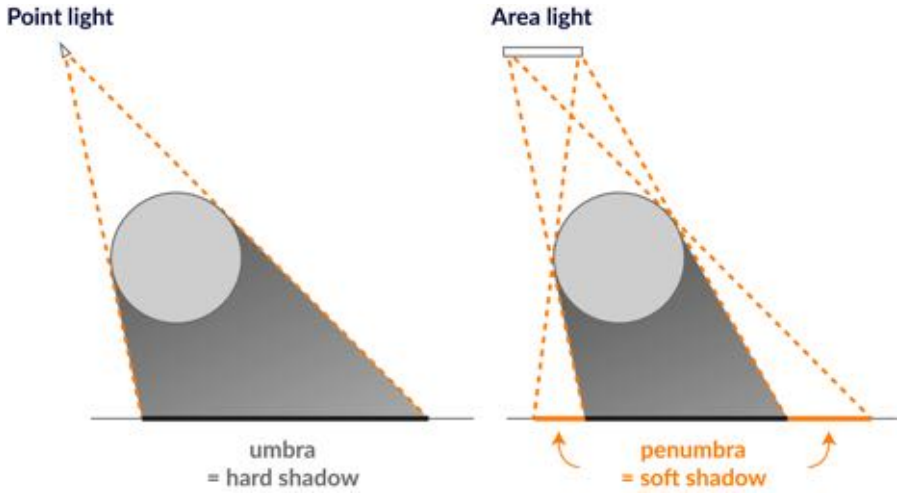


Figure 2.8: Light rays originating from a one-dimensional point (i.e., point light) can only produce hard shadows (left) since a surface can either be lit, or not. Area light sources, however, create an additional penumbra that manifests on surfaces that are only partially occluded (right): The more area of the light is visible from a single surface point, the lighter and vice versa

Light source

As there would be no shadow, if there was no light, a computer graphical abstraction of the latter is needed. Light emitted from physical light sources — such as the Sun — usually originates from an area, rather than a one-dimensional point. As soon as there is an area light source, nevertheless, cast shadows consist of not only a sharp delineated *umbra*, but also a *penumbra* (see Figure 2.8), which makes algorithms significantly more complex and computation more expensive. While there are modern variants of shadow rendering techniques that support area light sources as well, only point light sources, or directional lights (i.e., a light source infinitely far away, emitting parallel rays) are considered for now as they inherently produce hard shadows. Considering solar shadow maps, however, an obvious argument against this simplification would be the apparent (enormous) volume of the Sun: Its light is definitely emitted by an area. Given its distance to the Earth and its diameter, though, that fraction of sunrays hitting the Earth are almost parallel, and the — still existent but negligible — divergence is often neglected (refer to 2.1.5 for further discussion).

Ray tracing and other early shadow rendering approaches

According to Zobrist and Sabharwal (1992) and Foley et al. (1996) (among others), the first paper on *ray tracing* was published by Appel (1968)[3], which demonstrated how

[3]His method was later termed *ray casting*, as it lacked the characteristic recursive tracing of rays, *ray tracing* subsequently became associated with.

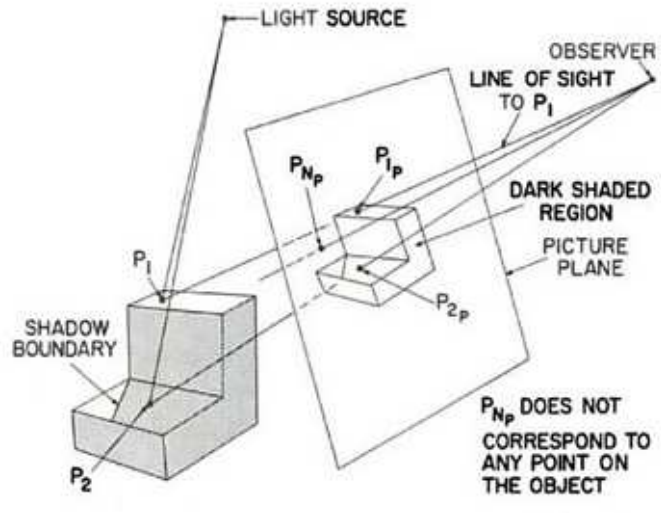
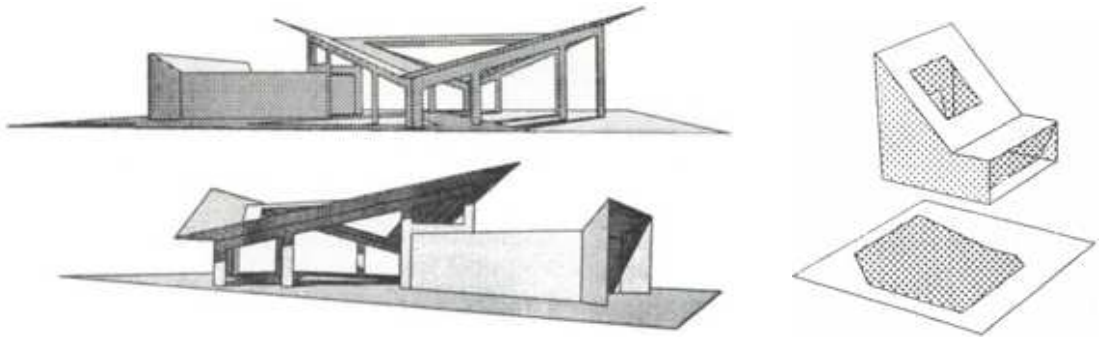


Figure 2.9: Shadow rendering through ray casting, illustrated by Appel (1968). P_{2_P} on the image plane is in shadow since a ray cast from P_2 towards the light source cannot reach it. P_1's secondary ray, however, reaches the light source, and P_{1_P} is, therefore, lit.

shadows can be integrated into arbitrary 3D scenes consisting of flat surfaces. The simple algorithm works as follows (also refer to Figure 2.9 for visual explanation):

- For every pixel of the image to be generated, a ray is sent from the observer's point (camera) through the picture plane onto the scene.
- The first intersection of a ray with an object (P_1, P_2) in the scene defines the color value of that pixel (based on the object's color), which might be subject to shading depending on the surface's normal and other factors.
- To determine, whether that point of intersection is in the shade or not, another ray is cast from the intersection towards the light source (or sourc*es*, if there are more): If these secondary rays cannot hit any light source (they are blocked by the same or another object), that part of the image (= pixel) must be in shadow and is colored accordingly (P_{1_P}, P_{2_P}).

Within his work, Appel wanted to evaluate the quality of generated images, as well as their required computation time. He concluded that the algorithm described above, capable of producing pictures as shown in Figure 2.10a, was — while simple to implement and low on memory consumption — "very time consuming, usually requiring for useful results several thousand times as much calculation time as a wire frame drawing" (Appel, 1968, p.4). By introducing *cutting planes* into the method and, therefore, rendering the image scanline after scanline (i.e., the intersection of the cutting with the picture plane creates the scanline), however, the shading of adjacent pixels could be significantly accelerated:



(a) Architectural scene (b) Toning

Figure 2.10: Images generated with Appel's approach, printed on digital plotters: A building structure consisting of flat surfaces shown from different perspectives in (a) demonstrates the qualitative capabilities. Calculation time for each view was about 30 minutes. For monochrome plotters available at the time, it was discovered that size-varying "+"-signs were preferable for grayscale shading, as shown in (b) (Appel, 1968)

Computing time reduced to less than the square root of the original approach (ibid.). A general advantage of ray tracing approaches is the inherent support for analytically defined surfaces, i.e., intersection points can be calculated to the highest precision, since no numerical approximation/interpolation is necessary. Considering solar shadow maps, this is, nevertheless, an irrelevant advantage, as virtual surfaces to be rendered are intrinsically non-analytical, rather quantized approximations of reality.

Besides leaving a marginal note (and thereby again leaving the pure computer graphical context) on the importance of shadow visualization "for evaluating the need for airconditioning or the availability of solar energy" (Crow, 1977, p.242), Crow introduced a new class of shadow rendering approaches: Projected shadow polygons, also referred to as *shadow volumes*. The technique's key concept is to create a three-dimensional polygonal reproduction of a point light's umbra (see 2.8) and precisely calculate shadowed areas by intersecting that volume with the scene — or as Crow puts it: *"Shadows may be defined by the projection of edges onto surfaces [...] or they may be defined by the volume of space they encompass."* (Crow, 1977, p.246) Any point inside the umbra's volume lies in shadow; any point outside is lit. A general algorithm works as follows:

- Silhouette edges of an occluder object are defined as those edges that connect forward-facing faces (i.e., their normal is directed towards the light source) with backward-facing ones. Those need to be found (see Figure 2.11c).
- Create new edges (i.e., additional geometry), based on vertices of the silhouette edges, by extending them away from the light source, following the direction of the point light source. Thus, a cone-like shape is formed (see Figure 2.11a).

- That cone's length can be defined arbitrarily (the longer, the farther the shadow's reach) — forming the very volume, objects are tested against (i.e., whether they lie inside the volume, partially or not at all — see Figure 2.11b).

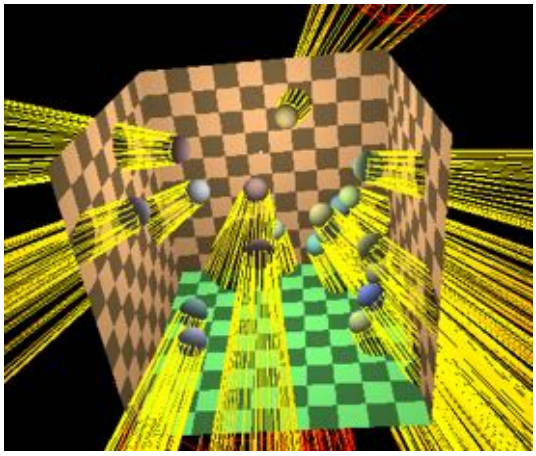
In his paper, Crow (1977) compared his invention with different classes of shadowing algorithms (among them, the scanline variant of the aforementioned *ray casting* approach) and evaluated them in terms of memory consumption, computation efficiency, as well as the difficulty of additional implementation (compared to a non-shadow variant). He concluded that overall, shadow volumes have the most appealing characteristics as they only require "somewhat more" (Crow, 1977, p.248) additional storage and cause the least increase in computation, as only polygon silhouettes are considered for the shadow volume calculation. As of additional implementation expenses, Crow concludes that as soon as there is already an algorithm to scan hidden-surfaces in place (which is a significant part in most of the compared approaches), shadow volumes are the most economical solution.

One needs to be aware that all implementations presented in this chapter to this point, were far from real-time shadow rendering in terms of speed. It took 12 years until Heidmann (1989) demonstrated a version of Crow's approach that utilized a GPU's stencil buffer that was eventually fast enough. However, shadow volumes have the inherent drawback of requiring additional geometry — which, in turn, consumes rasterization time on the GPU, therefore, negatively affecting overall performance. This worsens as the scenery gains detail: "Round" (i.e., smoother curves by approximating them with even more polygons) occluders result in drastically more complex shadow volumes as well, and non-planar polygonal structures (e.g., curved surfaces) are incompatible with the approach in the first place.

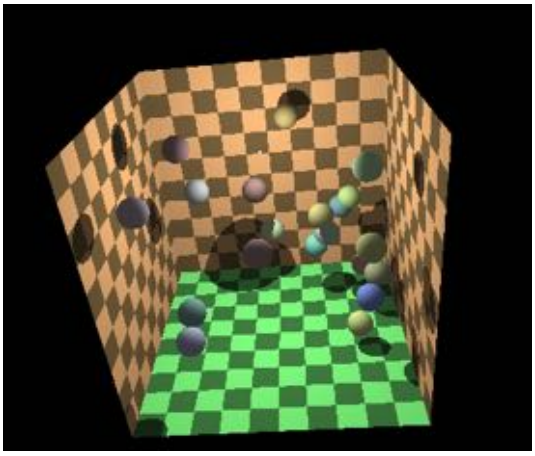
Shadow mapping

First and foremost, there are some interesting parallels: *Shadow mapping* is a term that was used frequently in this thesis by now. In Duffield's work, *"Solar shadow maps"*, he augmented topographic maps with shadow outlines; in other words, he *mapped shadows*. Ironically, within the per se unrelated field of computer graphics, a widely known shadow rendering approach is termed *shadow mapping* as well. Thus, taking the liberty of a little humour here, it might just be the logical conclusion to use *shadow mapping* to *map shadows* on ... *maps*. It is not just the definition of terms, another analogy between Knowles's work (i.e., exchanging the light source of a "Shadow Machine" with a camera in order to enable visual analysis of the generated image in regards to illumination), and the computer graphics technique invented by Williams (1978) was already outlined before (refer to *Architecture & "Sun Machines"*, in Section 2.1.1).[4]

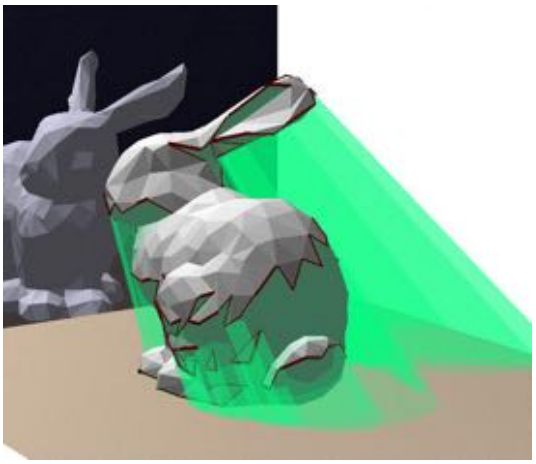
[4]Another delightful similarity lies in the names of Lance Williams (1978) and L. D. Williams et al. (1972): L. D. Williams was among the first ones who tried to enhance research on solar shadows in topographical contexts with the help of computation (refer to Section 2.1.1), and since both authors' names start with an "L" and had a scientific track record around the same time, the suspicion substantiated,



(a) A point light source in the center of the scene is the reference point for generated shadow volumes (depicted as yellow wireframe). Cone-like shapes are extruded inverse to the light source direction, through the occluder's silhouettes



(b) Intersection areas of shadow volumes shown in Figure (a) and occludees (= walls) are rendered as hard shadows



(c) Example of a more complex occluder. The red line on the bunny's body is the silhouette-edge from which the conical shadow volume is extruded from (green)

Figure 2.11: Various shadow volume examples. Image sources (a)/(b): "Shadow volume illustration" by *Rainwarrior*, Wikimedia Commons, CC BY-SA 3.0, (c): © Zach Lynn, zachlynn.com

Williams's approach utilizes the work of Straßer (1974) who, for the first time, outlined the functionality of a so-called *z-buffer* within the "future work" chapter of his PhD thesis (page 6-1, ibid.): *"[...] Dies kann geleistet werden, wenn man für jeden Punkt auch die Z-Koordinate oder den Intensitätswert abspeichert."* — which basically describes the process of storing the z-coordinate of every (visible) point. The z-buffer allowed Williams (1978) to construct an efficient algorithm based on the premise that "a scene rendered with shadows contains two views in one image. If we are content to cast shadows on a single wall or ground plane, these two views are simple projections." (Williams, 1978, p.271). The technique can be described by the following steps (see Figure 2.12a for visual reference):

1. Initially, the scene is rendered from the light source's view (here is the parallel to Williams et al.'s "Sun Machine"), and *depth values* for every pixel are obtained. Which means that the distance from every pixel towards the light source is encoded into a grayscale image that resides within the z-buffer (Figure 2.12a, center image).

2. Afterwards, the scene is rendered from the observer's view, and this is where ingenuity happens: Through a linear transformation, every pixel from the observer's view (due to the z-buffer, those pixels have depth values as well, hence, not only X and Y but also Z (= depth) coordinates) is transformed into the light source's view.

3. Now it only remains to be tested, whether the transformed pixel is *closer* to the light source, than the *depth value* obtained for that position before in *step 1.* In other words: Whether it is visible (Figure 2.12a, right image).

4. If the transformed pixel is visible, it must be lit. Vice-versa, if not, it must be in shadow. In both cases, the final image's pixel is shaded accordingly.

All this occurs as a *post-process* in *image space*, which means that pixel transformations happen after the scene has been rendered. Not only are the depth values of the final pixels now quantized by the z-buffer (which, in case of Williams (1978), had a resolution of 16 bits, resulting in just 65,536 different values), it also means that quantized pixels (i.e., rasterized squares) are reprojected again into a different view: Both, the reduction in depth-information as well as the rasterization plus reprojection, compulsorily lead to aliasing effects, whereas their intensity can be compensated through various strategies, such as a larger texture-size for the depth map (other approaches will be described in the following). Williams concludes that "quantization and aliasing are the chief drawbacks of image space algorithms." (Williams, 1978, p.271)

that they were, in fact, the same person; researching the topic of shadows among completely diverse areas, i.e., computer graphics and climatology. Minor investigation concluded, however, that L. D. Williams was from the Institute of Arctic and Alpine Research, University of Colorado and even though the name behind the first "L" could not be revealed, his career seems too decoupled from Lance Williams', who studied computer science at the University of Utah (again, a neighboring state of Colorado), and had a tremendously successful career in computer graphics, contributing some of the most important research in its field.

(a) The basic principle of shadow mapping: Depth map (middle) is rendered from the light's perspective (left) to allow depth comparison (right), defining which pixels are lit vs. in the shade.

(b) Shadow map size 256x256: Jagged outlines

(c) 2048x2048 creates smooth results

(d) "Shadow acne" visible on the white surface (self-shadowing where none should be)

(e) Disappearing shadows due to misaligned light frustum

Figure 2.12: Shadow mapping approach and visual drawbacks. Image sources: Redway3d Documentation (2019), © Redway3D

The underlying nature of *image space* algorithms, on the other hand, is foundation for their most significant advantage: Their performance is inherently decoupled from the complexity of the scene. Since *shadow mapping* works with the "final" image, its expense is only dependent on that image's resolution, which further means that it handles curved surfaces without additional cost (Williams, 1978). This stands in stark contrast to *shadow volume* or other added-geometry based shadow approaches, whose performance suffers significantly from increased complexity. Shadow mapping became even faster when Segal et al. (1992) showed how to exploit (hardware accelerated) *texture mapping* to further speed up the projection transformations.

Improving shadow mapping's visual quality

Various strategies have been developed to compensate obvious shortcomings concerning the visual quality of shadow mapping. Some major problems as well as techniques to counteract, are presented as follows:

- **Aliasing ("jagged shadows"):** As shadows are based on textures, their quality correlates with the texture size of the depth map. Pixels of two rasterized images are depth tested against each other after being transformed into the same space, and this rasterization is clearly visible if texture size is too low (Figure 2.12b).

  Solutions:

  - **Larger textures:** An increase of the shadow map's (depth map) texture size produces smoother shadows (Figure 2.12c) as the resolution of quantized pixels being reprojected between different views, increases. Larger textures, however, require more memory, which is constrained depending on available hardware, and are generally more expensive to render.
  - **View frustum:** Adaptation of the light source's view frustum according to the observer's view frustum allows more economical use of the shadow map: Parts of the scene which either don't need to be lit or are not visible to the observer anyway, do not need to be stored in a depth map. Its effective resolution is therefore, increased. If, however, the light's adapted view frustum is too small or misaligned, shadows might disappear where they, in fact, should be (Figure 2.12e). Furthermore, as soon as the observer moves — which is common and wanted in interactive 3D scenes — the adaptation of the light's view frustum causes shadow edges to *shimmer*, since the constellation between observer and light changes slightly, triggering recalculations of projections. An attempt to automatically approximate the observer's view frustum using a trapezoid, as seen from the light, as well as fix said continuity problem, is presented by Martin and Tan (2004).
  - **Filtering:** An intuitive guess would be to filter shadow maps just as it is commonly done with image textures (e.g., bilinear filtering leads to "blurred" textures from close observation): This, however, won't work since, even if

the depth maps were filtered, the final decision, whether a pixel is in shadow or not, is always *binary*. Hence, there would be hard and potentially jagged edges again. Even worse, if the depth map was filtered, depth values along (hard) edges of objects would be smoothed out as well, resulting in a depth representation far off the original scene (Reeves et al., 1987).
There are, nevertheless, numerous approaches to shadow map filtering, and it is the subject to ongoing research. They vary in their aim of either reducing aliasing artifacts per se or producing soft shadows that even consider penumbra. Reeves et al. (1987) introduced "Percentage Closer Filtering" (PCF), which takes several (e.g., 4x4) binary lit/unlit samples and filters those instead of the depth map itself. Fernando extended that approach into a realtime algorithm delivering "perceptually accurate" (Fernando, 2005, p.1) soft shadows, approximating area lights' penumbra — "Percentage-Closer Soft Shadows" (PCSS). Donnelly and Lauritzen (2006) invented "Variance Shadow Maps" (VSM), where the mean and mean squared of a distribution of depths are stored, allowing to use the depths' variance instead of just a single sampled value, therefore effectively addressing aliasing.

– **Other solutions** involve strategies that either split or warp the light's or the observer's view frustums in order to optimize utilization of depth maps by concentrating their resolution where it is actually appreciated by the observer. "Parallel-Split Shadow Maps" (PSSM) — as the name suggests — split the observer's view frustum into several depth layers parallel to the view plane (refer to the "picture plane" in Figure 2.9), whereas for each layer, a separate depth map is rendered (Zhang et al., 2006). This allows to increase resolution of depth maps, which are closer to the observer, and vice-versa (ibid.). A similar approach, but optimized for vast sceneries lit by directional light sources, such as the Sun, is "Cascaded Shadow Maps" (CSM) by Dimitrov (2007). "Perspective Shadow Maps" (PSM) by Stamminger and Drettakis (2002) warp the light frustum so that it exactly matches the view frustum, thereby providing optimal depth map resolution, which, however, comes at the expense of un-intuitive changes to the light source(s)' types and direction, that must be considered. Wimmer et al. (2004) solved this drawback later by adding a transformation in light space into their "Light Space Perspective Shadow Maps" (LiPSM, ibid.). This approach effectively transforms point light sources into directional light sources, and prevents any further light source affecting changes (e.g., their direction), thereby solving PSM's major issue.

- **"Shadow acne":** Shadow mapping is inherently capable of self-shadowing, which occurs when an object is not only the occluder for another structure, but also for itself. Due to quantization of the z-buffer, however, there is potential for so-called "shadow acne": When a pixel is transformed from a surface in the observer's view into the light view, it should — in theory — again lie *exactly* on that surface.

But prior quantization disrupts this theoretical precision — sometimes shifting pixels behind that lit surface, putting them into the shadow. This might lead to unappealing spots on a surface (see Figure 2.12d).

Solution: By introducing a *bias* to the depth comparison, which is subtracted from the z-position of the pixel, it appears above the surface again and is, therefore, correctly lit. Since the bias shifts *all* pixels, not only those causing shadow acne, a potential side effect called *"Peter Panning"* may occur: Due to the shadow offset, objects look like they are floating. To compensate, shadow bias needs to be fine-tuned in a way that removes acne while keeping Peter Panning at a minimum.

Discussion

For the interactive, web-based solar shadow map approach, presented in this thesis, a method for shadow rendering is needed that a) is fast enough to work within web browsers (even on mobile devices) but at the same time b) provides comprehensible quality and precision. A typical scene will consist of reasonably smooth terrain (approximating curved surfaces), dozens, maybe hundreds or thousands of 3D buildings, as well as vegetation. While shadow volumes and ray tracing would provide a basis for analytical approaches, such as calculation of shadow areas and other interesting aspects, their rendering cost would most probably be too high to run on mobile hardware with interactive frame rates.

The focus of this work, however, lies in the interactive exploration of solar shadows. Hence, analytics are not considered for now, and the level of complexity, among the scenes to be visualized, calls for a shadowing approach that is detached from the scene's complexity. Due to the maturity of shadow mapping and its widespread support through GPU hardware as well as (web-based) 3D-engines, it seems like the most promising method considering given requirements.

2.1.3 3D data for shadow rendering

Previously presented shadow rendering algorithms rely on three-dimensional data, raising the question, where relevant solar shadow map data can be sourced from. Given its popularity, openness, and worldwide coverage, OpenStreetMap (OSM) is a suitable starting point when it comes to geodata of most forms. It is a vast collaborative project, a "map of the world", created by people around the globe and free to use under the Open Database License (ODbL) (openstreetmap.org, 2020). Raw data served in the XML-based *OSM* file format, covers the whole Earth and is weekly updated in the so-called "planet file" which — zipped — has a file size of 85 GB (planet.openstreetmap.org, 2020). This data, however, is not designed to be directly used by applications such as a solar shadow map: It is much more served to undergo further processing to end up as directly usable data like map tiles, building models, and other derivatives. Assumingly, a big reason for this is that the operation of servers and providing necessary bandwidth for the world's mapping demands is a costly endeavor and, therefore, outsourced to other companies and organizations, which will be further investigated in Section 3.2.

Dimensionality of data

There are various *map rendering engines* (refer to Section 3.3.1) which seem three-dimensional at first glance, but are, in fact, "2.5D" or "pseudo-3D". To increase rendering performance (by relying on simpler models and optimized-for rendering engines), as well as reduce download times (less data to transmit), support for "real" 3D data is sacrificed. Three-dimensionality, by name, is defined by the three-dimensional description of spatial information: Therefore, every point in space (referred to as *vertex*) is specified by three coordinates, x, y, and z. 2.5D, on the other hand, while relying on *two* explicit coordinates (x, y), merely simulates the third dimension through various "tricks", whichever works best for the given scenario: 2.5D buildings, for example, benefit from different strategies, than 2.5D terrain, which is further elaborated as follows.

2.5D vs. 3D buildings 2.5D buildings, utilized by some map engines described in Section 3.3.1, simulate three-dimensionality by adding a *height* attribute to a building's 2D outline, effectively generating *prisms* with parallel top and bottom faces. Such objects, however, are incapable of reproducing diagonal structures (e.g., pointed roofs) and can only approximate them using several adjacent sub-prisms, ending up with step function-like results (see Figure 2.13a for a comparison between 2.5D and 3D buildings). Conclusively, shadows cast by 2.5D buildings are — besides edge cases, where the original object actually resembles a prism — less precise than those of 3D structures that are inherently capable of diagonal and other arbitrary forms. As further discussed in Section 2.1.4, Biljecki et al. (2017) argue that the shadow precision gained by upgrading from LOD1 (which usually equals or at least resembles 2.5D structures) to LOD2 ("real" 3D) is — on average — not worth the computational expense. As described there, however, the underlying building models for their study were synthesized, which is why cases of real and complex buildings, such as the *Karlskirche*, a church in Vienna (see Figure 3.3), are inherently strikingly misrepresented. Section 3.2.2 further elaborates on LOD differences of existing (thus, not synthesized) building structures.

The discrepancy between *storage* and *rendering* of such data needs to be considered. A model, specified and stored in 3D, does not necessarily need to be rendered in 3D and vice versa, a decision best made according to actual requirements. This becomes even more relevant when data, instead of being defined by explicit vertices, is rather *described*: The OSM specification for "simple 3D buildings" (wiki.openstreetmap.org, 2020), for example, allows — besides height and building-level attributes, the attribution of a *roof shape*: Among to choose from are gabled, half-hipped, pyramidical or "onion" — how these roof shapes are eventually rendered, however, is left to the actual implementation of the map rendering engine. Regarding dimensionality, OSM has even drafts on *4D* models (wiki.openstreetmap.org, 2018), which allow consideration of *time* within their data: For example, a building might exist only temporarily while another is in a planning or construction phase and, therefore, undergoes frequent changes. 4D models can respect such change over time.

(a) 2.5D buildings. Image source: Screenshot of Tangram, © Mapzen.com

(b) 3D buildings. Image source: Prototype screenshot

Figure 2.13: The approximation of diagonal structures through many prisms in 2.5D buildings is visible among some structures in Figure (a), while the 3D counterpart in Figure (b) allows more freedom in regards to modeling precision, positively affecting shadow quality

2.5D terrain In contrast to buildings, 2.5D is arguably enough to represent terrain adequately. Nevertheless, Earth's topography includes overhangs, caves, and other structures where explicit $x/y/z$-coordinates would be required to achieve a precise reconstruction, albeit at high additional costs. The reduction to 2.5D, therefore, has advantages: Data can be distributed through *2D textures*, so-called *terrain tiles*, which can then be used to displace planar meshes vertically in order to generate 2.5D terrain. In most cases, those heightmap tiles are aligned with their basemap equivalent, allowing for an efficient texturing of terrain with a context providing basemap. Since there can be only one height for a given x/y-coordinate, however, it is not possible to have objects *above* or *intersecting* each other. Ergo, height becomes a function of x and y, and this height can be stored encoded within the pixel of a 2D texture.

Applications requiring 3D building models Biljecki et al. (2015) broaden the scope, providing an overview of 3D city model applications beyond solar shadow maps (see Figure 2.14). Their study finds 29 use cases within more than 100 different applications, such as noise and radio-wave propagation, microclimate analysis, or population estimation (ibid.). While there are use cases that would also run on 2D or 2.5D data, they only consider those, which either *require* 3D data to operate or *significantly benefit* from the dimensional increment through, e.g., increased accuracy or new possible features, as a consequence (ibid.). An illustrative example of insufficiency regarding 2D data and 2D visualization is the cadastre of Rotterdam, The Netherlands: As shown in Figure 2.15, it is incapable of adequately storing and visualizing the property situation of two separate buildings, overlaying each other vertically. While this scenario would obviously benefit from 3D data and respective 3D representation, it raises the question, whether 3D data sourced applications (like solar shadow maps) take advantage of 3D visualization *per se* — a concern which will be further discussed in Section 3.1.1.

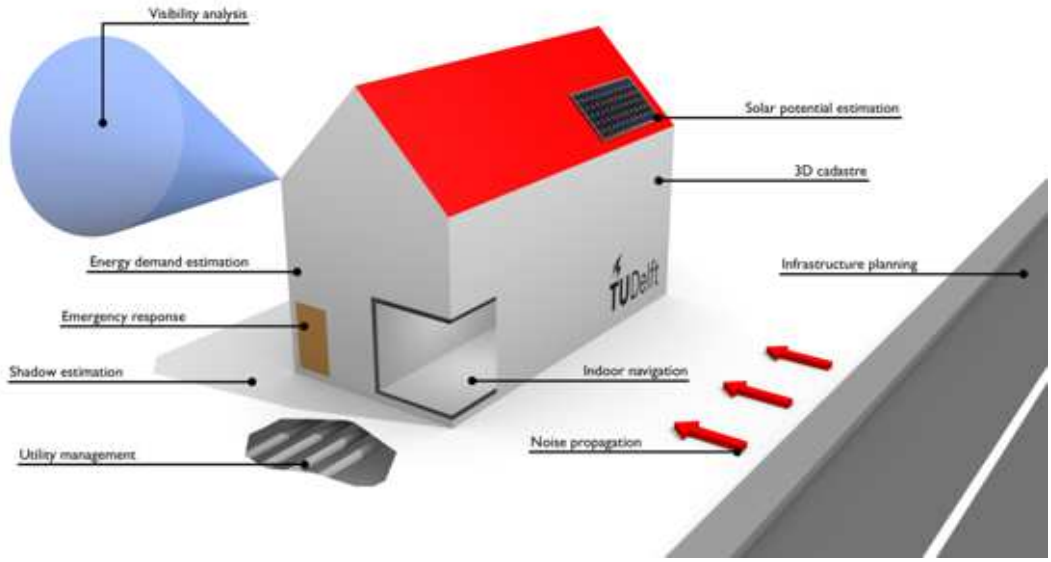


Figure 2.14: Biljecki et al. (2015) mention various application domains for 3D city models, including shadow estimation.

Figure 2.15: Since the 2D cadastre on the right is incapable of straightforwardly mapping the property situation of the two overlaying buildings on the left, it falls back to cutting the overlaid parts into several sections. A 3D data source, as well as visualization, would not require such measures. Image source: Biljecki et al. (2015), whereas the map originates from the Dutch Kadaster

Figure 2.16: Synthetically created buildings of varying LOD (LOD1 on the left, LOD2 variations on the inner two and LOD3 on the last) and the shadows, they cast (Biljecki et al., 2017)

2.1.4 Level of detail

To increase the performance of 3D applications, it is usually beneficial to employ various *Levels of Detail* (LOD), therefore, reducing the complexity of models far away from the observer, and avoiding unnecessary computation for detail, which would not be recognizable in the first place. 3D models are grouped among their fidelity, whereas the finer the detail, the higher the LOD. Hence, a lower LOD equals a higher degree of simplification of the mesh structure, achieved by iteratively removing details, eventually resulting in a representation far off its original.
Biljecki et al. (2017) apply LOD to 3D city models to answer the question of whether finer detail improves the quality of shadow estimations, and whether higher storage requirements and performance deficits in rendering more complex models contribute to significantly better results.

Their LOD levels follow the CityGML LOD definition by Kolbe et al. (2005), limiting them to levels from 1 to 3 (see Figure 2.16): LOD3 describes a close approximation to the ground truth (comparable to an optimized 3D scanned building with most of its details), LOD2 abstracts this down to a building of actual height but with simplified roof structure, while LOD1 is merely a 2D ground plan extruded to a certain height, therefore, omitting — among others — angled surfaces, including pitched roofs.

Obviously, a simplified structure is faster to render. However, its shadows (planar projections of the 3D LOD representation) are affected from such a simplification well. Instead of investigating real city models and layouts, Biljecki et al. (2017) used a tool to synthetically create buildings and city structures via randomized parameters for their experiment, and within that environment, conclude that shadows cast by lower LODs are *on average* not of significantly lower quality. As already argued in Section 2.1.3 already, however, distinct buildings like churches or other architectural sights are not reflected by synthesized buildings, and even if — on average — the error is not significant, it can have a drastic impact in individual situations (as seen in Figure 3.3).

Refer to Section 3.2.2 for a more in-depth analysis and evaluation of building-related LOD within the more practical context of Vienna's actual 3D city model.

2.1.5 The Sun

As the name tells, solar shadow maps' light source is the Sun. Its radius was not long ago, at least in astronomical terms, redefined by Emilio et al. (2012), who exploited the recorded mercury transits of 2003 and 2006 to increase measured precision ending up with 696,342 ± 65 km. Royer (2009) could prove that its shape is close to a perfect sphere, only slightly flattened on its poles, caused by centrifugal forces. The Earth orbits the Sun at an average distance of $149.6 * 10^6$ km, which dithers depending on the phase — resulting in a negligible deviation of the Sun's apparent diameter, as seen from Earth, averaging at 0.5331° (nasa.com, 2018).

From a computer graphics perspective, and as everyone who has ever seen the Sun immediately knows, this does not qualify for a point light source (i.e., one-dimensional), but rather an *area light source*. As Section 2.1.2 explained, this would have consequences: Instead of sharp shadow boundaries, area light sources cause computationally expensive *penumbra*. However, the penumbra directly correlates with the angular diameter, and therefore, the shadow's opening angle between the *umbra* and the adjacent fully lit surfaces is only 0.5331° as well. While this becomes recognizable at low Sun altitudes in combination with tall structures, as shadows stretch out on wide empty areas, it is neglected within this thesis. The Sun is thus abstracted as a *directional light source*, which describes a point light source, infinitely far away (refer to Figure 2.8 while imagining the point light source infinitely far away, resulting in parallel rays).

The plane, on which the Earth orbits the Sun, is called the *ecliptic* (see Figure2.17). Its angle towards the Earth's *equatorial plane*, which is normal to its rotational axis (itself aligned towards the north *celestial pole*), is 23.44° — termed *obliquity* (wikipedia.org, 2019a). This means that there is a divergence between the two rotational axes (Earth around itself versus Earth around the Sun) — the very reason for *seasons*: Figure 2.18 shows the related eight-shape pattern that can be observed when the Sun is photographed *at the same time* on different days throughout the year, called *analemma*.

While the whole system — Earth towards Sun towards the center of the Milky Way — is in constant motion (Figure2.17), slight deviations occur: For example, the obliquity angle oscillates between 22.1° and 24.5°, and is currently in its decreasing phase in a cycle that lasts 41,000 years (wikipedia.org, 2019a). Furthermore, there are variations in the Earth's orbital eccentricity (deviation from a perfect circle) around the Sun — which span over a 100,000 years cycle, as well as a "wobble" of Earth's axis that varies over a 20,000 year period (livescience.com, 2005). While some scientists believe that a coupling of those phenomena might even have caused ice ages, it definitely affects the Sun's position in the sky as seen from Earth *beyond* the year cycle: Thus, it will be at a slightly different position on January 23rd, 2020 than on January 23rd in, e.g., 2030 — at the same time of the day.

Earth's atmosphere causes refraction and slightly affects the perceived position of the Sun as seen from Earth. The lower the Sun's inclination, the stronger its impact as the sunrays' travel distance through the atmosphere increases. According to wikipedia.org

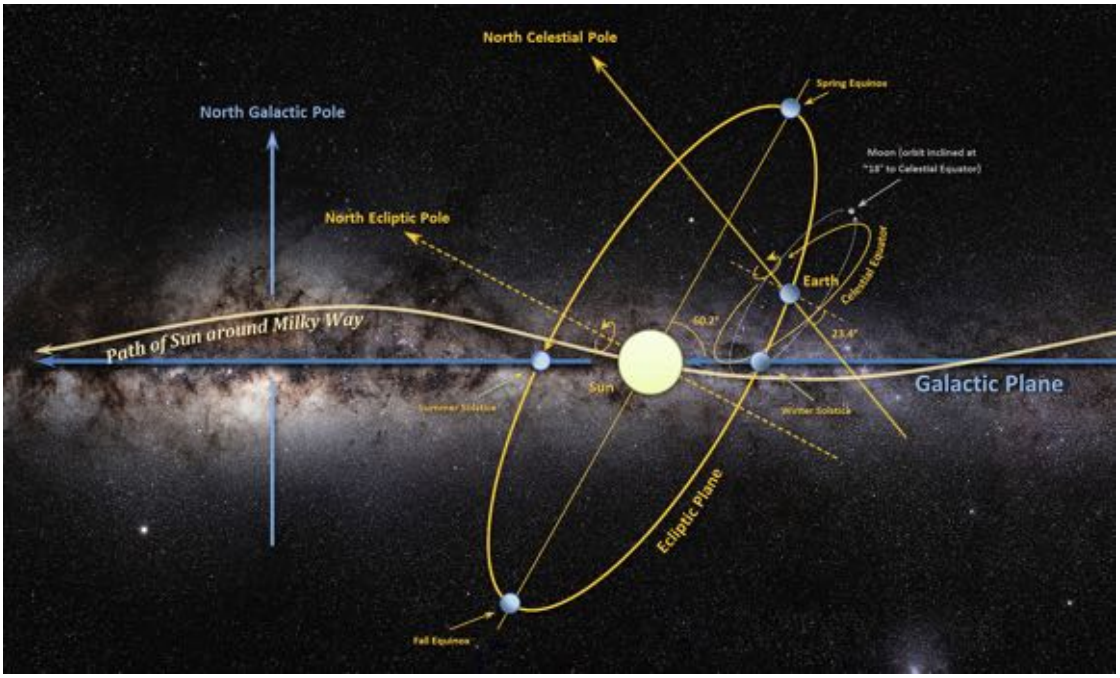


Figure 2.17: Earth's equatorial plane (celestial equator) is offset from the ecliptic plane at (currently) 23.44° as it orbits the Sun — the cause for seasons. The ecliptic plane itself is angled at 60.2° towards the galactic plane where the Sun orbits the Milky Way's center. A highly complex, gravitationally interlinked system. Image source: "Motion of Sun, Earth and Moon around the Milky Way" by *Jim slater307*, Wikimedia Commons, CC BY-SA 4.0 (colors adapted and cropped from original)

(2020a) — which itself refers to the research of Allen and Cox (2001) — this refraction *"is zero in the zenith, less than 1' (one arc-minute) at* 45° *apparent altitude, and still only 5.3' at* 10° *altitude; it quickly increases as altitude decreases, reaching 9.9' at* 5° *altitude, 18.4' at* 2° *altitude, and 35.4' at the horizon all values are for* 10° *C and 1013.25 hPa in the visible part of the spectrum."* (wikipedia.org, 2020a) Those 35.4' are interesting as they equal 0.59°, which is more than the Sun's apparent diameter as seen from Earth (0.5331°). This leads to the effect that *"when the bottom of the sun's disc appears to touch the horizon, the sun's true altitude is negative. If the atmosphere suddenly vanished at this moment, one couldn't see the sun, as it would be entirely below the horizon."* (ibid.).

All these factors influence solar shadows besides their obvious dependency on time and location on Earth. Thus, aa.quae.nl (2018) considers these facts within a complex mathematical model that allows the deduction of the Sun's position as a function of location on Earth and time. This model is actually implemented by SunCalc (SunCalc, 2016), and `www.suncalc.net` provides an interactive 2D map with respective functionality. Furthermore, its underlying system — as it was also used by at least one of the applications presented in the first chapter — is open-sourced and free to use.

Figure 2.18: This photography was assembled from various shots of the Sun over Stonehenge, taken at the very same time on different days of the year. The resulting eight-shaped path varies in its form on different locations around the Earth. Photo © Giuseppe Petricca, gmrphotographer.net

2.2 Existing approaches

As to existing approaches, three attempts, discovered in 2013 and 2014 (long before this thesis was even considered of becoming one) were already briefly introduced at the beginning of Chapter 1, to allow a fast dive into the topic. Since all of them never made it into (or remained as) an actual usable product, their review could not reach appropriate depth. This aside, in general, techniques aimed at visualizing solar shadows within urban areas for a single point in time or short time ranges — which are the focus of this thesis — are scarce, compared to related methods that average solar insolation over the whole year. The latter is a valuable research area for *photovoltaic potential*, as solar panels are inherently static devices: Therefore, it is reasonable to install them at locations that are most efficient throughout the year. Related work regarding solar potential is provided, among others, by Bremer et al. (2016); Catita et al. (2014); Freitas et al. (2014); Murshed et al. (2018); Wieland and Wendel (2015), as already introduced in Chapter 1 (refer to list item 5a within Section 1.1.2). Furthermore, there are solar analysis plugins for professional Building Information Modeling (BIM)-software targeted towards city

planners and architects, such as Autodesk Revit (Autodesk Revit, 2020). Google's web application "Project Sunroof" (google.com/get/sunroof, 2020) uses a DSM sourced by Google Earth data to provide users an estimation of solar potential for addresses within the United States of America. Even though the underlying DSM is three-dimensional, the generated visualization is 2D. Hence, vertical structures such as building facades, are neglected in the final output.

Leaving photovoltaic potential behind and coming back to solar shadow maps, this section will present the impressive and highly relevant work on "Shadow accrual maps", accompanied by its implementation "Shadow Profiler". "PhotoPills" — which stands in stark contrast to the previous application — provides a strange but creative way of retrieving shadow-related information. User experience (UX)-wise, it is essentially the antithesis of how a solar shadow map could work, at least compared to most other work discussed in this thesis. However, it still is a popular tool among photographers. "Stadtplan3D", another application with a quite limited feature set as well, covers the city of Vienna and demonstrates what can be achieved through a web application — but also shows the unfulfilled potential of a web-based approach.

2.2.1 Shadow accrual maps

The issue of higher population density, therefore, larger building structures in urban areas and accompanying challenges concerning management of sunlight in cities (refer to Section 1.1.4), is also the primary motivation of the work by Miranda et al. (2019). To provide guidance for city planners and architects, they focus on *time-integration* of solar shadows, as it was already mentioned in this thesis' introduction (refer to 1.1.6). By accumulating ("accrual" = accumulation) solar shadows over time, as they move in accordance to the Sun's motion relative to Earth, it is possible to visualize *how long* regions are shadowed (see Figure 2.19), thus, better understand the impact of existing buildings — but also of potential new structures before they are built.

While many different approaches on shadow rendering exist (see Section 2.1.2), none of them is inherently capable of accumulating shadows over time. However, they can provide the basis on which to build on, to create support for this feature. A naive approach would be to just render shadows for any time interval (e.g., every minute) and then average over those generated shadows. This method, however, becomes expensive very fast as too many light sources would need to be considered: For example, averaging over 5 hours with intervals of 1 minute, results in 300 light sources, which is out of scope for real-time rendering on most current hardware.

Therefore, Miranda et al. (2019) exploit an assumption (which they also validate in their paper) that given short enough time frames, solar shadows move linearly (see Figure 2.20) on *flat surfaces*. They tested this linear approximation for randomized durations and concluded that for $n = 60$ minutes, linear shadow motion between $n = 1$ and $n = 60$ maintains an acceptable divergence from the actual shadow position. Based on this linear movement, only two projections from the light's perspective need to be done (at time

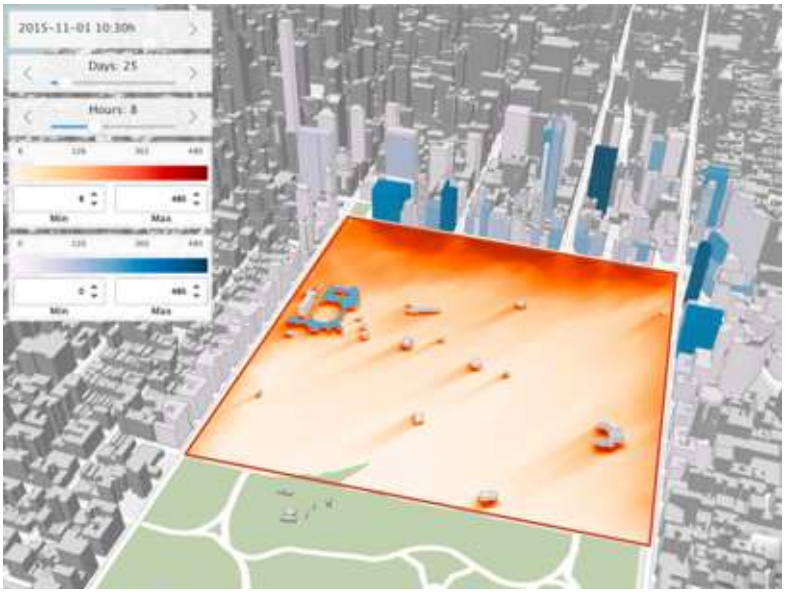


Figure 2.19: Screenshot of "Shadow Profiler", the visual analysis system created by Miranda et al. (2019): The UI on the left allows to specify a starting date and a time frame (in days and hours) which specifies the shadow accumulation duration. The blue buildings in the map are being evaluated for their shadow impact among the cityscape: Red areas are in shadow (the darker, the longer), and the darker the blue of the new buildings, the higher their ratio on those cast shadows.

t_1 and t_n), while any time step in between can be interpolated (*shadow accrual maps*). Thus, their approach allows for a significant reduction of comparably costly shadow map computations. If, however, the aimed for time-integration period increased from 60 minutes towards weeks or months, it would nevertheless result in too many of those 60-minute-spanning calculations: To solve that, Miranda et al. make use of another fact: *"While the direction of sun light at a given time in summer will be drastically different from the direction in winter at the same time (depending on the geographical location), the change in direction on consecutive days in summer (or winter) is minimal"* (Miranda et al., 2019, p.6). Their strategy is to compute shadow accrual maps only once for similar Sun positions (that may occur on different days at similar times among the specified long time frame, grouped together) and then, by applying a higher *weight* on them, to increase their impact on the overall shadow accumulation.

In order to fulfill individual needs, the assumption of linear temporal shadows — given short enough time frames — is combined with two different shadow rendering methods: Shadow mapping (refer to Section 2.1.2) as well as ray tracing (refer to Section 2.1.2). Accordingly, "Shadow Profiler", the application created by Miranda et al. as an artifact of their research, supports two different operation modes:

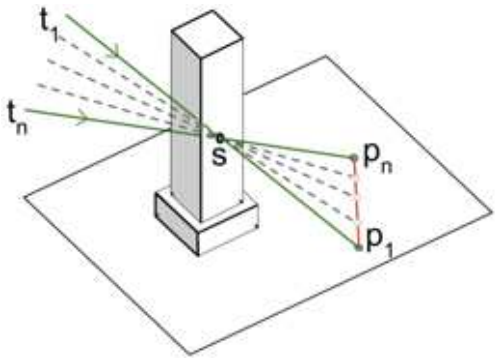


Figure 2.20: For short enough time frames (e.g., 60 minutes), the Sun's and shadow's motion is approximately linear. The point in shadow (p_1) at the time t_1 moves in a straight line towards p_n for time t_n (Miranda et al., 2019)

1. **"Shadow accrual maps":**
   Shadow accrual maps extend the fast, but quantization affected, shadow mapping approach to support time-integrated shadows. Instead of using a 2D texture for a single depth representation of the scene from the light source, Miranda et al.'s approach uses a 3D texture (= shadow accrual map), whereas every 2D slide of the 3D texture is used to store a depth map for a single time step t_i, ($1 \leq i \leq n$, referring to Figure 2.20). Every p_i, representing a point in the shadow caused by s at time step t_i, moving from p_1 towards p_n linearly, is stored in a separate 2D slide. The main acceleration factor here is that the depth maps of each of these intermediate time steps are not computed separately, but interpolated within a single pass on a modern programmable GPU by calculating p_i as follows:

$$p_i = p_1 + (p_n - p_1) \times \frac{tan((i-1)\Theta/n)}{tan(\Theta)}$$

   Whereas Θ is the angle between the light sources at time step 1 and n (Miranda et al., 2019)

   In contrast to traditional shadow mapping, which stores the distances from the light sources to the surface points, shadow accrual maps, however, store the distance from the occluder to the surface. Considering this deviation, Miranda et al. proof in their paper, that a 3D shadow accrual map (generated in a single pass covering n time steps, given — again — assumed linear motion of temporal shadows) *equals* n separately calculated 2D shadow maps. Hence, there is no loss in quality, while computation cost is still significantly reduced compared to a naive shadow mapping approach (see Figure 2.21). After the shadow accrual map is computed, the time-integrated shadow for each pixel can be calculated by straightforwardly counting the number of times p_i lies in shadow: The more often, the darker the shadow.

   This method provides fast frame rates, enabling interactive exploration of shadow behavior in the city. Due to its shadow mapping related nature, it needs to make

sacrifices in regards to visual quality and precision, nevertheless.

2. **"Inverse accrual maps":**
   To compensate for the quality drawbacks caused by shadow mapping's inherent quantization, Miranda et al. (2019), furthermore, created "inverse accrual maps" as an extension of ray tracing. The approach takes advantage of the aforementioned linear motion of temporal shadows (given short enough time frames) as well, accelerating required computation significantly compared to regular ray tracing (again, see Figure 2.21), reaching about half the speed of shadow accrual maps. Following its underlying principle, for all points visible from the observer (i.e., not occluded by buildings), rays are cast towards the light source, and all intersections with occluders until the light source is reached, are mapped. Due to that fact, there is a link between the cast shadow and the very building that occluded that surface area; information which can be used for further analysis.

   While interactive exploration is lost due to higher computation demands, the high precision of this approach combined with its fundamental mechanics allow for further analysis: Given a user-defined polygonal area R on the surface, it is possible to calculate the shadowed area (in m^2) within. Additionally, a *shadow score* can be calculated in a way that respects the beneficial effect of shadows in hot summers, as well as their (usually considered) negative impact in winters: Shadows can then be color-mapped and visualized accordingly, e.g., cooling shadows in summer in shades of blue (the darker, the more temporal shadow) while shadows in winter remain red as shown in Figure 2.19. The relevant weights for this score can be user-defined: E.g., positive weights for hot summer months, negative weights for cold winter months, and something in between for intermediate months (Miranda et al., 2019). The third analysis feature concerns *building contribution*: By exploiting the aforementioned fact that there is a connection between shadows and their occluders, it is possible to precisely quantify the shadow area caused by every building (as seen in Figure 2.19).

Discussion

"Shadow accrual maps" by Miranda et al. (2019) is an impressive piece of work. Time-integration of shadows was an early idea, even an essential pillar, during the conception phase of this thesis. During the process of working on the related prototype implementation, it got, however, somehow neglected, as the performance of naive approaches in web-based environments was poor, and many other things needed attention. Miranda et al.'s work actually surfaced very late in the development of the prototype, where it was too late already for it to be seriously considered. Their rendering speed gains, compared to naive shadow mapping and ray tracing approaches, as shown in Figure 2.21, are spectacular (5–10X faster). This, furthermore, magnifies as the rendering resolution or the integrated time range increase.

Nonetheless, the approach contradicts with relevant aims of this thesis:

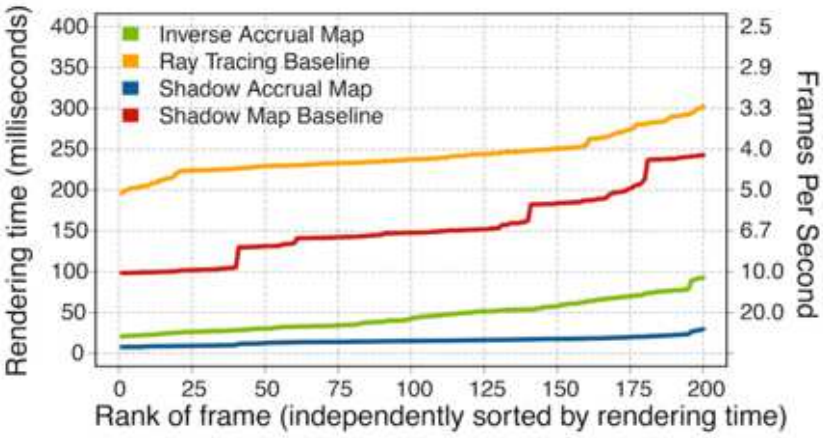


Figure 2.21: Performance comparison between shadow accrual maps, inverse accrual maps, as well as their related naive (baseline) implementations (Miranda et al., 2019)

- It is not web-based: Built upon C++ and OpenGL 4.3 (inverse accrual maps even require OpenCL 1.2), it requires technologies not available within a web browser and also rarely available in the native environments of mobile devices. However, 3D textures (as required by both methods presented in this paper) are part of the Web Graphics Library (WebGL) 2.0 specification and — in contrast to iOS-based hardware — supported by recent Android devices (caniuse.com, 2020; gist.github.com/TimvanScherpenzeel, 2020; threejs.org, 2020a). As soon as support for 3D textures/WebGL 2.0 gains broader support by web browsers, attempting implementation of web-based shadow accrual maps seems reasonable.

- It does not support uneven surfaces as shadow receivers: The whole method of significantly accelerated shadow mapping or ray tracing for temporal shadows is based on the linear motion of those shadows on — exclusively — *planar surfaces*. While this limitation is acceptable in the depicted — rather flat — Manhattan, NYC, it will render any application in cities, with somehow significant terrain, useless. Furthermore, their approach is not capable of rendering shadows cast on buildings. Miranda et al. are aware of these drawbacks and consider tackling them in upcoming work.

- Hardware requirements: According to the authors, the computer used for performance tests consisted of a "Intel Xeon E5-2620 CPU, 128 GB RAM, and an Nvidia GTX 1080 graphics card with 8 GB of GPU RAM" (Miranda et al., 2019, p.8). While mobile hardware undergoes constant impressive progress, especially memory-wise, this is another level. Shadow accrual maps are especially hungry for memory, requiring 60 times as much GPU RAM than regular shadow mapping, as one accrual map 3D texture needs to store all 60 time steps necessary to cover one hour.

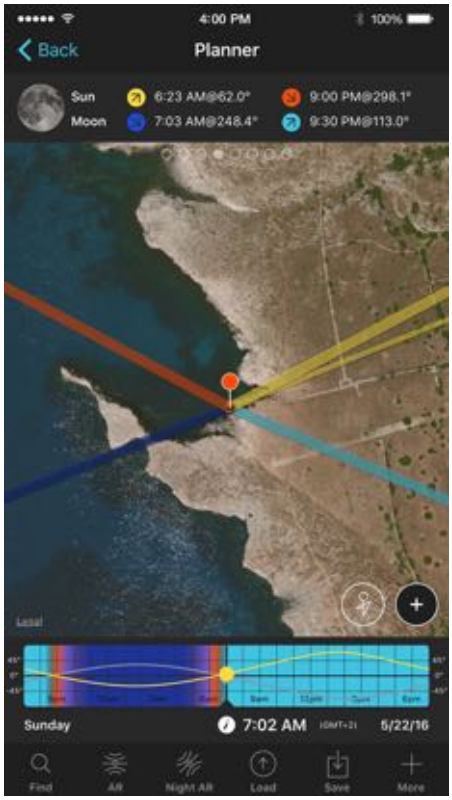


Figure 2.22: The red pin represents the specified location. Sunrise direction is visualized using a thick, yellow line, while the red line represents sunset direction. The direction of the Sun for a specified time is shown via a thin yellow line. This line gets dashed as soon as terrain occludes the Sun and solar shadows occur. Screenshot © photopills.com (2020)

2.2.2 "PhotoPills"

The native mobile application "PhotoPills" (photopills.com, 2020) comes as an example for a creative, albeit cumbersome way of retrieving solar shadow information. Aimed at photographers, it integrates assorted features to help them in the planning and execution of shootings by providing (among others) information about the position of the Sun, the Moon, and even the Milky Way. As for the Sun, it visualizes the direction of sunrise and sunset for a user-defined point on a two-dimensional map (see Figure 2.22), as well as the azimuth of the Sun for an arbitrary time. By using a slider UI-element, time can be adjusted, triggering the line representing the Sun's azimuth to rotate accordingly: As soon as terrain (and only terrain, no buildings or other structures/phenomena are considered) would start to hide the Sun from the specified viewpoint, that line simply becomes dashed.

Instead of visualizing solar shadows right on this map, "PhotoPills" is merely capable

of providing the binary information, whether a *single point* is in shadow or not. While a terrain-based solar shadow map could be created by sampling the map manually on a fictive regular grid (by moving the red pin around), this is clearly not the intended user interaction, and therefore the app's method is definitely an outlier, if not even an extreme case, compared to all the other approaches discussed in this thesis. This app was discovered during research related to the photography use case, mentioned in Section 1.1.5. It is used by professional photographers and receives good to excellent ratings, certainly confirming that its user experience is understood and valued, albeit for a limited, professional user group. However, as the shadow-lookup feature is just one among many, the overall acceptance might be biased by the app's other facets.

2.2.3 "Stadtplan3D"

The city of Vienna, Austria offers a 3D web-based city map that is capable of rendering shadows via wien.gv.at/stadtplan3d (2020). It is a web application, based on CesiumJS (refer to Section 3.3.1) and uses the city's 3D LOD2 roof model (refer to Section 3.2.2, as well as the city's own DTM for terrain. The latter seems to be of rather high resolution, as it is, even though being 2.5D, capable of depicting nearly vertical structures, such as artificially constructed river beds. The map itself is limited to Vienna. It is, however, possible to zoom out far enough to see the globe, albeit being empty outside Vienna's borders (2.23c). A drawback of this approach is a rather short visible range: As soon as the user zooms out further, buildings in the back start to be cut off (i.e., the view frustum is very limited, see Figure 2.23b) — probably to maintain rendering speed through limiting polygon count.

Regarding shadows, the application supports visualization for arbitrary days and times, using shadow mapping. The shadow map's texture size is too small to reduce the clearly visible jagged shadow boundaries, and shadows jitter significantly as the camera moves or rotates. On smartphones, the UI to enable shadows was unavailable altogether; their performance and quality could, therefore, not be tested.

"Stadtplan3D" gives an idea about what is possible within a web-context: The free to use visualization allows interactive exploration within a 3D approximation of the city of Vienna, including terrain, and provides basic, solid features like adjusting date and time. It is, however, limited to Vienna, and the quality of the shadows and visual range show room for improvement. The UI, as a whole, feels dated, and some features — including shadow rendering — are plainly removed on smartphones[5].

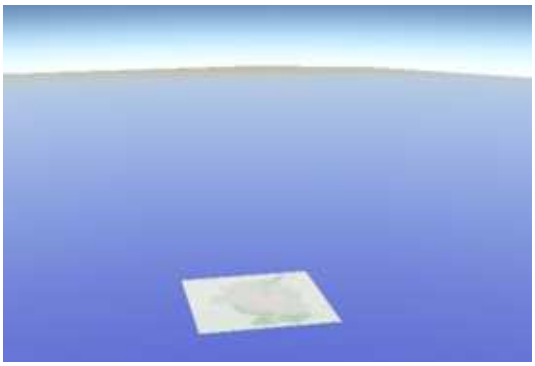
2.3 Discussion

The applications presented in Section 2.2 span out a space that contains a rough spectrum of possibilities: Within this space, "Shadow accrual maps" is on the cutting edge of what technology allows. Available computation power is utilized through intelligent design of

[5]Tested on an Apple iPhone 7 on 2020-01-14

(a) Vienna's "Stadtplan3D" web application and its UI to choose date and time for shadow visualization

(b) A narrow view frustum limits the visible range for buildings



(c) There is virtually no zoom-out limit for CesiumJS apps by default: Stadtplan3D only covers Vienna, though, resulting in an empty globe

Figure 2.23: Screenshots of "Stadtplan3D" (wien.gv.at/stadtplan3d, 2020) provided by the city of Vienna. Image source: Stadt Wien - ViennaGIS (www.wien.gv.at/viennagis/)

algorithms, allowing realtime and near-realtime features like the visualization of temporal integrated shadows over arbitrary time frames. These capabilities, nevertheless, come at the cost of accessibility: The hardware requirements necessary for this application to work, are — while not beyond reach — definitely unattainable by current mobile devices or within web-contexts. However, it is only a question of time until mobile hardware catches up, achieving the required processing and memory power.

"Stadtplan3D", on the other hand, runs fine in modern web-browsers, albeit suppressing shadow rendering on mobile devices — a crucial feature within this thesis. It nicely demonstrates, however, 3D rendering of a well enough detailed 3D city model and also considers terrain — something "Shadow accrual maps" is inherently not capable of since its shadow visualization only works on flat surfaces. "Stadtplan3D" also provides a first glimpse into the quality of potential 3D city models at LOD2, which is — at least visually — convincing (more on that in Chapter 3). Also, the on-demand downloading of the required 3D data on a mobile device works reasonably fast for further consideration. In regards to the introductory spectrum of possibilities, the shadow feature of "PhotoPills" stands out — if only in a direction not to go further. It, however, serves as a reminder to remain flexible when it comes to the design of an application's user interactions.

Most of the discussed approaches have distinct advantages, while all of them lack at least one critical feature among those defined within this thesis' problem space in Section 1.2. However, they blatantly expose areas of potential improvement, which, being equipped with gained knowledge from reviewed literature, becomes suddenly achievable. At least in theory: A new prototype that combines an optimized 3D city model with terrain (and maybe vegetation as well), inspired by the on-demand download of such data and equipped with an efficient, but at the same time visually appealing shadow rendering is worth the effort of trying to build it. Combined with an intuitive UI and UX it should be mature enough to be tested against the pre-defined problem space.

Further motivation comes from the non-engineering related literature, covering urbanization, the continually rising number of humans living in cities, as well as their "right to light". It is also fueled by the vast amount and transdisciplinarity of application scenarios that were also discussed from a historical perspective at the beginning of this chapter. A modern and accessible approach to a solar shadow map shall be pursued.

CHAPTER 3

Methodology

"I would rather have questions that can't be answered than answers that can't be questioned."

— Richard Phillips Feynman

Following the state of the art literature and related existing approaches, contained within the problem space of solar shadow maps, as they were presented and evaluated in the previous part, this chapter will lay its focus on the underlying requirements of a technical prototype.

By building such a prototype, it shall be investigated, whether a tool that solves the considerations and challenges mentioned in Section 1.1.6, is even feasible with available data on current end-user hardware. As the aim is to provide interactive exploration, the system needs to work fast enough to render at least 30 FPS on desktop computers, tablets, and smartphones. All this while integrating most, if not all, of the data listed in 1.1.6, "Data".

Furthermore, it needs to be taken into account that many people have never contemplated solar shadow maps. Hence, there are no established ways of interacting with them. At the same time, digital mapping technologies are widespread throughout the internet: According to buildwith.com (2019), 117,716 of the top 1 million websites use digital maps — 87 % of them are based on *Google Maps* technology while the remaining 13 % are shared among *Yandex, Mapbox, OpenStreetMap*, and others. UX- and UI-wise, mentioned top stakeholders share similarities; therefore, it is reasonable to take up their widely known design patterns and augment them with solar shadow map-related features.

A solar shadow map relies on heterogeneous data: Terrain, vegetation (i.e., trees) as well as buildings usually reside in completely different file formats and reference coordinate systems. They might not even contain actual three-dimensional information but are rather descriptive meta-data — in other words, abstraction levels diverge. Their integration into a single application requires notable consideration. Another crucial aspect is file size:

3D data with a level of detail that is high enough to provide a basis for precise shadow rendering is usually too big to be used in a web context: Download times would be overly time-consuming for appropriate UX. Therefore, and since interactivity indicates freedom for users to pan, zoom, and rotate the map, dynamic retrieval of — in the best case — worldwide data is a necessity.

3.1 Design considerations

There are many ways to design the prototype of a solar shadow map. Various reasons for a three-dimensional representation were given throughout this thesis, and so the question remains, how this could be implemented in practice. An important aspect here is *zoom*, as it directly influences the amount of data that needs to be loaded and rendered. The further the observer is away from the surface, the more terrain, buildings, vegetation, and basemap textures need to be considered simultaneously. Similar thoughts apply for unconstrained panning of the map. If so, data needs to be dynamically downloaded given the currently visible scene as this cannot happen upfront: E.g., OSM's global "planet file" (refer to Section 2.1.3) alone currently measures 85 GB — LOD2 buildings not included.

An approach towards this challenge would be a total restriction on zooming and panning, thus, restricting the viewable area. This, however, departs too far from the idea of a map and even more from the paradigm of interactive exploration. By expanding the explorable area to the size of Vienna, however, it would still be too stark of a constraint if there was a fixed zoom: Who knows what the user wants to explore at the very moment? The hills and parks of Vienna's northern border or their apartments in the city center? It can be argued that such freedom is a requirement. Given that, a conclusion — following mathematical induction — can be drawn: If functionality for zoom and pan on a city- and neighborhood-scale can be achieved, it compulsory also works on a world-scale.

In any scenario, a requirement is that heterogeneous data needs to be combined, which refers to research question *"1./a) [...] how can the required heterogeneous data be a) retrieved, b) processed and* ***c) combined*** *in a way that enables this* [a solar shadow map]*?"* (refer to Section 1.2). Section 3.2 will then cover data *retrieval* and *processing*, whereas the latter will also be further discussed in Chapter 4.

3.1.1 2D pre-rendered vs. interactive 3D visualization

In Section 2.1.3, the question was raised, whether a 3D data sourced application actually benefits from a 3D visualization — or if 2D might just be enough. Actually, this is more of a discussion in the context of UX and UI, since even an actual three-dimensionally constructed map, containing shadows generated via 3D shadow rendering methods, can be constrained to appear and behave like a 2D visualization. By using orthographic projection for the observer's view and disabling any tilting of the camera, the map practically becomes 2D in its interaction behavior, while still providing realtime shadows created from 3D data for arbitrary points in time.

Going further in the strategy of omitting the third dimension from the user would be offline rendering of 2D tiles based on 3D models and, instead of allowing complete temporal freedom, providing users a limited set of discrete moments of possible shadow scenarios. Immediate drawbacks of this approach are, however:

- **Storage requirements:** Considering an average 12 hours of daytime in western Europe, as an example, just the depiction of shadow situations in 10-minute intervals results in 72 different Sun positions among those 12 hours. If map tiles should be rasterized, for every supported zoom level, 72 different versions of the whole map need to be created — and potentially stored. In case of vector tiles, it might be possible to re-use shadows for more than one zoom level; different levels of detail, however, would most likely produce misalignments of shadows and buildings or vegetation. Another possibility would be a server-sided on-demand rendering of 2D tiles: Hence, for every client request (i.e., clipping and time), tiles are processed on the server, and as soon as they are done, transferred to the client. While this approach frees up storage resources, it increases computation time which needs to be considered as soon as client requests reach a critical level.
  Realtime 3D visualization, on the other hand, allows interactive shadow visualization. Hence, shadows don't need to be pre-rendered, stored, and transmitted. Required 3D model data (e.g., buildings, terrain, vegetation) for the current map clipping demands a prior download from the client. Some data is dependent on zoom levels (e.g., the basemap as well as terrain heightmaps) and needs to be updated as soon as the zoom changes (similar to the aforementioned 2D approach) — other data is resolution independent, such as Sun position or building models (this changes again, as soon as LOD is added). But since the rendering itself happens interactively on the end-user's device, any finer granulation of zoom, perspective, clipping, or solar situation can immediately be visualized without any further data download or processing. Freedom in temporal and spatial aspects is one of the core advantages with the added benefits of a reduced storage footprint on the server-side as well as reduced continuous downloads.

- **Limited freedom:** While vector maps allow at least seamless zooming, raster maps do not. 3D maps with a rasterized basemap, however, share that same disadvantage of hard steps between zoom levels. All 2D attempts intrinsically disallow tilting; hence, shadow situations on facades or other vertical structures can not be investigated by the observer. In regards to time, only those shadow situations which were pre-rendered and downloaded are available. These aspects constrain the free exploration of shadow maps.

- **Data transmission:** Data necessarily needs to be transmitted from the server to the end-user's device (i.e., client). For the 2D approach, this means that any current region for a specific point in time needs to be requested from the client and transmitted from the server to that client. Tiles themselves can be compressed and, therefore, reduced to practical file sizes, speeding download times up, allowing to

display an initial scenery rather fast.
Interactive 3D approaches, in contrast, require all the 3D model data for the given clipping upfront in order to be able to render the final image. This results in potentially longer initial download times while reducing the need for subsequent data, as needed for 2D cases of zooms or changes in time.

While a solar shadow map could arguably work in 2D instead of 3D (albeit with reduced features), a consequent step of thought would be to renounce *visualization* of results in general: Instead of providing the rendering of a map, depicting where sunny areas are, it would be legitimate to just voice-navigate a human where to go in order to find some Sun. Or maybe the interest lies in bare numbers nevertheless, like the number of hours the Sun shines through an apartment's windows during the whole year. There are numerous related examples, and they further illustrate the decoupling of dimensionality between data and results. Conclusively, the decision depends on the desired usage scenarios.

3.1.2 3D slippy map

An established pattern to solve the inherent issue of a zoom- and pannable digital map in 2D is a so-called *slippy map*: Instead of downloading and rendering the whole Earth at the same time, its surface — in this case, *planar projected* — is divided into sets of rasterized *tiles*; one set per zoom level. These tiles can then be dynamically requested from a server, so only those, which are currently visible at the current zoom level, are loaded and rendered. This basic method allows users to pan and zoom a map interactively and became a standard in web mapping related UX. Thus, besides the improved situation in data retrieval and rendering, the adoption of this approach increases the familiarity with a solar shadow map for new users, as many of them are used to interact with mainstream online maps. Since rasterized tiles (used for the basemap and terrain) are tied to a specific zoom level, quality is maintained — however, at the cost of "hard cuts" during the change of zoom levels. OSM's wiki has relevant resources covering slippy maps as such (wiki.openstreetmap.org, 2019a), zoom level considerations (wiki.openstreetmap.org, 2019c), and indexing of tiles (wiki.openstreetmap.org, 2019b).

Most importantly, this pattern can be brought into 3D space: Instead of placing tiles directly on a 2D canvas, 2D planes act as a proxy and are positioned in 3D spaces — planar, next to each other, then textured (as well as displacement mapped in case of terrain tiles) accordingly. The 2D optimization that only those tiles are loaded and created, which are visible to the observer, also applies to the 3D variant — however, with a significant deviation: Since the observer's view can be tilted, due to perspective projection, tiles along the horizon could be loaded in vast amounts until the data set's end. This can be avoided by either limiting the tilting depending on the zoom level, introducing a LOD approach, or limiting the view range.

3.1.3 Coordinate systems, tiles and other data

A coordinate system allows the reference of any location on Earth's surface. An established way is to use *latitude* (Lat.) and *longitude* (Long.): The former is for the vertical aspect of a point on the globe's surface, thus, the angle between the equatorial plane (refer to Section 2.1.5) and the point in question, imaginary connected through Earth's center, spanning a range from 90° S (South Pole) to 90° N (North Pole). The latter — horizontal position — is also specified by such an angle, albeit starting at the *prime meridian* and spanning over 180° W and 180° E, covering the whole world.

WGS

To further standardize coordinate systems, Spatial Reference Systems (SRS) are used to define, among others, units, coordinate limits, and a specification of the referenced surface (e.g., planar, cylindrical, spheroid, et cetera). Different SRS vary in these attributes and are optimized for respective use-cases and geographic coverage — whereas, through this standardization, conversion between different SRS is enabled. A system designed to work on the whole Earth is the *World Geodetic System (WGS)*, its current revision being *WGS 84* with European Petroleum Survey Group (EPSG) identifier EPSG:4326. It defines the location of the prime meridian at 102 m east of the *Greenwich meridian* at the Royal Observatory (wikipedia.org, 2020e), and models Earth's surface unambiguously as an "oblate spheroid with equatorial radius $a = 6378137$ m at the equator and flattening $f = 1/298.257223563$" (ibid.). *WGS 84* is popular in cartographic applications that require full Earth coverage, such as Global Positioning System (GPS), and also used for Vienna's tree cadastre (refer to Section 3.2.4), or as input for the library `sphericalmercator`, which is used for coordinate conversion and further described in Section 3.1.3.

Mercator projection

As map tiles are 2D, a projection needs to be applied between the Earth's spherical surface and the targeted flat plane. A widespread cylindrical projection is the *Mercator projection*, famous for its significant inflation of area close to the North and South Pole: Due to the cylindrical unrolling of the Earth's sphere, those areas need to be stretched notably in order to be flattened. A variant of this projection is *Spherical Mercator* or *Web Mercator* (EPSG:3857), which slightly adapts the classic Mercator projection and is used by basically all relevant online mapping platforms, such as "Google Maps, Mapbox, Bing Maps, OpenStreetMap, Mapquest, Esri, and many others" (wikipedia.org, 2020d). Due to the fact of *Web Mercator* being the quasi-standard for online maps, it is reasonable to use it for the considered prototype.

Tiles, their indexing and zoom levels

Regarding slippy maps, wiki.openstreetmap.org (2019b) lists relevant aspects that need to be considered when working with map tiles. Every tile is indexed via its a) *zoom* level, b) its x-coordinate and c) its y-coordinate:

- Horizontally, the x-index, related to the longitude, ranges from 0 to $2^{zoom} - 1$, whereas the leftmost edge equals 180° W while the rightmost edge equals 180° E (ibid.).
- Vertically, the y-index, related to the latitude, ranges from 0 to $2^{zoom} - 1$ as well, whereas the top edge equals 85.0511° N and the bottom edge equals 85.0511° S in a Web Mercator projection (ibid.). Due to this limitation, neither the North nor the South Pole can be visualized in a Web Mercator variant, as both are projected at infinity (ibid.).

This leads to some conclusions:

- Tiles are quads.
- With every zoom level increase by *one*, there are *four times* as many tiles as before. In other words: With every zoom increase, a single tile gets split up into four *subtiles*, calling for the use of a *quadtree* data structure.
- While locations on Earth using *latitude/longitude* are usually defined with *latitude* first, followed by *longitude*, this now changes to the mathematically established style of mentioning the x-coordinate first.
- The higher the *zoom* level (the closer the observer to the surface, the finer the level of detail), the more tiles are needed to cover Earth's surface:
  - For example, at zoom level 4, 256 tiles are enough to span over the whole world, and the resolution allows the depiction of continents in appropriate detail.
  - At the other end of the scale, at zoom level 17, the level of detail is high enough to depict adjacent blocks of houses. To allow such a resolution, applying the aforementioned formula, Earth coverage now requires 17,179,869,184 (17 billion) tiles. Such vast amounts of data call for on-demand access.

Conversion between coordinate systems

In order to integrate heterogeneous data with varying coordinate systems into a single 3D scene, conversion is necessary (refer to research question *1./a* in Section 1.2). As everything eventually comes together in a 3D space, spanned up by a three-dimensional Cartesian coordinate system, it becomes the reference in which all the different systems — with mostly geodetic origin — need to be mapped into. Since map tiles are already rasterized and indexed, they can be directly placed adjacently in the 3D scene. They actually define the *scale* of the whole map as distances on their rasterized version need to be considered for any other geo-referenced object in the scene. Some required conversion formulas and functions are described as follows:

- **Get x/y tile indices for *latitude/longitude* (lat/lng) at a given *zoom* level** (wiki.openstreetmap.org, 2019b):

  A prior requirement is to convert the latitude from *degrees* to *radians*:

  $$lat_{radians} = lat * \frac{\pi}{180}$$

  Tile indices (x_{tile}/y_{tile}) can then be retrieved via the following formulas, whereas $n = 2^{zoom}$ (ibid.):

  $$x_{tile} = n * \frac{lng + 180}{360}$$

  $$y_{tile} = \frac{n}{2} * \frac{1 - \log(\tan(lat_{radians}) + \cos(lat_{radians})^{-1})}{\pi}$$

- **Conversion from *latitude/longitude* into screen pixels and vice-versa:**

  The library `sphericalmercator` (github.com, 2019) provides "projection math for converting between Mercator meters, screen pixels (of 256x256 or custom-sized tiles), and latitude/longitude." (ibid.), which is highly relevant in a solar shadow map to, e.g., position the map at a desired location, place trees or buildings at their correct position, et cetera. Among the functions the library provides, the two relevant ones are:

  - `px(ll, zoom)`: Takes an array of *WGS 84* Lat./Long. coordinates in (`ll`) and a `zoom`-level, and returns screen-pixel coordinates (x/y).
  - `ll(px, zoom)`: The inverse of the former.

Consideration of spatial reference systems

Building data might be based on different reference systems than WGS 84 (EPSG:4326) or Spherical Mercator (EPSG:3857). There are numerous amounts of spatial reference systems that, by being limited to a constrained area on the globe, increase possible resolution within — a benefit appreciated by municipals. For example, the 3D city model of Vienna (as further elaborated in Section 3.2.2) references the "Gauß-Krüger in Meridian 34" (EPSG:31256). Values in this coordinate system need to be converted before they can be correctly placed in a scene based on WGS 84 or Spherical Mercator.

3.2 Data

This section aims to provide answers for research question *1./a)* (refer to Section 1.2), in regards to the *retrieval* and initial *processing* of data. Regarding the scope of data, it was concluded that it is reasonable to start with the city of Vienna, Austria, as a first test scenario for a solar shadow map, due to the following reasons:

- There is a personal connection to the author, as he lives here for 15 years, and it is also the home of TU Vienna, where this thesis is written. The city is, therefore, familiar, and as mentioned in the introduction, the original idea also manifested here six years ago.
- Vienna is diverse, considering its topography and city structure: Not flat but not too hilly, some urban parts but also lots of parks and forests. There are areas with tall buildings, others are wide and open, while the old town consists of small little streets and complex, old architecture.
- The city has an exemplary, albeit not perfect, Open Government Data (OGD) program — covering data of 3D buildings, terrain, and even a tree cadastre — all free to access, but updated very infrequently.
- Due to the local reference, validation of the prototype's correctness and quality is simplified.

3.2.1 Variants and sources of required data

Introduced in chapter 1 (refer to 1.1.6), the actual data considered in our approach of a solar shadow map consists of **3D buildings, 2.5D terrain, 3D vegetation,** and a **2D basemap**.

The first three are all are potential shadow occluders as well as receivers. Another occluder, weather phenomena, is consciously excluded from the method for now. The 2.5D terrain shall be augmented by a 2D basemap (= texture) to provide navigational context by displaying street names and points of interest.

Chapter 2 (Section 2.1.3) introduced OSM as a universal geodata source for cartographic needs, which, however, is not designed to be directly accessed and used from within custom web applications. As a solution, there are commercial data providers — themselves significantly relying on OSM data, but running their own server infrastructure — providing Application Programming Interface (API) endpoints that can be directly accessed. For the *basemap* and *elevation data/terrain*, examples of such services are *Mapbox.com*, *Mapzen.com* (now being a Linux Foundation project), or *Toursprung.com*. For *buildings* and *vegetation* data, Vienna's OGD program will be considered.

3.2.2 Buildings

OpenStreeMap building models

In regards to 3D building data, OSM needs to improvise as its underlying data model is required to stay compatible with existing 2D maps and renderers. Therefore, *volume* (i.e., the third dimension) is created by enhancing 2D building outlines with *tags* (= metadata), such as `height`, `roof:height`, `roof:shape`, etc., as shown in Figure 3.1. As further elaborated in the following section, this attribution creates room for interpretation compared to a 3D file format such as Drawing Exchange Format (DXF), where every

Figure 3.1: Screenshot of the OSM editor with the map part showing Vienna's Stephansdom (openstreetmap.org, 2020). It displays the respective attributes of selected (sub)structures (marked in the satellite imagery map view) and allows their editing (highlighted in the left-hand table). Those attributes include various roof parameters, such as: `roof:colour`, `roof:direction`, `roof:height`, `roof:material` and `roof:shape`. By translating these variables into 3D shapes, LOD2 approximations of such models are possible: E.g., the selected substructure of Stephansdom is 60 m high and has a 38 m high roof, which is gabled and oriented at 35° north-east

vertex is precisely defined by explicit $x/y/z$ coordinates. Even more, this interpretation of metadata needs prior engineering: It requires a processing step where an actual 3D model is created out of the OSM provided metadata, which is a project on its own. Considering coverage in Vienna, only sights are attributed in such detail — the majority of buildings are still represented as a LOD1 model based on their vertically extruded outline. As further argued at the end of Section 3.2.2, LOD1 diverges too far from the real building to provide realistic shadows; hence, a higher detailed data source is needed. OSM's LOD1 world coverage is nevertheless impressive and might, in the future, serve as a fall back in areas where no LOD2 is available.

Some cities go beyond LOD1 representation or metadata described roof structures, freely and directly providing at least LOD2 variants of their city models. Besides Vienna, there are New York City, USA, and Berlin, Germany. Additional European countries with open data initiatives — covering various cities which are, however, not further researched yet — are Belgium, Finland, France, Germany, The Netherlands, and Switzerland (citygmlwiki.org, 2019).

3D city model of Vienna

Vienna provides an LOD1 "Baukörpermodell" (building model) (BKM Vienna LOD1, 2020), as well as an LOD2 "Dachmodell" (roof model) (wien.gv.at, LOD2) via its OGD initiative, covering the whole city area, totaling at around 200,000 buildings (ibid.). Its height data references the "Wiener Null", defined as 156.68 m above sea level,

and the reference coordinate system (as mentioned already at the beginning of this chapter) is the "Austrian National Coordinate System" with EPSG code 31256, also known as "Gauß-Krüger in Meridian 34" (ibid.). All LODs (refer to Section 2.1.4) and its data variants can be freely retrieved via their "Geodatenviewer" (geodata viewer) — accessible via www.wien.gv.at/ma41datenviewer/public/start.aspx — and used under the precondition of correct attribution of the "Creative Commons attribution 4.0 international license (CC BY 4.0)" (ibid.).

LODs

LOD1:
The LOD1 model is available as *DXF* as well as *BKM* (both CAD file formats), and it is created by aerial photography analysis to get height values and applying those to extruded ground plans, retrieved from the city's area map (BKM Vienna LOD1, 2020). To approximate the ground truth, buildings are split into several prisms, based on their "distinctive height structure" (ibid.): *"Für das Baukörpermodell wird ein Gebäude aufgrund seiner markanten Höhenstruktur in einzelne Teile zerlegt."* (ibid.). To retrieve the lower height of pass-throughs or overlaying structures, where aerial photography obviously fails, manual measurements are carried out (ibid.). According to BKM Vienna LOD1 (2020), the precision of height measurements is ± 25 cm.

LOD2:
LOD2 is available as either *CityGML* or *DXF* 3D mesh and according to wien.gv.at (LOD2), the model is half-automatically generated by intersecting the city's DSM with its area map and model fitting "prototypical roof shapes" (ibid.) accordingly. Whenever the merger was insufficient, the roofs are manually refined. Hence, acquired data lacks the utmost precision, as it could be achieved via photogrammetric measurement (ibid.). A highly relevant aspect, since the models will eventually be placed within 2.5D terrain, is that a building's bottom surface is defined by the lowest intersection with the terrain (ibid.). This prevents buildings located within an angled slope to have parts of it floating above the ground.

LOD3:
Vienna actually provides an LOD3 model as well. It takes the progression from LOD1 to LOD2 further and supports even finer details, like small facade elements, gable windows, or other roof structures. According to wien.gv.at (LOD3), its data is exclusively photogrammetry based, whereas the final triangulation into 3D models happens as an automated process. The LOD3 is updated every three years and, in 2018, covered 81,000 buildings of Vienna's inner districts, omitting the other 119,000 buildings, which are included in the LOD1 and LOD2. Furthermore, the data cannot be easily retrieved via the previously introduced "Geodatenviewer" and requires personal correspondence and even administrative charges. Furthermore, since LOD3 provides finer details, also the vertex count and file size, thus computation costs increase. It can be — even though this was not tested yet — assumed that LOD3 does not dramatically improve the shadow precision as the added detail, as the name suggests, affects only small structures of a

(a) LOD2 (b) LOD3

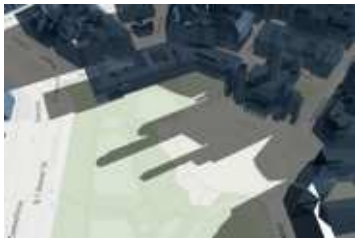
Figure 3.2: Comparison of Vienna's LOD2 (a) (wien.gv.at, LOD2) and LOD3 (b) (wien.gv.at, LOD3) shows LOD3's finely detailed roof structures and facades. Image source: Stadt Wien - ViennaGIS (www.wien.gv.at/viennagis/)

building (see Figure 3.2 for a comparison between LOD2 and LOD3). Thus, LOD3 is not further investigated within this thesis, but might be evaluated in future work.

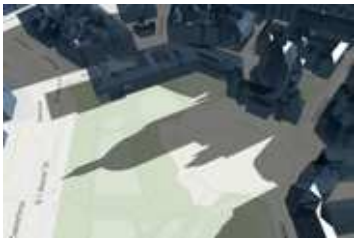
Comparison of LOD1 and LOD2:
Due to its simpler models, LOD1 requires less storage size than LOD2. E.g., a DXF-file containing *Karlskirche*, among other buildings, as shown in Figure 3.3, requires 16.6 MB instead of 27.4 MB. This, however, manifests in regards to modeling precision: The LOD1 in Figure 3.3a obviously lacks the church's prominent dome construction, as it is replaced by a narrow column. The LOD2 version (Figure 3.3b), however, while subjectively approximating the dome structure nicely, significantly reduces the height of the two major pillars in front of the church (compare to Figure 3.3c). Furthermore, there seem to be unexpected misalignments as, for example, the little cupola on top of the main dome, while being visible in LOD2, is omitted in LOD. This can visually be deduced from the cast shadows in Figure 3.3. This is, however, surprising, as both LODs — according to the aforementioned specification — rely on the same airborne laser scanned DSM for building height. Reasonable explanations for this divergence would be a) manual interference or b) misalignments between the DSM's grid and the split LOD1 prisms. It does not explain why the side pillars in the LOD2 model are too small, in any case.
Considering shadows, the lack of the main dome in LOD1 produces a notable gap, while the wrong height of the side pillars in LOD2, however, consequently reduces their shadow area in turn. LOD2 is at least theoretically more capable of capturing curved surfaces like domes or pointed roofs, which are frequent in Vienna. Hence, it is the aim to continue working with LOD2 and address its higher storage and rendering demands accordingly.

LOD2 Data formats

Compared to the DXF variant, the CityGML version contains additional metadata like, e.g., the area and angle of roofs, written out surface heights, as well as a unique ID to allow identification of whole buildings (wien.gv.at, LOD2). While the latter might become



(a) LOD1. Image source: Prototype screenshot



(b) LOD2. Image source: Prototype screenshot

(c) Karlskirche. Photo: "Karlskirche Abendsonne" by *Thomas Ledl*, Wikimedia Commons, CC BY-SA 4.0 (cropped from original)

Figure 3.3: LOD1 (a) and LOD2 (b) show drastic differences in their structure, significantly affecting respective shadows. Both deviate from the ground truth (c)

useful for additional features, it is not further considered for now. The CityGML-metadata obviously generates storage overhead, but both data sets are enormous nevertheless: The sum of all CityGML models, covering whole Vienna results in a roughly 11 GB sized big file collection, while DXF still requires 9.3 GB. Those file sizes obviously contradict fast downloads and, therefore, usage on mobile devices and require further processing and/or compression to become usable (refer to Chapter 4).

Update cycles

Unfortunately, the LOD2 model is already outdated in some parts of Vienna, as it was acquired in 2013 and not updated since then. Among others, this affects structures like the "DC Tower 1" (see Figure 3.4), a significant shadow occluder and actually the tallest building of Austria. Its construction was finished in early 2014 and is therefore not included in the LOD2's last iteration. However, it is mentioned that city municipals want to continue the offering (and assumingly update) of the LOD2 model (wien.gv.at, LOD2). The LOD1 model, in contrast, undergoes a more frequent 3-year update cycle.

3.2.3 Terrain

As argued throughout this thesis, terrain is an obvious shadow occluder and receiver and should, therefore, be considered within a 3D solar shadow map. Figure 1.7 depicts a shadow scenario in the hilly outer districts of Vienna, where, in the late afternoon, buildings are in topographic shade. Compared to 3D buildings — rather than loading 3D meshes into a scene — 2.5D terrain can be created on demand by applying a depth map (also referred to as heightmap) onto a 2D plane. The aforementioned *terrain tiles*, thus, are just image files, containing such *depth maps* (or *heightmaps*).

Terrain tiles, usually provide elevation data encoded among the R, G and B (red, green, and blue) channels available in common web-supported image file formats, such as PNG.

Figure 3.4: Since 2014, "DC Tower 1", located in Vienna, is Austria's tallest building. Because the city's LOD2 data set was not updated since 2013, it cannot be considered within a solar shadow map, albeit being a significant shadow occluder. Photo: "Ansicht von der Reichsbrücke aus gesehen" by *Hubertl*, Wikimedia Commons, CC BY-SA 4.0

This approach increases precision by offering 3 * 8 bit = 24 bit compared to just grayscale depth maps (8-bit), allowing a maximum theoretical resolution of 16,777,216 vertical units. Given Mount Everest's summit as the highest point on Earth at 8,848 m (wikipedia.org, 2020c) and the Mariana Trench at around 11,000 m depth (measurements vary according to wikipedia.org (2020b)), spanning up a range of around 19,848 m, the available 24 bit (2^{24} different values) would allow a vertical resolution of 1.183 mm — most probably exceeding the accuracy of any accessible global-scale elevation measurement method by far.

However, as already described in Section 2.1.3, only *one* height value can be stored per x and y coordinate, as two-dimensional textures are used. Hence, through this method, overhangs, caves, bridges — among others — and even precisely vertical structures, cannot be modeled, as they would require at least one (and potentially more) "height values" per x/y coordinate.

Decoding

Elevation data, embedded within terrain tiles, needs prior decoding before it can be further processed. Decoding formulas vary among data providers: Height h (in meters) within Mapbox terrain tiles, for example, can be retrieved via the following formula (docs.mapbox.com, 2020a):

$$h = -10000 + (R * 256 * 256 + G * 256 + B) * 0.1$$

(a) Terrain tile
© Mapzen.com

(b) Basemap tile
© Toursprung.com

(c) Textured 2.5D terrain (Mt. Fuji). Image source: Prototype screenshot, basemap © Toursprung.com

Figure 3.5: The distinctive pattern of the heightmap (a) within the terrain tile is caused by the height values' encoding spread among all RGB channels of the image. A 2D plane mesh (not depicted), which is tessellated enough to provide reasonable resolution, is then displaced based on the decoded height values of (a). By applying the basemap texture (b) to this tessellated and displaced mesh, 2.5D terrain (c) is generated

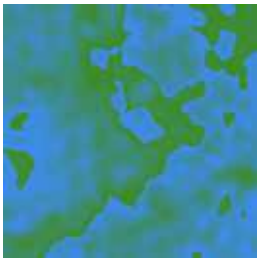
Mapzen's elevation data, needs to be decoded via (mapzen.com, 2016):

$$h = R * 256 + G + B/256 - 32768$$

Figure 3.5 illustrates the whole process of creating 2.5D terrain by starting with a 2D terrain tile, which is applied on a 2D plane and textured with the according basemap.

Data quality

Depending on the region and the data provider, quality varies. Before Mapzen elevation data was considered, tests with the Mapbox equivalent exposed problems. Since the data set is termed "Mapbox Terrain-RGB" (docs.mapbox.com, 2020a), one could assume a legit DTM, thus *terrain* elevation and explicitly no DSM, which includes buildings' and other artificial structures' heights. However, for tests within Vienna, situations like those depicted in Figure 3.6 occurred, where there are hills where flat plains should be. Berlin shows a similar picture while in New York City, everything seems all right, albeit the real terrain in NYC is — in fact — rather flat in the first place. Respective correspondence with Mapbox was unfruitful as the reply was the following (via email): *"Mapbox maps rely on data from OpenStreetMap, a global volunteer project that everyone can contribute to. If you are familiar with this area, I encourage you to go to openstreetmap.org, create an account, and make improvements directly."* Which is, however, wrong, as OSM "does not generally carry elevation data" (help.openstreetmap.org, 2011). Also, according to their own website (mapbox.com, 2019), OSM was not listed as a source for elevation



(a) Terrain tile © Mapbox.com

(b) Terrain tile overlaid with the respective basemap © Mapbox.com

(c) The 3D view shows hills where none should be. Image source: Prototype screenshot

Figure 3.6: Mapbox terrain data gives wrong results within the city of Vienna (probably, the supposed DTM is actually a DSM model in some areas), exposing the sampled height of buildings and other structures, instead of terrain

data (this section seems to be removed from the website as of January 2020): *"We processed the best elevation and landcover information we could find from around the world — 24 different datasets from 13 organizations, including the US Geological Survey, the Norwegian Mapping Authority, and the Canadian GeoBase data portal."* Confronting Mapbox with these facts led to no further reply.

Since this was unacceptable for the test case of Vienna and rendered integration of 2.5D terrain useless, an alternative was needed. It was found in Mapzen's terrain tiles (mapzen.com, 2016), which provide global coverage and — in contrast to Mapbox, where beyond a certain usage limit, tile access is subject to charge — are completely free to use. Furthermore, since they are offered via "Registry of Open Data on AWS" (registry.opendata.aws, 2020), the service should be reliable and fast. This data seems to be correct, as the hills depicted in Figure 3.6c are gone, and no buildings sink into the ground, all while the little hills surrounding Vienna (as shown in Figure 1.7) look valid. The quality of the data, however, has not been evaluated in-depth yet.

Retrieval

The tiles (such as the one depicted in Figure 3.6a), which cover the whole globe for zoom levels from 0 to 15, can be retrieved via the *AWS* endpoint `https://s3.amazonaws.com/elevation-tiles-prod/terrarium/{z}/{x}/{y}.png`, whereas `{z}` represents the zoom level, `{x}` and `{y}` the respective tile coordinates. For details regarding tile coordinates and zoom levels, refer to Section 3.1.3.

3.2.4 Vegetation

Vegetation, especially trees, provide shadows. This is well known, especially during hot summers in Vienna, where temperature levels easily reach 30° C or more. Vienna's OGD

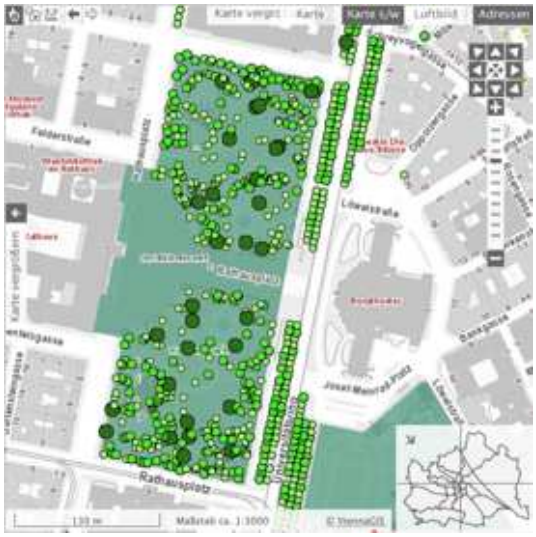
```
{
    "type": "Feature",
    "id": "BAUMKATOGD.fid-2a28b15c_1634012747b_-6bb9",
    "geometry": { "type": "Point", "coordinates": [16.27671731106308, 48.185618028039514] },
    "geometry_name": "SHAPE",
    "properties": {
        "OBJECTID": 165901664,
        "BAUM_ID": 111236,
        "DATENFUEHRUNG": "magistrat",
        "BEZIRK": 13,
        "OBJEKT_STRASSE": "Goldmarkplatz",
        "GEBIETSGRUPPE": "MA 28 - Straße, Grünanlage",
        "GATTUNG_ART": "Tilia platyphyllos (Sommerlinde)",
        "PFLANZJAHR": 1952,
        "PFLANZJAHR_TXT": "1952",
        "STAMMUMFANG": 181,
        "STAMMUMFANG_TXT": "181 cm",
        "BAUMHOEHE": 4,
        "BAUMHOEHE_TXT": "16-20 m",
        "KRONENDURCHMESSER": 3,
        "KRONENDURCHMESSER_TXT": "7-9 m",
        "BAUMNUMMER": "1005",
        "SE_ANNO_CAD_DATA": null
    }
}
```


Figure 3.7: Extract of Vienna's tree cadastre, covering the metadata of a "Tilia platyphyllos" (*Sommerlinde* in German or *large-leaved lime* in English). Highlighted values are considered for further processing, at foremost the approximation of a respective 3D model (data.gv.at, 2019a)

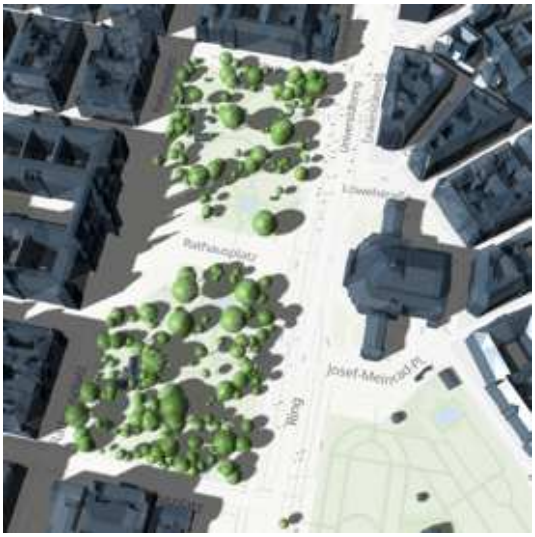
initiative provides access to its tree cadastre, a freely downloadable 127.3 MB JSON file (data.gv.at, 2019a), which was last updated in June 2019. "In addition to all roadside trees, the official tree register also includes some (but not all) trees growing in parks and wooded areas in Vienna." (ibid.) — which also means that trees within private properties are, therefore, not covered.

A typical JSON node of this data set, representing a tree, can be seen in Figure 3.7. According to unconfirmed sources, the file contains at least 200,000 such nodes (thus, trees); however, due to the sheer size of the file and since only extracts of it were processed yet, this cannot be confirmed. Although these trees are not explicitly defined 3D models, but instead described via vague attributes, they can still be used to create a three-dimensional approximation (similar to the aforementioned metadata-based construction of LOD2 roofs for 3D buildings). Therefore, the following attributes are considered:

- `geometry`: Contains a `coordinates`-array providing float values of the tree's *WGS 84* based latitude and longitude coordinates in order to position the tree correctly on a map.
- `stammumfang`, `baumhoehe`, `kronendurchmesser` ($circumfence_{stem}$, $height_{tree}$, $diameter_{crown}$): The crucial attributes to model a rough tree structure consisting of two main parts, as follows:



(a) Screenshot of Vienna's environment city plan, showing a part of its tree cadastre. Image source: Stadt Wien - ViennaGIS (www.wien.gv.at/viennagis/)



(b) Data depicted in (a) is used to create approximating 3D tree structures, capable of casting shadows. Image source: Prototype screenshot

Figure 3.8: 3D tree cadastre visualization

  - The stem as a cylinder with a radius $r_{stem} = circumfence_{stem}/(2 * \pi)$ and a height of $h_{stem} = height_{tree} - diameter_{crown}$
  - The crown as a sphere with radius $r_{crown} = diameter_{crown}/2$ vertically offset to $height_{tree} - r_{crown}$

- `gattung_art` (*species*): In theory, this can be used to even visualize distinctive features of various tree species, such as crown shape, or even leaves. Initially, however, it might just be used to colorize the crown of the tree accordingly.

- `pflanzjahr, baumnummer` (*yearofplanting*, *treenumber*): This data is just conserved as some users might be interested. It is, however, not relevant in regards to shadow visualization.

Following this approach, based on tree cadastre data (its 2D representation is depicted in Figure 3.8a), simple 3D tree structures can be modeled and visualized — capable of casting approximated shadows as shown in Figure 3.8b.

3.2.5 Basemap

The basemap was already discussed in Section 3.2.3, where it is described how its texture tiles are applied on the terrain-offset planes in order to finalize 2.5D terrain. Their sole purpose is to provide a classic map's navigational context, thereby communicating where

on the map one actually *is*. Depending on the data provider, basemaps can be configured completely arbitrarily; among possible attributes are colors, fonts, label sizes, icons of points of interest (PoI), or textures. These can be specified in a way that supports shadow visualization, e.g., by increasing contrasts between shaded and lit surfaces or by removing irrelevant noise from the texture.

Basemaps are then made available as either a) vector tiles or b) rasterized tiles. Vector tiles are tempting as they allow seamless zooms, provide perfectly sharp rendering at all and between zoom levels, load faster due to smaller size and allow distinctive features such as text labels that are always horizontal, no matter the map's orientation, as they're logically decoupled from the rest of the map. Since the prototype to be built, however, is 3D based, vector tiles are highly complicated to project on displaced terrain, which is why, for now, it is not further considered. Rasterized basemap tiles, however, can be retrieved via web-endpoints — just like their terrain equivalent.

The following basemap providers with worldwide coverage are considered:

- *Mapbox* (`www.mapbox.com`): Allows extensive map customization via its *Mapbox Studio*. Some pre-configured styles are available through a raster tile API which looks as follows: `https://api.mapbox.com/styles/v1/mapbox/streets-v9/tiles/{zoom}/{x}/{y}{@2x}.{format}?{access_token}`.

  The parameters `{zoom}`, `{x}` and `{y}` behave exactly like those of the aforementioned terrain tiles (Section 3.2.3), therefore, defining the zoom level, as well as the x- and y-coordinate of the tile. If `{@2x}` is added to the URL, the requested tile is delivered in doubled resolution (e.g., for high resolution (high-DPI) displays). The `{format}` specifier allows choosing between various image formats like color indexed PNGs or quality-reduced JPG, to save bandwidth. The last parameter is the `{access_token}`, the unique identifier of the receiving end for Mapbox accounting (as mentioned earlier, Mapbox eventually has to be paid). (docs.mapbox.com, 2020d)

- *Maptoolkit* (`www.maptoolkit.net`): Provides similar features as the Mapbox variant, albeit has a steeper learning curve in its map editor. An arrangement could be made with the data provider to allow free usage during the development process of the prototype, which is why the API endpoint cannot be published here. A major tradeoff for this special case, however, is that provided tiles are server-generated *on demand* and, thus, not pre-cached as it is the case with Mapbox. This will slow down the initial loading times of solar shadow map scenes.

For details regarding tile coordinates and zoom levels of basemap tiles, as with terrain, refer to Section 3.1.3.

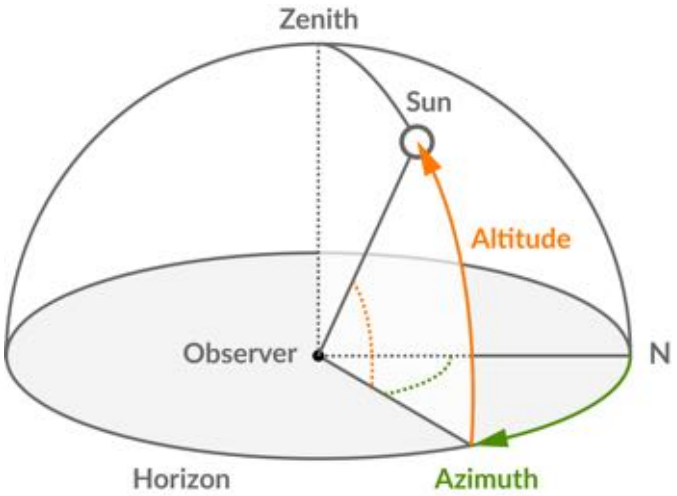


Figure 3.9: Azimuth is measured from the north and altitude refers to the vertical angle between the Earth's surface tangent and the Sun at the observer's position

3.2.6 The Sun

The Sun is the directional light source of a solar shadow map (for further discussion, refer to Section 2.1.5). Its position in a 3D scene is, therefore, highly relevant to simulate shadows that actually resemble reality. A method is needed that delivers this position as a function of a) location on Earth (in latitude and longitude coordinates) and b) time — while paying respect to all relevant astronomical and physical factors. A library providing this functionality is SunCalc (SunCalc, 2016). One of its core functions — `getPosition` — returns azimuth and altitude of the Sun for the parameters mentioned above. Azimuth refers to the horizontal angle of the Sun in the sky, measured from the north clockwise, while altitude is the vertical angle measured from the horizon (see Figure 3.9). According to a discussion in SunCalc (2016)'s GitHub repository, even atmospheric refraction is taken into account (github.com, 2016).

SunCalc (2016) defines the return values of its `getPosition` function as follows:

- "`altitude`: sun altitude above the horizon in radians, e.g. 0 at the horizon and PI/2 at the zenith (straight over your head)" (ibid.)
- "`azimuth`: sun azimuth in radians (direction along the horizon, measured from south to west), e.g. 0 is south and Math.PI * 3/4 is northwest" (ibid.)

Relative to a given observer point in 3D space on the Earth's surface, the position of a directional light, representing the Sun at the distance of radius r can be obtained through basic trigonometry. It must, however, be taken into account that the Sun's azimuth moves clockwise while sin and cos move counter-clockwise on the unit circle. Furthermore,

SunCalcs's azimuth of 0 is targeted towards the south (instead "east", where sin and cos start in the unit circle); thus, $\pi/2$ needs to be subtracted.

$$x = r * \cos(-azimuth - \pi/2)$$
$$y = r * \sin(-azimuth - \pi/2)$$
$$z = r * \tan(altitude)$$

3.3 Means of visualization

How could solar shadows based on three-dimensional data actually be visualized on the web? Section 3.1 ("Design considerations") already elaborated integration of heterogeneous data and how a containing 3D scene could look like, while Section 3.2 further discussed "Data" within this context. This section's purpose, however, lies in the evaluation of existing tools and platforms that might be adapted or directly used in order to achieve the desired functionality of a solar shadow map. Thus, it addresses the main research question *"1.) Is it technologically possible to interactively visualize solar shadows caused by terrain, vegetation, and buildings with sufficient precision in a web-based context? Therefore allowing an accessible, visually appealing map user experience?"*, especially focussing on the web-based context as well as the demanded interactivity, while the final answer to this question will be provided in Chapter 4.

This section, therefore, evaluates the suitability of web-based map rendering engines and more universal 3D engines. Since the aim was not to reinvent the wheel, the former were investigated first, as they provide a superior map-related feature set. However, as the constraints in areas as extensibility, performance, and flexibility were too limiting, it was necessary to go a level down and consider 3D engines.

3.3.1 Web-based map rendering engines

Back when the idea to work on this thesis surfaced, it was tempting to imagine a solution that works for the whole planet: If the problem of visualizing shadows interactively can be solved within a small region, the step to cover Earth might just be a small one. Map rendering engines provide Earth coverage as well as map-related features and UX — occasionally even some of the features, a shadow map requires. Taking an existing approach and extending it with lacking but required capabilities would be a huge time saver compared to creating something from scratch.

Initial requirements for an extensible web-based map rendering engine were:

- **Support for 3D buildings**
- **Support for 2.5D terrain**
- **Integrated shadow rendering with adequate precision, quality, and speed**

	3D Buildings	2.5D Terrain	Shadow rendering	Mobile UX & performance	Open source	Custom 3D Data	Vector Tiles
Mapzen Tangram	⊘	⊘	⊘	✓	✓	~	✓
Mapbox GL JS	⊘	⊘	⊘	✓	✓	~	✓
OSM Buildings	✓ OSM-based	⊘	✓	✓	✓	~	⊘
ArcGIS	✓	✓	✓	not tested	⊘	✓	⊘
CesiumJS	✓	✓	✓	?	✓	✓	⊘

Table 3.1: Five map rendering engines were reviewed against solar shadow map-related requirements. The two last columns contained optional features, while the first ones were mandatory. Critical but not fully supported features are highlighted

- **Appropriate mobile UX and performance**
- **Open-sourced to allow extensibility/customization**

Non-deal-breaking if unavailable, but nevertheless useful features would have been: **Support for custom 3D data (e.g., trees)**, as well as **vector tiles**. Five different mapping engines were tested against those requirements. Table 3.1 gives an overview of the results, which will be further investigated as follows.

Mapzen Tangram

Mapzen Tangram (mapzen.com, 2020) is an open-source WebGL based JavaScript map rendering engine that, by the time it was discovered during research for this thesis, had just gone out of business. A few demos, nonetheless, remained on the website, which were — at first glance — promising, especially considering their smoothness and speed, including on mobile devices. This can probably be attributed to Tangram's use of vector tiles (instead of raster tiles) as well as its optimization towards 2.5D buildings on flat surfaces (Figure 2.13a is actually a screenshot of one of their demos). The drawback of this approach, however, is the lack of support for "real" 3D buildings as well as terrain. Shadow rendering is not supported, but since the library is open-sourced, this could — in theory — be added subsequently. Nonetheless, since two significant requirements (3D buildings and 2.5D terrain) are not supported, this was not further considered. While *Tangram* seemed dead upon first discovery (i.e., there was no ongoing development or maintenance), it became a Linux Foundation Project in January 2019 (linuxfoundation.org, 2019) and, as of 2020, seems revived since then.

Mapbox GL JS

Similar to *Mapzen Tangram* in its appearance and UX is *Mapbox GL JS*, which is also written in JavaScript and uses WebGL for hardware-accelerated rendering (docs.mapbox.com, 2020b). Just as *Tangram*, it is, however, an engine optimized for 2.5D buildings on flat

surfaces and, thus, does not support 3D buildings or 2.5D terrain. There are ways to place arbitrary 3D models (such as LOD2 building blocks) into a Mapbox GL JS scene by extending it with a *Three.js*[1]-based *custom layer* (docs.mapbox.com, 2020c). It is still an open question how this strategy scales for large amounts of models, such as the simultaneous visibility of, e.g., whole parts of Vienna. But even if it worked well, Vienna's city model includes vertical offsets to blend into hilly terrain — which Mapbox GL JS does not support in the first place.
As Mapbox is a big player in the digital maps industry, the open-sourced Mapbox GL JS is probably here to stay. It is actively developed by Mapbox employees who also take part in open discussions about issues and feature requests. One of these asked for "Support 3D terrain meshes" (actually demanding 2.5D) (github.com/mapbox, 2015), a feature that is still not integrated five years later.

OSM Buildings

OSM Buildings (osmbuildings.org, 2020) actually started as a web-based viewer for 3D buildings, obtained from OSM data (refer to Section 2.1.3). Its distinctive feature is the consideration and actual 3D rendering of some OSM 3D building attributes, such as `roof:shape`, etc., as they are listed in OSM's online editor (see Figure 3.1). It later branched into a 2.5D map rendering engine, which was subsequently extended into a 3D version (osmbuildings.org, Solutions). The system additionally provides shadow rendering out of the box, which is why it was seriously considered as base technology to be extended to gain support for 2.5D terrain. Correspondence with the main developer behind *OSM Buildings* was established to evaluate the implementation effort. Gained insight, however, was not convincing as it probably would have required major changes to many parts of the rendering engine. The probability of reaching a dead-end was present, which is why this engine was kept in evidence for a while but eventually not further pursued.

ArcGIS Online

ArcGIS (arcgis.com, 2020) is a collection of commercial GIS applications. Among them is *ArcGIS Online*, a closed-source web-mapping platform. While it supports shadows and 2.5D terrain by default, the fact that it a) is not extensible due to unavailable source code and b) eventually costs money, led to the decision not to further consider it.

CesiumJS

CesiumJS is the open-source alternative to *ArcGIS*. According to their website (cesiumjs.org, 2020) it is an *"open source JavaScript library for creating world-class 3D globes and maps with the best possible performance, precision, visual quality, and ease of use. Developers across industries, from aerospace to smart cities to drones, use CesiumJS to create interactive web apps for sharing dynamic geospatial data."* (ibid.). Its feature set is vast: Not only does it support 2.5D terrain, it also allows the embedding of maps

[1] Three.js is a web-based 3D engine which will be discussed in Section 3.3.2.

Figure 3.10: Screenshot of Vienna's "Stadtplan3D" (wien.gv.at/stadtplan3d, 2020), based on *CesiumJS*. As the latter was reviewed in 2018 for the first time on mobile devices, all tested demos ran on half resolution on high-DPI devices (which as of 2020 seems to have changed). Shadow quality was low as well: unfiltered, low resolution (jagged), and strong temporal artifacts were omnipresent. Image source: Stadt Wien - ViennaGIS (www.wien.gv.at/viennagis/)

onto Earth's globe, rendered as a 3D spheroid. As Table 3.1 shows, support includes shadows and 3D buildings.

There is, however, a question mark in Table 3.1's column of "Mobile UX & performance": Back in 2018, when work on this thesis took of, and CesiumJS was first tested, it did not seem visually appealing, and its UX and UI appeared outdated. The renderer did not support high-DPI displays and looked blurry, shadows were jagged and had massive "shadow acne", especially on mobile hardware (refer to Figure 2.12d). Vienna's "Stadtplan3D" (refer to Section 2.2.3), built on CesiumJS, produces visual quality that is below what is possible today on the web (see Figure 3.10). Since *CesiumJS* is open-source, similar to *OSM Buildings*, the question raised whether adaptations could be made in an efficient manner to improve this situation. However, in an unfortunate potential scenario, rendering quality was only so low, to allow interactive frame rates in the first place. And as soon as improvements are made in these regards, frame rates would drop significantly. These doubts were reassured by conversations that were had with companies like *Toursprung.com*, who tried to work on a performant 3D mapping of trekking tours and despaired at the sheer size of CesiumJS that, especially in a mobile context, seemed too bloated to allow fast and efficient rendering.

Discussion

None of the map rendering engines available in 2018 were fully convincing considering a solar shadow map context: *Mapzen Tangram*, *Mapbox GL JS*, and *OSM Buildings* lacked support for required 2.5D terrain, and the former two were not even designed to handle real 3D buildings. *ArcGIS* was not free to use and closed-source, and while *CesiumJS* basically fulfilled all the requirements, it could not deliver in terms of visual quality, and its vast complexity rendered potential customization intimidating. The other contestants, however, demonstrated visual and UX-related potential (full resolution, high FPS, fast loading) in a web-context, so why should one settle with anything less. In regards to shadow visualization, it seemed like there is much more possible given modern hardware and WebGL support, not to mention *time-integration* of shadows. The tested map rendering engines conclusively showed that all the desired features are actually achievable — even if none of them implemented them all. By going a level deeper into web-based 3D engines and creating a tool from scratch that fulfills all requirements, it can be tested, whether the result is still a) precise, b) fast (interactively usable) and c) visually appealing (refer to the main research question in Section 1.2, as well as at the beginning of this main section).

3.3.2 Web-based 3D engines

3D map engines are usually built upon underlying 3D engines. Hence, if more control about the outcome is required, than a map engine provides, the logical consequence is to go a level deeper. *WebGL* was frequently mentioned in this thesis yet. It is the JavaScript-based web port of *OpenGL*, a widely supported graphics API that, foremost, enables hardware (GPU) accelerated 3D rendering. Before WebGL, graphics hardware was not directly accessible out of web-contexts; hence, 3D scenes were heavily constrained in quality, complexity, and achievable frame rates. WebGL is comparatively low-level; thus, it does not supply simple methods or structures to create an interactive 3D scene. Instead, it merely provides a basic set of functions that eventually draw and illuminate lines and triangles. Any abstracting logic, which is nevertheless necessary for a (complex) scene beyond such basics, needs to be implemented in addition. This is where 3D engines come into play.

Three.js

An example is *Three.js* (threejs.org, 2020b), a JavaScript-based 3D library that abstracts away low-level WebGL commands, and provides logical entities that simplify the creation and management of a 3D scene instead. Figure 3.11 depicts the basic structure of a Three.js application: The *Renderer*'s job is to visualize the *Scene* from the *Camera*'s (observer's) perspective. The *Scene*, a tree graph (often referred to as *scenegraph*), contains all relevant objects to be rendered, as well as one or more light sources — otherwise, all would be black. The *Mesh* objects themselves refer to *Geometry* and *Material*, which can be either loaded from external resources or procedurally created.

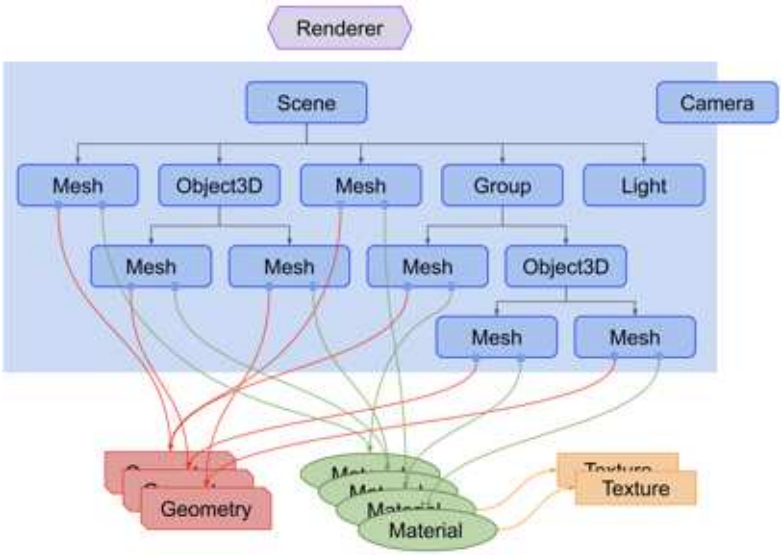


Figure 3.11: Structure of a typical Three.js scene. Image source: threejsfundamentals.org © Google Inc.

Applying this structure, a solar shadow map could be schematized as follows:

- The *Camera* is positioned and oriented in accordance with the clipping of the map, a user wants to see. This includes x/y coordinates, as well as *zoom* (z coordinate) and the potential tilting. The position of the camera's target defines the visible area and the current location on the map, which in turn specifies the Sun's position in the sky. The camera's distance to the scene's center (which lies on the map's surface) corresponds with the *zoom* level and the tiles to be displayed: The closer the camera, the higher the *zoom* and the smaller (i.e., more detailed) the tiles.

- Within the *Scene*, the Sun is represented through a *Light*-source and positioned according to desired time and location on Earth. Due to planar projection, as already discussed in previous sections, the Sun's position dynamically changes as the camera target (i.e., position on the map) changes.

- Besides the *Light*, the *Scene* consists of various *Mesh* objects, such as textured terrain tiles, building models and trees. Their resources (geometry and textures) are downloaded from servers on demand.

- Eventually, the *Renderer* puts everything together and draws the scene from the *Camera*'s view, lit and shadowed by the *Light*-source according to specified quality parameters, such as resolution, shadow rendering method, anti-aliasing, or post-processing effects.

A setup like this encompasses all the required elements of a solar shadow map. The next section will present some required algorithms, while Chapter 4 will go in-depth with the actual implementation.

Alternatives

Besides *Three.js*, another 3D engine — actually more of a *game* engine — that was further considered was *Unity* (unity.com, 2020). Back in 2018, a promising bridge between Mapbox and Unity was introduced (mapbox.com, Maps for Unity) that allowed to directly load Mapbox data (streets, buildings, and elevation) into a Unity 3D-scene. This would have provided a) automated integration of global terrain data, b) LOD1 building data (almost global, or as much as OSM covers), and c) Unity-built-in shadow rendering features. However, Unity — in contrast to Three.js — is not a web-first 3D engine, and instead directly targets operating systems of gaming consoles, mobile devices and desktop computers. It, however, provides a "WebGL Player" (docs.unity3d.com, 2019), which — admittedly — was not considered (or maybe even not available) in 2018. There was, however, a "Unity Webplayer" (unity3d.com/de/webplayer, 2018) whose support was already retracted at the time. Given the questionable web support and the reduction of buildings data to LOD1, the decision was made on Three.js.

3.4 Algorithms

This section covers and discusses algorithms that were designed during the process of creating a 3D solar shadow map prototype.

3.4.1 3D tile flood fill

In order to achieve a "slippy map" in 3D (refer to Section 3.1.2), 3D tiles need to be created, which fill up the view plane. Compared to a 2D map, the observer's viewing angle does not need to be normal towards the map's surface but can be arbitrarily tilted. To fill up the scene and not have any holes, i.e., missing tiles, a classic non-recursive *flood fill* algorithm (Algorithm 3.1) was the inspiration to create an adaptation — termed *3D tile flood fill* — that fills up the currently visible area with 3D tiles until it is completely covered, starting from the very center of the current view, as illustrated in Figure 3.12.

Flood fill is the classic algorithm used for a "bucket fill" tool in paint programs: An initial pixel and a replacement-color are defined. The pixel's color (*target-color*, c_t) and those of its neighboring pixels are then changed into *replacement-color* c_r, *unless their color differs from target-color*. The algorithm continues with neighboring pixels until all pixels, that are neighboring and initially had the *target-color*, got the new color (*replacement-color* applied. The definition of "neighboring" is arbitrary: 4-way considers only horizontal and vertical neighbors, while 8-way also considers diagonally aligned pixels.

The concept of a *3D tile flood fill* of the current view plane takes this basic idea and introduces adaptations:

Algorithm 3.1: Non-recursive queue based 4-way flood fill

Input: A a pixel-coordinate $\mathbf{P} = (p_x, p_y)$ and a replacement-color c_r

```
1  c_t ← getColorOfPixel(P);
2  if c_t = c_r then
3  |  return ;
4  end
5  changeColorOfPixel(P, c_r);
6  queue ← [P];
7  while length(queue) > 0 do
8  |  (p_x, p_y) ← getAndRemoveFirstPixelOfQueue(queue);
9  |  P_top ← getPixelAt(p_x, p_y − 1);
10 |  if getColorOfPixel(P_top) = c_t then
11 |  |  changeColorOfPixel(P_top, c_r);
12 |  |  pushIntoQueue(queue, P_top);
13 |  end
14 |  P_right ← getPixelAt(p_x + 1, p_y);
15 |  if getColorOfPixel(P_right) = c_t then
16 |  |  changeColorOfPixel(P_right, c_r);
17 |  |  pushIntoQueue(queue, P_right);
18 |  end
19 |  P_bottom ← getPixelAt(p_x, p_y + 1);
20 |  if getColorOfPixel(P_bottom) = c_t then
21 |  |  changeColorOfPixel(P_bottom, c_r);
22 |  |  pushIntoQueue(queue, P_bottom);
23 |  end
24 |  P_left ← getPixelAt(p_x − 1, p_y);
25 |  if getColorOfPixel(P_left) = c_t then
26 |  |  changeColorOfPixel(P_left, c_r);
27 |  |  pushIntoQueue(queue, P_left);
28 |  end
29 end
30 return ;
```

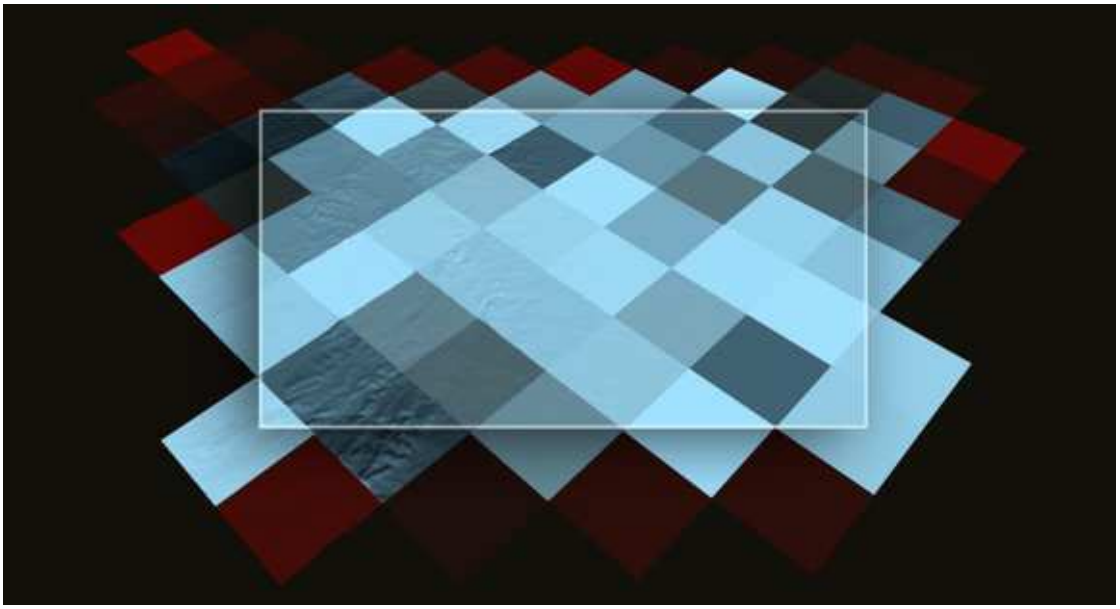


Figure 3.12: An adapted flood fill algorithm is used to fill the view plane with terrain tiles. The highlighted rectangle depicts the visible view frustum (consisting of bright blue tiles, which are at least partially intersecting the view plane) while the dark red tiles are their neighboring tiles, where the algorithm eventually stops (the view plane is slightly distorted due to the changed field of view after zooming out)

- Instead of pixels, the considered objects are *tiles*
- It always starts at the *center tile* of the current view. The center tile x/y indices are retrieved by converting the current *Lat./Lng.* coordinates and the *zoom* level into tile indices using the formulas in Section 3.1.3
- The critical part is that instead of comparing color whatsoever, it is checked whether the current tile *intersects the view frustum*. This visibility check is done with a primitive empty polygon to reduce the required computation.
- If the tile is visible, it is initialized by loading all required data (basemap texture, terrain texture) and displacing the flat 2D plane accordingly, before it is added to the scene. The tile is also pushed into a *queue*, just like in the flood fill example.
- Now, the loop pattern of the flood fill example is applied and executed until the queue is empty. Instead of pixel coordinates, tile coordinates are used.

The advantages of this approach are that only the tiles which are currently visible in the scene are loaded and added to the scene graph. This saves a) download time and b) rendering resources. Similar techniques can also be applied to the loading of buildings — which in the case of Vienna's data set are also divided into tiles, albeit with different

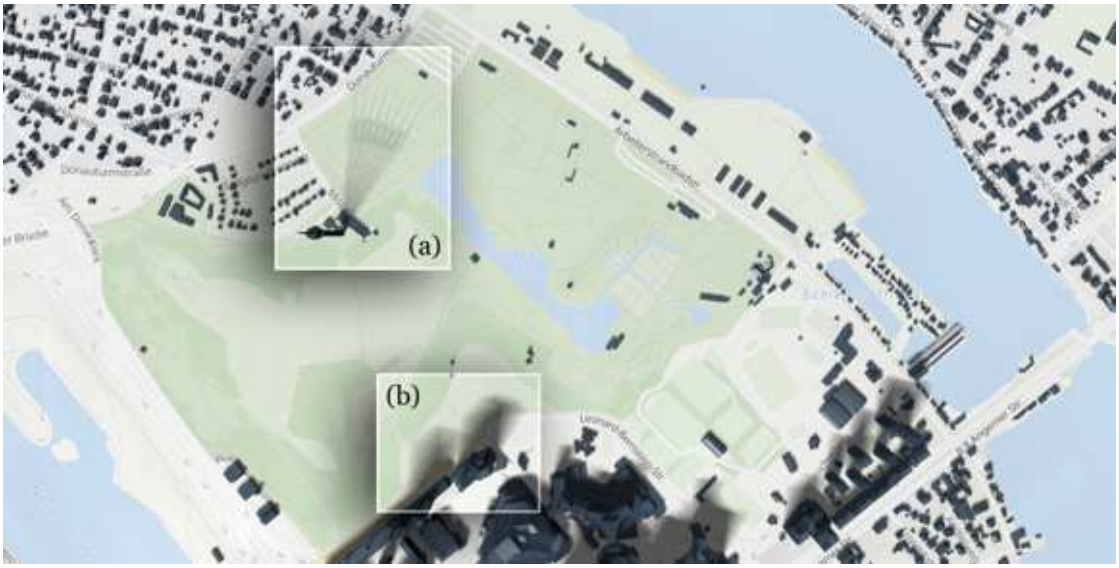


Figure 3.13: Using a naive approach, time-integrated shadows can be rendered in a web-context using shadow mapping and n light sources. The rendering shows a part of Vienna around noon and covers the time frame of 1 hour while using $n = 8$ light sources. In (a), due to the structure's high and thin shape ("Donauturm"), the shadows can even be counted. For more regular structures, however, as in (b), overlapping between the shadows produces convincing results. Image source: Prototype screenshot

divisions — and vegetation. To limit the larger amount of loaded and rendered tiles, as the camera tilt increases, either the latter could be constrained (i.e., a tilting low enough to maintain a maximum number of tiles), or the radius describing the circular area, where tiles should be loaded, is limited. In the case of the latter, the visible range can be reduced accordingly by using *fog*. A more sophisticated approach would, furthermore, employ a lower LOD for tiles further away from the camera.

3.4.2 Time-integration of shadows

As it was discussed in Section 2.2.1, "Shadow accrual maps", their sophisticated and visually convincing approach on rendering temporal shadows cannot currently be achieved within a web-context. Nevertheless, a shadow mapping based (see Section 2.1.2) attempt is made to provide a naive version that — while significantly impacting rendering speed — gives an immediate idea of how a shadow situation changes over time.

By arranging n directional light sources along to the Sun's path over time (n equals the number of integration steps) and reducing their light intensity down to $1/n$, visualizations, as shown in Figure 3.13 can be achieved. This is done by calculating the Sun's azimuth and altitude at the initial time (t_{start}), as well as at $t_{end} = t_{start} + offset$ using *SunCalc*'s functions, as elaborated in Section 3.2.6), whereas $offset$ can be any duration less

than 24 hours and small enough for t_{end} to be prior sunset. Thus, this approach — compared to "Shadow accrual maps" — is only reasonable for time-integration within a day. Afterwards, a step-angle is calculated both for azimuth and altitude to distribute light sources evenly. Algorithm 3.2 describes the process in detail for directional light sources by positioning the lights' origins and targets accordingly (again, using formulas introduced at the end of Section 3.2.6).[2] To avoid numerical instabilities, it is vital to keep the camera of the probably vast 3D scene a solar shadow map consists of, near or at the *origin* of the scene (refer to Section 4.2.1 for more details). This is the reason why the sunlights' targets in Algorithm 3.2 are all set to $(0, 0, 0)$ — and the same applies for the camera (observer) target, as only the scene's other objects are moved inversely when the map is panned.

Algorithm 3.2: Positioning of n light sources to visualize time-integrated shadows between t_{start} and t_{end}

Input: The Sun's azimuth and altitude at t_{start}: az_{start}, alt_{start}, the Sun's azimuth and altitude at t_{end}: az_{end}, alt_{end}, the number of integration steps n and the radius between the light source and its target r

if $n < 2$ **then**
  return ;
end
$angleStepsize_{az} = (az_{end} - az_{start})/(n-1)$;
$angleStepsize_{alt} = (alt_{end} - alt_{start})/(n-1)$;
$lightIntensity = 1/n$;
for $i \leftarrow 0$ **to** $n-1$ **do**
  $light \leftarrow$ createDirectionalLightWithIntensity($lightIntensity$);
  setLightTargetPos($light, (0, 0, 0)$);
  $target_x \leftarrow r * \cos(-(az_{start} + angleStepsize_{az} * i) - \pi/2)$;
  $target_y \leftarrow r * \sin(-(az_{start} + angleStepsize_{az} * i) - \pi/2)$;
  $target_z \leftarrow r * \tan(alt_{start} + angleStepsize_{alt} * i)$;
  setLightOriginPos($light, (target_x, target_y, target_z)$);
end
return ;

[2] As Three.js does not allow to set a light source's orientation, origin, and target locations need to be set. The light source's direction is, thus, a result of both positions.

CHAPTER 4

Implementation & Reflection

"If you wish to make an apple pie from scratch,
you must first invent the universe."

— Carl Sagan

This chapter will provide insight into the concrete implementation of a solar shadow map in *JavaScript* using *Three.js* as a 3D engine. It will demonstrate that it is, in fact, possible to *"interactively visualize solar shadows caused by terrain, vegetation, and buildings with sufficient precision in a web-based context"*, allowing an *"accessible, visually appealing map user experience"* (quoting the main research question as it was asked in Chapter 1, Section 1.2).

Similar to the previous chapter, its structure is divided into the main sections *Data* and *Visualization* — completed by a *Qualitative evaluation*, which will compare the implemented simulation with the ground truth. As the former is inherently an approximation of reality, shadow scenarios visualized by the created prototype shall be qualitatively compared to their real-life counterparts: Outdoor pictures of Vienna that contain visible shadows, backed by Exchangeable Image File Format (EXIF) data, which include creation-date and -time of the photo. As a result, the research question shall be answered, whether the implemented simulation resembles reality well enough to be of actual use.

4.1 Data

Actual implementations in regards to retrieval and compression of data, coordinate reprojection, and other means of data processing will be discussed as follows.

4.1.1 Initial data retrieval

This section describes initial data retrieval processes of Vienna's OGD data which, per se, is not structured and optimized for direct and dynamic usage within a solar shadow

map. It, therefore, requires prior processing (among others, conversion, compression, and splitting) to become suitable for a web-context.

Vienna's LOD2 city model

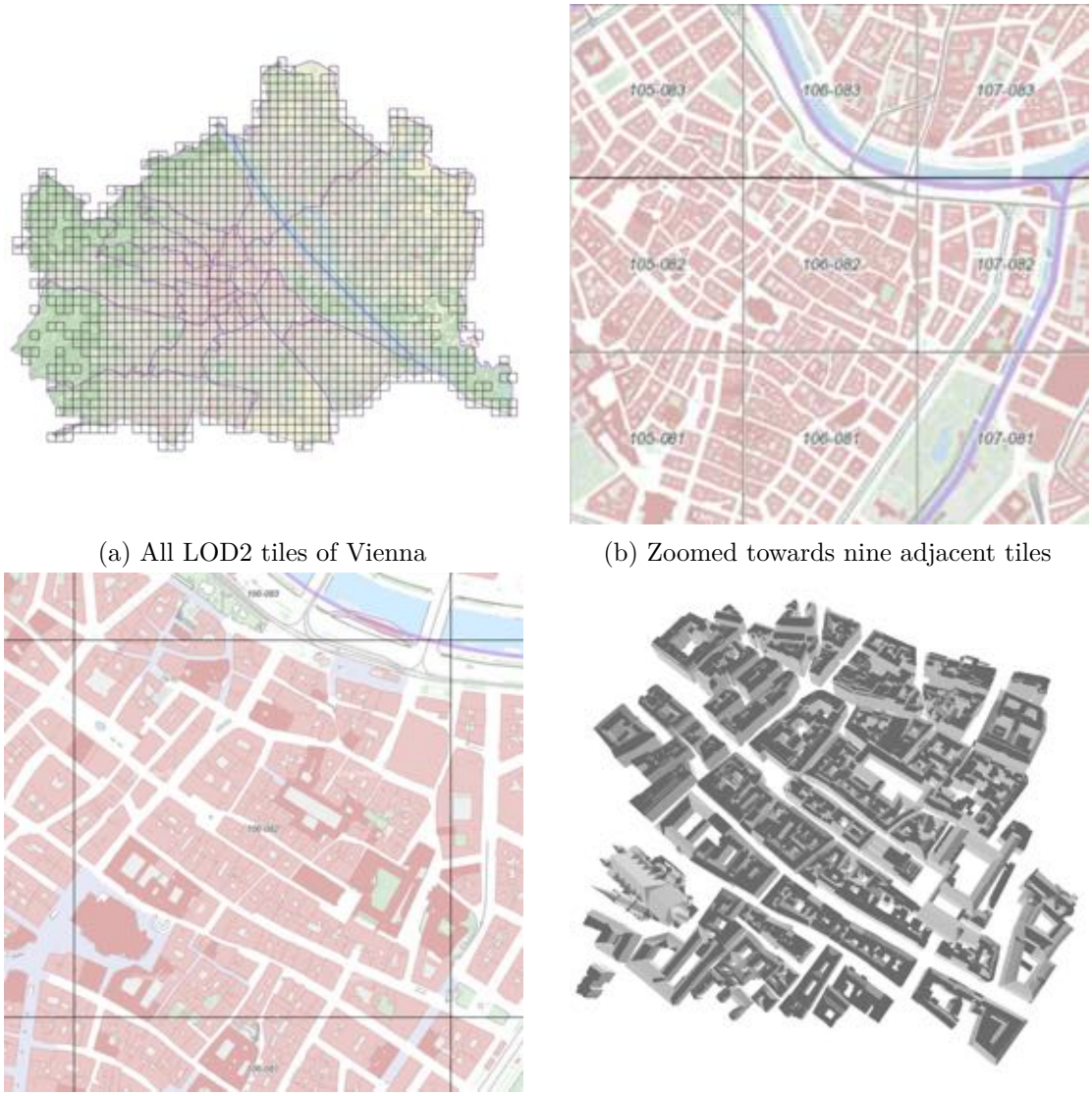
As mentioned in the previous chapter, the city of Vienna maintains an online "Geodatenviewer" (geodata viewer), representing the interface to retrieve various OGD, including the city's LOD2 building model. The LOD2 is organized in *tiles*, covering the whole of Vienna. Figure 4.1a shows the layout, in which every black square represents an available tile. There are some regions (e.g., in the west), where no tiles are available, as there are assumingly no buildings to consider (which needs further evaluation, nevertheless). As shown in Figure 4.1, the "Geodatenviewer" is a slippy and zoomable map, and by clicking on a tile, the user is confronted with the choice of either downloading a *CityGML* or *DXF* variant of the 3D tile, which looks like Figure 4.1d — hence a not sharply separated area (i.e., full buildings often cross borders into another tile's area), containing 3D meshes of building structures.

There is no interface to download all the LOD2 tiles of Vienna at once. However, since the "Geodatenviewer" exposes the single tiles' URLs, a script was written that iterates through the x and y coordinates, and retrieves all tiles (ZIP-compressed files) among the rectangular area, starting from the most southwestern to the most northeastern tile. Since Vienna is not a perfect rectangle, this means that some URLs cannot be resolved, and end up in empty files that need to be filtered out — ending up in 1,404 usable files. Since the underlying data contains 3D vertices and no rasterized data such as pixels, there is no requirement for *zoom*-related indexing, as it is for basemap and terrain raster tiles.

The data set's metadata page (data.gv.at, 2019b) mentions that the update frequency of the model is irregular. A related attribute "temporal extension (start)" is set to 2013: Since it lists the same year as (wien.gv.at, LOD2), it assumingly describes the date of the last update, as there is also no "temporal extension (end)" equivalent (or any other date attribute) available. There is a link to a JSON of all metadata (including a `begin_datetime` attribute), which might be considered in the future to become aware of an updated city model (data.gv.at, 2019b). This metadata, however, most probably will not tell *which* tiles were updated; thus, requiring a complete download and replacement of the whole model to keep it up-to-date. Since there was no update besides the initial offering yet, there is, nevertheless, no definite knowledge on how the data model or its metadata reflect such an event.

Tree cadastre

In contrast to the building model, which is split into 1,404 files, the tree cadastre is a single JSON file (127 MB) offered as a direct download from its metadata info page (data.gv.at, 2019a). It covers the whole of Vienna and, as presented in Section 3.2.4, probably contains metadata of roughly 200,000 trees. The amount of actual trees is,



(a) All LOD2 tiles of Vienna

(b) Zoomed towards nine adjacent tiles

(c) A single LOD2 tile

(d) The 3D model contained in a single tile

Figure 4.1: LOD2 coverage of Vienna within the city's "Geodatenviewer". Image/data sources: Stadt Wien - ViennaGIS (www.wien.gv.at/viennagis/)

however, unconfirmed, as direct processing of the file reaches the limits of available editors and hardware, and, therefore, requires other strategies, such as transferring the data into a database.

For the first tests, a fraction of the data (1,999 trees) was extracted and cleaned from unnecessary metadata to reduce file size. Within the JSON file, there seems to be partial ordering (i.e., frequently, neighboring trees seem to be adjacent in the file as well) — but only on a local scale. It is evident that the provided data requires structure (e.g., by splitting it in accordance with LOD2 building tiles) to allow flexible usage within a solar shadow map context. Hence, for the prototype, only the mentioned subset of 1,999 trees is considered for shadow visualization and, eventually, loaded into the scene as a whole.

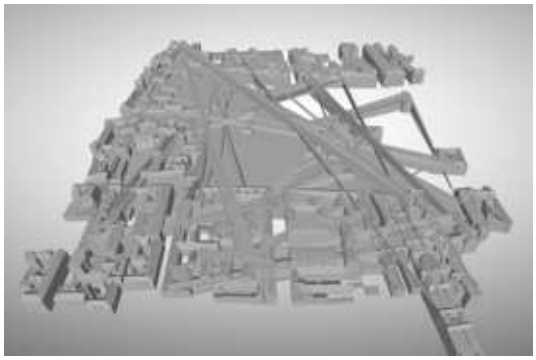
Suggested future work to compensate for shortcomings

- *By third parties (i.e., municipals of Vienna):*
  - An API that allows direct download of a) the complete LOD2 city model, b) only updated tiles within a date/time-range and c) tiles based on their x/y index, as this would significantly improve the process of both initial downloading of the model, and keeping it up to date during later stages.
  - Clarify attributes on OGD metadata pages.
  - Update the provided data at more frequent intervals.
- *Internal:*
  - Implementation of an update mechanism that considers the LOD2's metadata JSON to gain awareness about changes in the city model.
  - Introduce hierarchical structure into the tree cadastre file. A reasonable strategy would be to split the data set up into smaller JSON files that match the city tiles' dimensions.

4.1.2 3D mesh compression

The notable file size of Vienna's LOD2 model was already mentioned before: Its CityGML variant reaches 11 GB (unzipped) while the DXF version still requires 9.3 GB. Given 1,404 available tiles, this averages at around 6.6 MB per tile — too big for reasonable UX, as required data transfer would a) take too long and b) consume too many resources (which becomes especially relevant on mobile devices). Lossy data compression was considered while keeping compatibility with *Three.js* model loaders in mind.

Draco (github.com, Draco) was the first tested compression library. Since it does not support *CityGML* as an input format, the tool *CityGML2OBJs* (Biljecki and Arroyo Ohori, 2015) was used to convert the model into the *Wavefront OBJ (OBJ)* 3D file format which, unfortunately, was not able to produce *triangulated* output meshes from the provided source files. Since *Draco* requires them triangulated, however, the whole conversion



(a) Usage of any *compressionLevel* > 0 destroys the mesh

(b) Reduction of *quantizePositionBits* produces visible distortion along straight lines and on roof structures (highlighted)

Figure 4.2: Artifacts created by lossy *glTF Pipeline* compression. Data source: Stadt Wien - ViennaGIS (www.wien.gv.at/viennagis/)

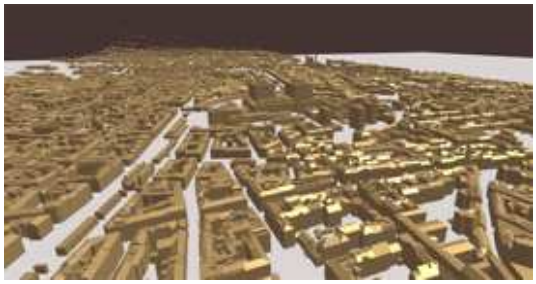
and compression pipeline was rendered useless. On the positive side, nevertheless, was a dramatic size reduction of the mesh from 11 GB down to 778 MB, even without compression. A significant role in this probably played the removal of CityGML's metadata.

glTF pipeline

Another tool, *glTF Pipeline*, provides a collection of "content pipeline tools for optimizing glTF assets" (github.com, glTF Pipeline), including *Draco* as well. Compression with the same non-triangulated *OBJ* mesh as before suddenly worked, albeit with insufficient results. Any value larger than 0 for the `compressionLevel`-attribute basically destroys the mesh (Figure 4.2a shows the result for `compressionLevel = 1`, which reduced the model's size from 2.2 MB to 421 KB — but also rendered it unusable). However, even at a low compression level of 0, the model's size went down from 2.2 MB to 1.6 MB. File size was further brought down by lowering `quantizePositionBits`: After reducing the default 14 bits down to 8, there was visible distortion in the model, though; straight edges along adjacent buildings became wavy, roof structures got disordered (see Figure 4.2b) — all for mere 100 KB of size reduction as the model still measured 1.5 MB.

Draco

The single reason why *Draco* was not further investigated yet, was the lack of a triangulated *OBJ* city mesh. Hence, *Blender*, an open-source 3D modeling software, was used to triangulate a sample city tile manually. Afterwards, the file could indeed be processed by *Draco* and not only provided visually convincing results (i.e., there were no compression artifacts recognizable) but also reduced the file size of the 5.9 MB (triangulated) mesh to 369 KB. This was not only the best compression ratio achieved yet but also had no



(a) *Draco* compressed city models appear flawless without recognizable artifacts

(b) During initial tests, several hundreds of *Draco* compressed Vienna city tiles could be rendered simultaneously within a *Three.js*-scene at interactive frame rates

Figure 4.3: *Draco* compressed meshes in *Three.js*. Data source: Stadt Wien - ViennaGIS (www.wien.gv.at/viennagis/)

visual drawbacks. However, batch-processed triangulation of 1,404 city tiles bore the risk of introduced artifacts or defects among the meshes, which would require further quality evaluation. The aim, therefore, was to leave the mesh as close to its original state as possible.

What eventually made the conversion and compression pipeline complete, thus, was the switch from Vienna's *CityGML* model to its *DXF* version: Using *Autodesk FBX Converter 2013.3* all 1,404 city meshes could be batch converted into *FBX*, whereas the same tool provided further batch conversion into *OBJ* — all while maintaining features and triangulation. *Draco* now was able to flawlessly batch-compress all tiles into substantially file size-reduced *Draco* meshes (.drc files) which — given a provided *Three.js* compatible *Draco* mesh loader — could also be added into a *Three.js* 3D scene (Figure 4.3). During the various conversion steps, file size of the full city model underwent the following steps: 9.3 GB (DXF) to 1.5 GB (FBX) to 1.4 GB (OBJ) to *47 MB*; a drastic 99.5 % reduction from the original DXF source and still a notable 96.7 % reduction compared to the OBJ variant.

Corto

Another aspect of mesh compression, besides the compression ratio, is the required computation time for encoding and especially decoding, as the latter needs to happen during runtime on the client. This is why another mesh compression library — *Corto* — was considered more closely, as it lists impressive stats in these regards (github.com, Corto). Actual tests, however, provided mediocre results (see Figure 4.4): The model visibly suffered from the lossy compression, which, on top, produced a bigger file than *Draco*. While further testing and adjustment of *Corto*'s compression parameters assumingly would have delivered better results, this was, nevertheless, not further pursued — much to the fact that *Draco* provided more than impressive results already out of the box.

Figure 4.4: Compression with *Corto* resulted in wavy edges where straight lines should be, visible throughout the model but especially within the highlighted region. Data source: Stadt Wien - ViennaGIS (www.wien.gv.at/viennagis/)

Suggested future work to compensate for shortcomings

- *Internal:*
  - Even though the *Draco* compressed meshes appear visually flawless, their quality was not investigated analytically yet. This should be done in order to confirm that lossy compression of a 3D city model is a valid approach.

4.1.3 Coordinate reprojection

Aspects of coordinate systems, including Vienna's use of the local *Gauß-Krüger in Meridian 34* (EPSG:31256), were introduced in Section 3.1.3. Intending to integrate Vienna's 3D city model into the same 3D scene as basemap, terrain tiles, and vegetation, reprojection becomes a requirement.

GIS-based workflow

The initial strategy was to reproject all 1,404 tiles' coordinates into *WGS 84* (EPSG:4326)[1] using *QGIS*, a free and open-sourced GIS tool (Figure 4.5), thus, embedding the new coordinate system right into the source files. *QGIS*, however, labeled the DXF source "invalid" and refused further processing. Attempts to fix the model (using community-recommended *GRASS/v.clean* or *QGIS* built-in *Vector geometry/Repair geometry*) tools were unsuccessful: The former was unable to execute while the latter found errors in the whole model and removed many relevant and visible triangles in the process.

Proj4js

Proj4js (proj4js.org, 2020) is the JavaScript port of *PROJ* (`www.proj.org`), a generic coordinate transformation tool. Its primary function is the reprojection of points from one coordinate system into another, specified by EPSG codes and related attributes. In

[1]The reason for *WGS 84* over *Spherical Mercator* lies in the fact, that the tool to further convert geo-coordinates into pixel coordinates relies on WGS 84 input

Figure 4.5: A 3D city model tile overlaid on Vienna's OSM basemap in *QGIS*

theory, a precise approach would reproject *all* vertices of a 3D city tile into the target coordinate system to maintain their relation to each other. If *Proj4js* ran on the client's device, this would take significant time, hence, slow down the initialization of the visible city model.
It can be argued, however, that the coverage of a single city tile is, in fact, small (i.e., spanning several blocks in both directions): Therefore, a reprojection of all its vertices might not be necessary, as potential misalignments are probably not even recognizable. Thus, the suggested approach is only to *reproject* a single point, i.e., the center of a city model tile's bounding box, and merely *move* the rest of the model along with the reprojected point. This way, a tile is placed at its correct position, deviation of vertices *within* a single tile are negligible, and adjacent city tiles are placed precisely next to each other and in visually acceptable alignment with the basemap, as Figure 4.6 shows. There are, nevertheless, minor deviations, but it is not known whether the imprecision is actually caused by the reprojected tiles or just originates from the basemap.

Since the library to finally place 3D objects in the composed solar shadow map, *"sphericalmercator"* (github.com, 2019), as introduced in Section 3.1.3, relies on *WGS 84* coordinates, *Proj4js* is configured to convert from EPSG:31256 (*Gauß-Krüger*) into EPSG:4326 (*WGS 84*). Their respective *Proj4js* definitions are as follows (epsg.io, 2020):

- **EPSG:31256:**
```
+proj=tmerc +lat_0=0 +lon_0=16.33333333333333
+k=1 +x_0=0 +y_0=-5000000 +ellps=bessel
+towgs84=577.326,90.129,463.919,5.137,1.474,5.297,2.4232
+units=m +no_defs
```

- **EPSG:4326:**
  `+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs`

Suggested future work to compensate for shortcomings

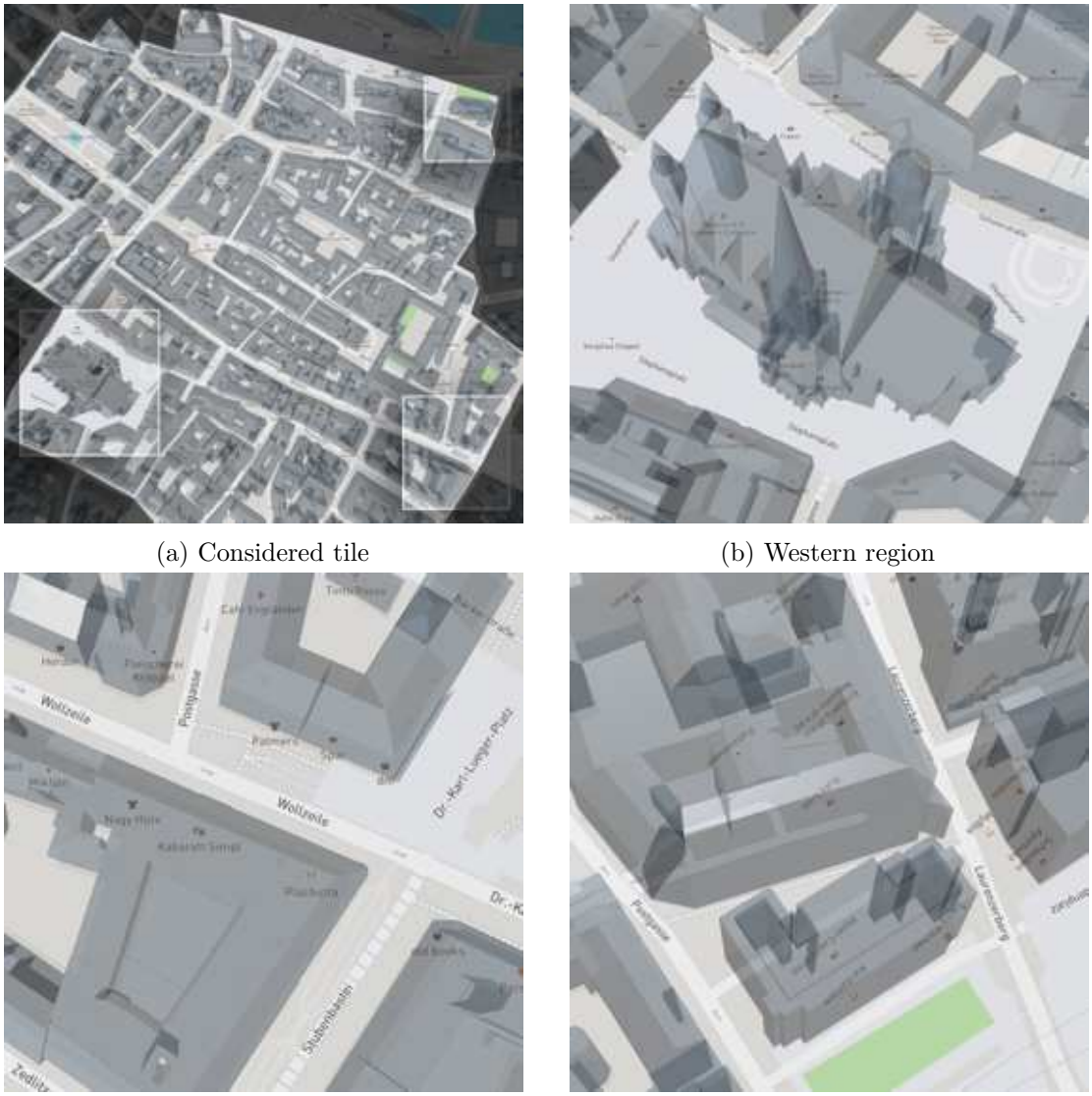
- *Internal:*
  - As of now, the quality of the single-point reprojection of 3D city tiles was only visually tested. An actual measurement of projection errors could validate/invalidate the chosen approach.
  - Instead of online reprojection of a single point, the source DXF files could be edited a priori. Even though there was no software found which was able to process available data, it can be assumed that there is a solution. The last resort would be a custom implementation.

4.1.4 Dynamic data retrieval

In contrast to initial data retrieval (see Section 4.1.1), data like basemap and terrain tiles are dynamically and directly loaded from respective web servers (refer to Section 3.2.3 as well as 3.2.5). Furthermore, data that underwent necessary preprocessing, like the previously discussed 3D city model tiles (also refer to Section 4.1.1), is moved to web servers as well to be dynamically retrieved from the developed prototype later. This becomes mandatory given the presented prototype's nature of a "slippy map", which, in this case, includes full coverage of Earth's surface. While interacting with the map, required data is continuously loaded, processed and added to the scene (refer to the description of the flood filling inspired approach in Section 3.4.1) in the background: In the best case, this happens without blocking or freezing of any user interaction, thanks to modern JavaScript features, such as *async/await* for asynchronicity and *web workers* for multithreading.

A typical process being triggered by an observer-view change looks as follows:

1. A view change triggers a new *async* flood fill run.
2. The flood fill results in n required potentially visible basemap, terrain, and 3D building tiles, which are not added to the scene yet.
3. Each tile abstraction — in parallel and depending on its nature — requests its related a) basemap tile, b) terrain tile, and/or c) 3D building tile based on its location on the map (i.e., its index) from respective servers.
4. Each download is started and handled by a *web worker*.



(a) Considered tile

(b) Western region

(c) South-east region

(d) North-east region

Figure 4.6: The considered 3D city model tile is outlined in (a). By investigating different locations on the tile (highlighted areas zoomed in in figure (b), (c) and (d)) far off the tile's center (which is the reprojected point, hence, destined to match the target coordinate system), it can be confirmed, that the deviation from the *Spherical Mercator* basemap, even at the tile's corners, is negligible — if even caused by the reprojection at all. Image sources: Prototype screenshots

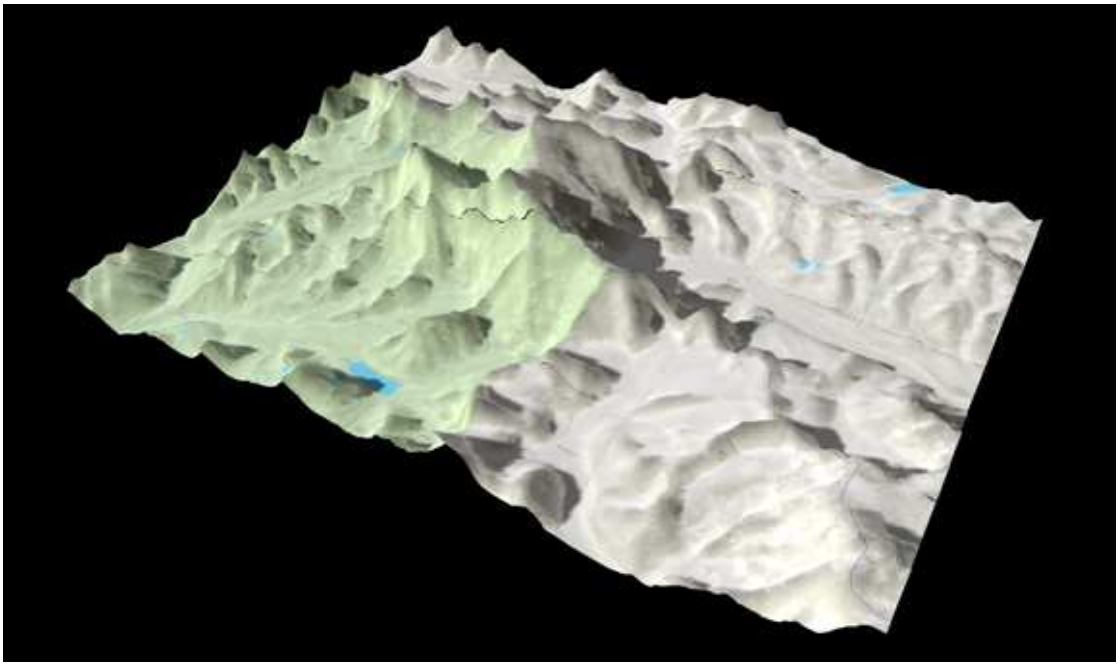


Figure 4.7: Mount Everest and its surrounding mountain range as rendered by the developed prototype, consisting of 7 x 5 basemap textured terrain tiles. Image source: Prototype screenshot

5. As soon as the data is downloaded, the *web worker* triggers further processing: A tessellated 2D quad undergoes a vertex-shader based displacement mapping, according to the retrieved terrain data. The basemap texture is applied, as well. In the case of 3D building tiles, the mesh is created and reprojected (see Section 4.1.3).

6. Every tile is added to the scene as soon as it is ready (asynchronously). All this happens in the background without blocking any user interaction.

4.1.5 Displacement mapping of terrain tiles

Through the aforementioned dynamic retrieval of terrain tiles and decoding of embedded elevation data (using the formulas described in Section 3.2.3), a *vertex-shader* program can displace vertices of tessellated 2D planes accordingly to generate 2.5D terrain. Meshes (as shown in Figure 4.7) are, therefore, created on-demand as a user navigates to arbitrary coordinates (in case of Mount Everest, these are 27° 59' 16" N, 86° 55' 29" E (wikipedia.org, 2020c)).

Shortcoming: Elevation retrieval after mesh displacement

Since vertex-shader based displacement is one of the last processes before the final mesh is rasterized and transferred to the screen buffer, there is no way of retrieving height

information afterwards, as the mesh is not accessible for such computations anymore. This fact becomes relevant if, e.g., a user wants to know the altitude of a highlighted location in the scene, or if other objects must be positioned based on the terrain's shape. If, however, the displaced mesh as such was accessible in the 3D scene, height for any desired point could be measured by casting a ray into the scene and calculating its distance to the point of the first intersection with the terrain.
This becomes highly relevant for placement of vegetation such as trees: Compared to the Vienna OGD 3D city model, where all building models intrinsically contain their actual vertical offsets, the tree cadastre does not. Thus, as every tree's position is only specified through its latitude and longitude, their correct placement requires elevation retrieval for each x/y coordinate, they are "planted" into; information, a vertex-shader displaced terrain cannot provide out of the box. Advantages such as performance, self-shadowing, and elegant implementation, however, outweigh this drawback. To support height calculations within the scene, and, therefore, correct tree placement, an elevation-texture-sampling approach is considered for the future.

Shortcoming: Seams and normal vector discontinuity

In the current implementation, each terrain tile is created without consideration of its four neighbors. Hence, there is no interpolation between differing height values across adjacent tile borders, causing visible *seams* as soon as the terrain is not perfectly horizontal (see Figure 4.8). The steeper the surface and the lower the viewing angle, the stronger the visible effect. Figure 4.9 illustrates and explains this issue further.

Just as height values are currently not interpolated across neighboring tiles, the same applies to face normals. Hence, other detrimental artifacts are normal vector and accompanying *shading* discontinuities: Similar to discussed seams, as the normal directions are not interpolated across adjacent edges, there is a recognizable gap (shown in Figure 4.10).

Both issues can be solved by considering neighboring terrain textures during the vertex-shader based displacement mapping and interpolating a) height values and b) normal vectors accordingly. In theory, such processing should remove both types of artifacts completely.

Suggested future work to compensate for shortcomings

- *Internal:*
  - Implement an alternative for a ray casting based elevation measurement, as this cannot be done with the presented vertex-shader based displacement mapping approach. Finding the relevant elevation tile and sampling its height values to retrieve elevation data for a) user-defined locations and b) terrain guided placement of objects (e.g., trees), might be a successful and performant approach.

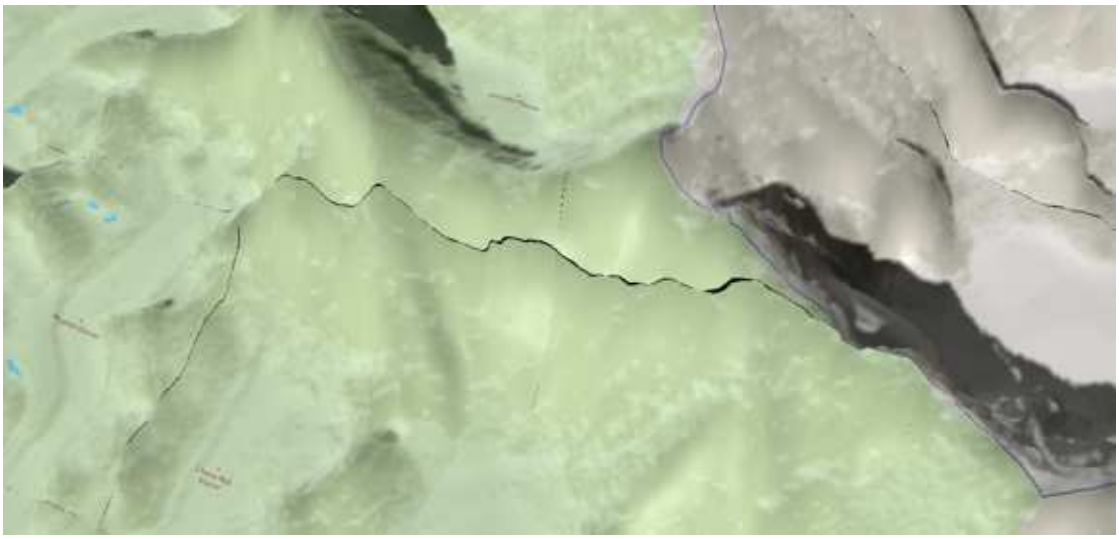
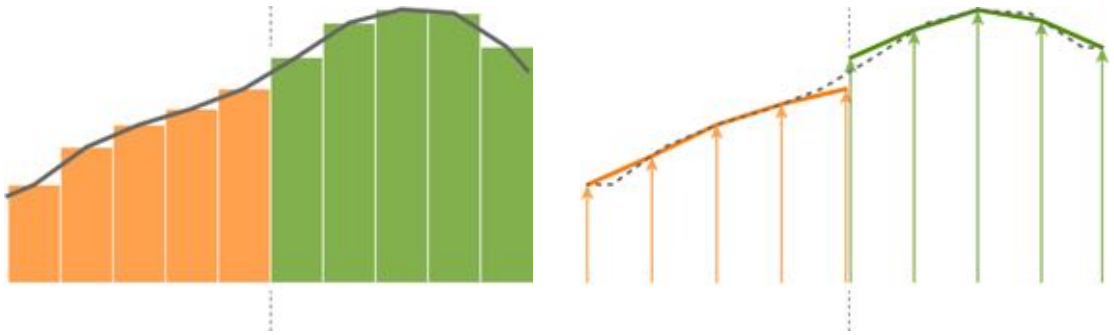


Figure 4.8: Seams between displacement mapped tiles are caused by the height difference between adjacent tiles' edges. The steeper the surface, the more prominent the seams, as the offset between adjacent edges increases. Since this figure is a magnification of Figure 4.7, the identical seam, among others, can also be seen there. Image source: Prototype screenshot



(a) Sampling (b) Reconstruction

Figure 4.9: In (a), the grey curve represents the sampled terrain, decoded, and averaged as pixel values among two different tiles (orange and green bars). The bars at the tile's border (dotted line) have a significant height gap. When terrain meshes are subsequently displaced in (b), by sampling the bilinearly interpolated elevation textures, there is a cut-off at both tile's borders, not considering the interpolated height between them. This results in a visible height gap identical to the one between the center bars in (a)

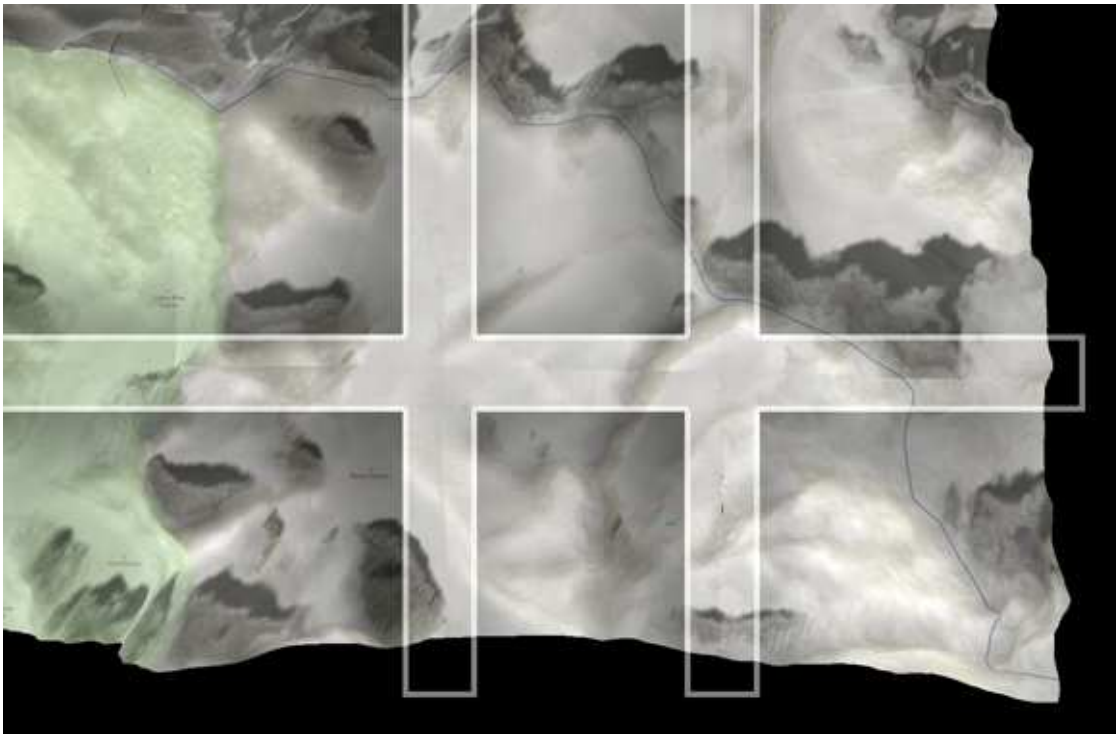


Figure 4.10: The highlighted area shows the visible and unnatural difference of shaded surfaces that are split between neighboring tiles. This is caused by the discontinuity of normal vectors among the tiles' adjacent edges and can be solved by interpolating them accordingly. Image source: Prototype screenshot

– Fix the visible seams between adjacent tiles by interpolating height values and normal vectors.

4.1.6 Trees

The metadata in Vienna's OGD tree cadastre is used to create tree approximations in 3D procedurally. Section 3.2.4 describes the selected metadata as well as the simple formulas used to model a tree's stem and crown. Following this laid out process, trees, as shown in Figures 4.11 and 4.12 can be modeled, which are naturally also capable of casting shadows.

Shortcomings

As described in the previous section, the current prototype does not provide simple retrieval of terrain elevation for a specific coordinate (i.e., a tree's latitude/longitude position). Since Vienna is not entirely flat but slightly hilly, trees are usually — besides some lucky coincidences — not placed at their actual vertical position. Instead, their

z-position is a hardcoded value affecting *all* trees equally. This leads to trees floating above the surface or other trees sinking into the ground, as shown in Figure 4.12.

The trees' shape is very roughly approximated: Only a stem (= low poly cylinder) and a crown (= low poly sphere) are procedurally created, based on the aforementioned `stammumfang`, `baumhoehe`, `kronendurchmesser` ($circumfence_{stem}$, $height_{tree}$, $diameter_{crown}$. In other words, all trees look basically the same and only vary in proportions and size. While details like species or color might be less relevant in a solar shadow map context, a tree's shape obviously does affect the shadow it casts.

Section 3.2.4 also mentions the unstructured JSON source of the tree cadastre and the current situation, that only a subset of available data, consisting of 1,999 trees, is currently considered in the developed prototype. By iterating through this subset, the contained trees are "planted" accordingly — thereby ignoring the visible map clipping whatsoever.

Suggested future work to compensate for shortcomings

- *Internal:*
  - Again, implement a method to retrieve elevation for a specific location in order to plant trees at their actual vertical position (also refer to the previous Section 4.1.5).
  - Consider a tree's species to approximate its natural shape better. E.g., a fir tree has a distinctively different shape than a chestnut tree. Such information — even if not perfectly replicating ground truth — can be used to closer approximate reality. Furthermore, some species keep their needles (or maybe even leaves) during winter, while others lose them; and with them occluding surface area. Respecting such seasonal cycles could further improve the simulation's realism.
  - Intelligent preprocessing of the 200,000 trees covering OGD cadastre. A promising strategy would be its division into file-based tiles, where the tiles' size and position are adapted according to Vienna's LOD2 building tiles. Following this approach, all potentially visible trees, are also loaded and created within the current scene.

4.2 Visualization

Various strategies and methods that were presented and discussed in this thesis, are combined to create an actual interactive 3D solar shadow map. The *3D tile flood fill* algorithm, introduced in Section 3.4.1, is used to not only load and display basemap-textured terrain tiles but also 3D building tiles, as they are added into the *Three.js Scene*.

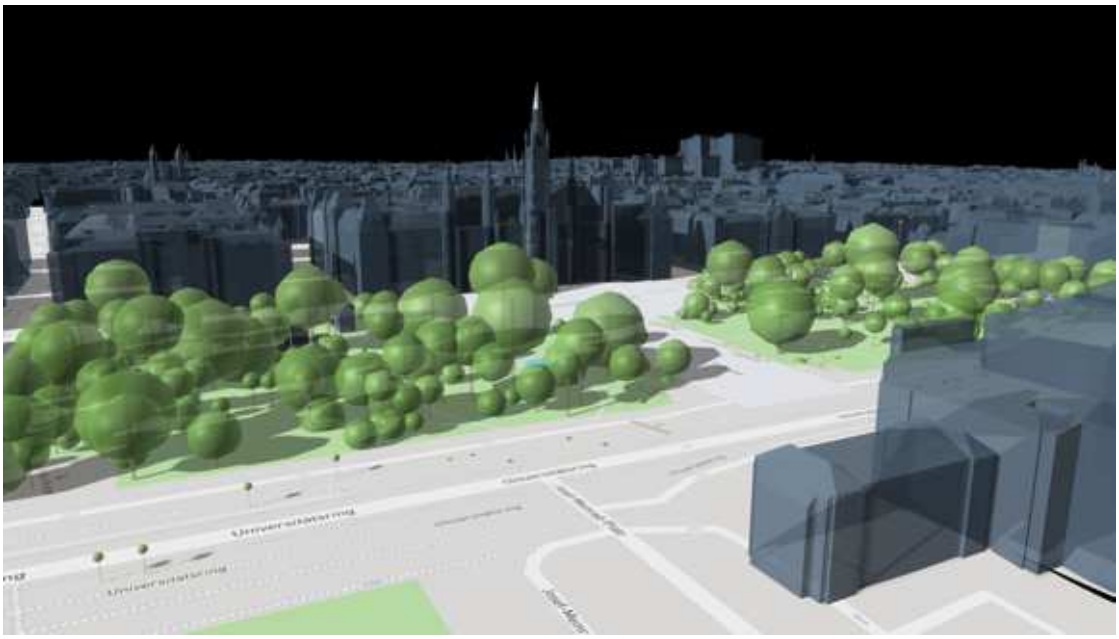


Figure 4.11: Procedurally created trees, varying in size and stem/crown diameters, created according to the metadata, Vienna's tree cadastre provides. Image source: Prototype screenshot

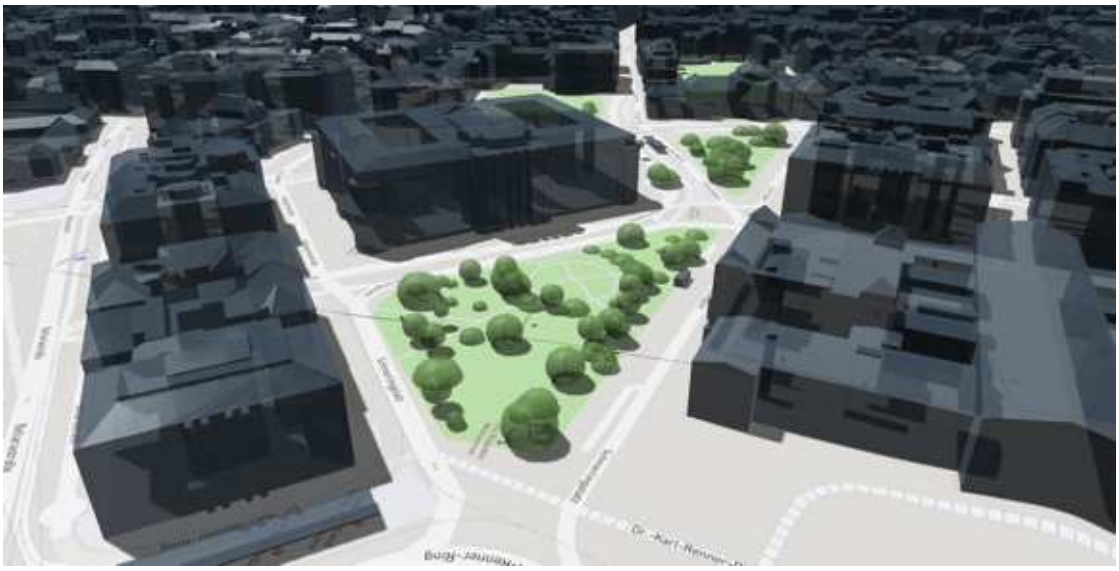


Figure 4.12: Since the current prototype does not support retrieval of terrain elevation for arbitrary locations on the surface, most trees are currently not placed at their correct vertical position and either sink into (as shown here) or float above the ground. Image source: Prototype screenshot

4.2.1 Camera

The *Three.js Scene* is observed by a perspective camera. Its distance to the scene's center on the map surface (= camera target) defines the level of detail used for the *basemap* and *terrain*, as already discussed in Section 3.3.2. This has several implications:

- The actual resolution of a tile — irrelevant its zoom level — is always the same.
- However, the displayed *content* of each tile, varies in regards to the depicted detail: Figure 4.13 shows the same location (Vienna) at different zoom levels (5, 11 and 16), and while the tiles' sizes are equivalent, the depicted content varies. At level 5, a single tile often displays several countries while at zoom level 16, a tile only spans over several housing blocks.
- Hence, the further the camera is from its target, the *larger* the 2D tessellated plane meshes need to be, as the camera uses perspective projection, which inherently reduces the size of objects in the distance.
- Zoom itself is a smooth process; i.e., the user can zoom the map to any desired size. There are, however, hard steps between tiles' level of detail as they are rasterized. As a consequence, prior to the switch to a closer zoom level, basemap and terrain detail are at the lowest possible quality before the higher detailed version is loaded and displayed. This is where vector tiles would excel, which, as discussed in Section 3.2.5, are hard to integrate into a 2.5D terrain, however.
- In the current implementation, all tiles per zoom level are stored in a multidimensional array and the very moment, the integer-zoom changes, a switch is triggered: The previously visible tiles are hidden from the *Scene*, and the tiles of the new zoom level are shown (and their retrieval is initiated, if they have not been cached yet). This is not the most elegant solution, as there are hard cuts and holes if there are no pre-cached tiles. A *quadtree* approach that replaces tiles with subtiles, only when the latter are actually loaded, is more elegant and considered for future work.

An equivalent solution would have been to maintain a constant distance between the camera and its target (i.e., not moving it), and scaling up the tiles instead. This, however, would have required more complex logic and also disallow usage of provided *Three.js* libraries, taking care of a camera's panning/zooming/tilting.

Unrasterized data

Since 3D buildings are not based on rasterized data, zoom-related versions — as it is the case with basemap and terrain tiles — are not required, because intrinsic vector data scales well among varying observer distance. It is, however, a waste of resources for far observer distances, since all the detail cannot be discerned in the first place. As of now, only the LOD2 version of Vienna's city model is used. The current prototype eventually

(a) Zoom level 5

(b) Zoom level 11

(c) Zoom level 16

Figure 4.13: The same location (Vienna) at different zoom levels. The screenshots were captured during initial loading to depict tiles which were not fully loaded yet, creating corresponding holes. A tile close to the center in (b) has its basemap already loaded while the terrain is still missing, disclosing its height delta as a visible seam. Image sources: Prototype screenshots

fades out and hides building and vegetation, as the camera increases its distance to the map, to save performance.

Scene origin and numerical instabilities

Since the prototype covers the whole Earth, the *Scene* covers a vast area. Its origin at 85.0511° N, 180° W corresponds to x/y-coordinates of 0/0 within the *Three.js Scene*. The opposite end at 85.0511° S, 180° E, however, is mapped to roughly 262000/262000, which are big numbers, leaving not enough bits for a fine-grained mantissa, able to reflect changes on a local scale. During development, this led to numerical instabilities that manifested in literally shaking trees as the camera rotated or jittering shadow maps. It was eventually compensated by, instead of moving the camera and directional light source, as the user pans and zooms the map, centering both within the *Scene*'s origin — and moving the map tiles instead. A fact that was already introduced in Section 3.4.2 when the Sun's light source positioning was discussed.

Suggested future work to compensate for shortcomings

- *Internal:*
  - Add support for different LOD of building models and, maybe, even trees: A future implementation could load and display the LOD1 model if a) the zoom level is far out, or b) there is no LOD2 model available; thus, the LOD1 serves as a fallback. The same applies to trees.
  - Right now, since the *3D tile flood fill* loads and creates tiles until they cover the view plane, camera tilting is limited (i.e., angles flatter than $\pi/3$ are not allowed). This is done in order to avoid large numbers of tiles or — in worst-case scenarios — an infinite loop within the algorithm. The camera's perspective projection further amplifies the issue. A solution to this would be a) LOD for the terrain, i.e., creating a low polygon version in the distance and eventually cutting it off or, much more straightforward, b) limiting the maximum number of flood filled tiles in the first place. For the latter, a suitable metric needs to be found.

4.2.2 The Sun

The Sun is considered again, this time for its representational form as a directional light in *Three.js*. In nature, and as presented in Section 2.1.5, Sun and Earth are both spheres, whereas the latter orbits the former. Due to the planar projection of Earth's surface, as it is implemented in the presented prototype, the orbiting motion is reprojected into an $x/y/z$ position of the Sun, orbiting the "flat Earth" in the *Scene* — according to time and current location on the map. Therefore, the reference location is always the 2D *center* on the map's surface as it is currently visualized, which is constantly translated into *WGS 84 latitude/longitude* coordinates. *Latitude*, *longitude*, and *time* are then, as previously

(a) (b)

Figure 4.14: Europe at noon on June 1st in figure (a) and during sunset (b), as it occurs at 20:32 CEST on that day. As Vienna is the currently focussed location, and due to planar projection of Earth's surface, all visible area is subject to the same Sun inclination, based on Vienna's latitude and longitude. In reality, however, the Sun would set two hours earlier in Bucharest (on the bottom right) than in London (upper left). Remark: The slight brightness gradient in (b) is caused by *Phong Shading* of the map and not related to Earth's (nonexistent) curvature, whatsoever. Image sources: Prototype screenshots

explained, used by *SunCalc* (SunCalc, 2016) to compute the Sun's *azimuth* and *altitude*. Using formulas presented in Section 3.2.6, the $x/y/z$ coordinates of the directional light source — representing the Sun — can be retrieved to position it accordingly.

This overall approach has the following impacts and side effects:

- Even though a few humans believe in the opposite, Earth is not flat. In the presented prototype, however, it is. Hence, if *for the currently centered location*, Sun has already risen, it has risen *for the whole Earth*, as the inclination of a directional light source is the same for every point on a planar surface. Figure 4.14 illustrates this fact.
- Since the Sun's position is dynamically updated as the map is panned along its longitude, the light situation changes accordingly. Also, the time of sunrise and sunset in the UI is updated (refer to Section 4.2.4).
- In order for the UI to stay consistent during longitudinal pans, the depicted time should be automatically adjusted based on the timezone the current location resides in. This feature is not implemented yet; thus, the displayed time always and

only refers to the location the map was initialized for. There are web-services, however, providing timezone information and APIs, which are considered for future integration.

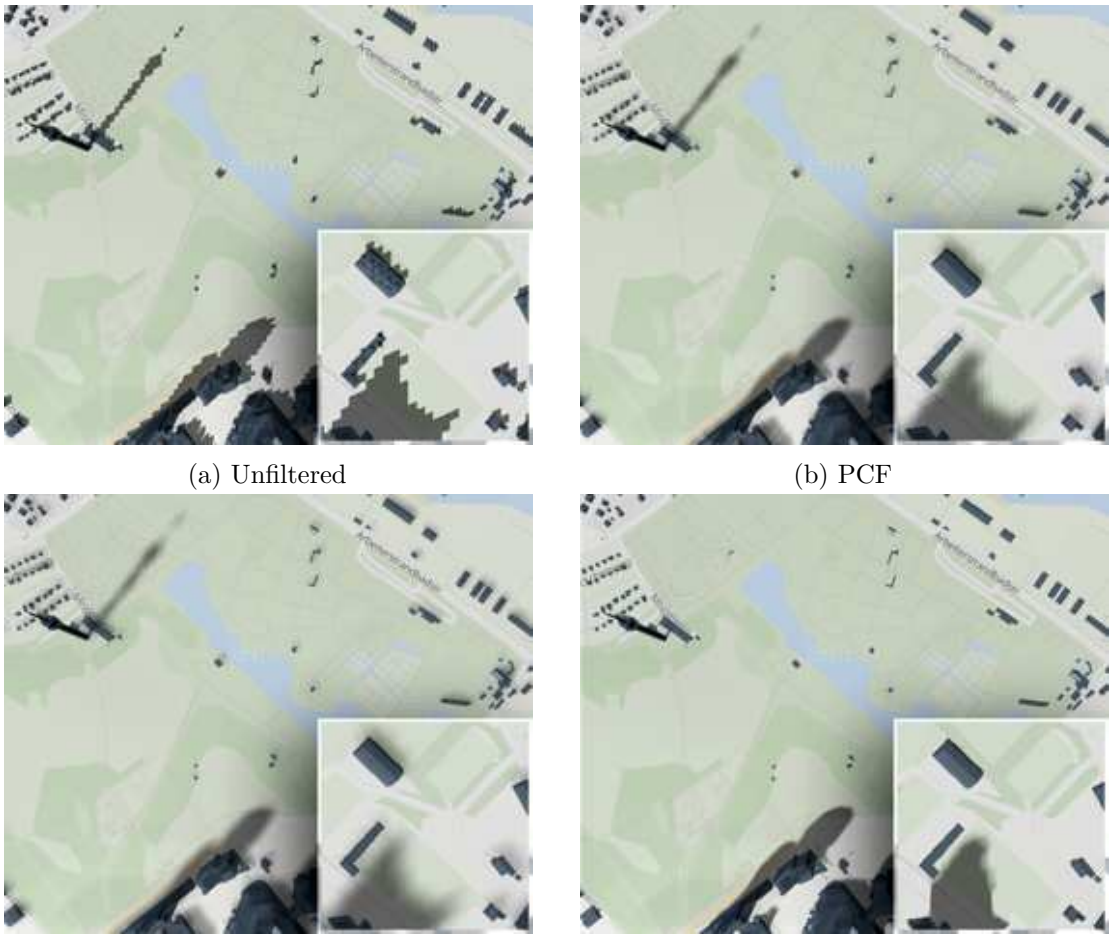
4.2.3 Shadow rendering

Chapter 2 provided insight into various shadow rendering methods and brought up arguments for *shadow mapping* as the appropriate way to visualize shadows within an interactive solar shadow map context (refer to Section 2.1.2). Among the various presented shadow mapping variants, *Three.js* supports *four* of them out of the box:

- Regular, unfiltered: `THREE.BasicShadowMap`
- *Percentage Closer Filtering (PCF):* `THREE.PCFShadowMap`
- *Percentage-Closer Soft Shadows (PCSS):* `THREE.PCFSoftShadowMap`
- *Variance Shadow Maps (VSM):* `THREE.VSMShadowMap`

In order to choose the most applicable method, they were compared within two different environments: One city context (Figure 4.15) and one terrain scenario (Figure 4.16) without buildings. It shows that the gradually changing slopes of the latter are challenging for unfiltered shadow mapping, whereas within a city context and flat ground, even the unfiltered variant produces solid results — as long as the depth map resolution is high enough. This is why the depth map size for the city scenario was reduced to 512 x 512 pixels to amplify the characteristics of each technique, as the visual quality among all variants at high resolutions is otherwise too good to show discernable differences.

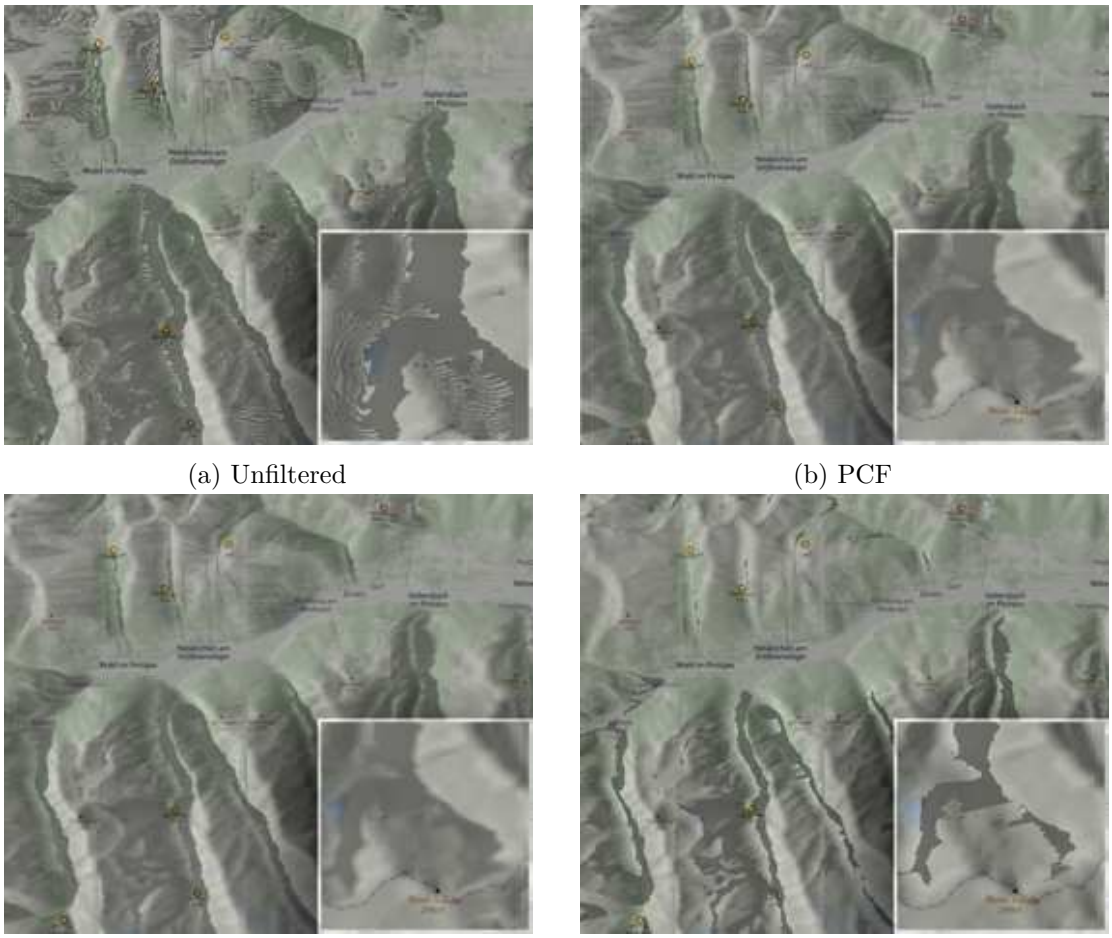
PCSS provides the best overall compromise of visual quality and speed on modern hardware. Given a high enough depth map resolution, shadows are sharp, and (temporal) aliasing is reduced to bearable levels — all while still maintaining interactive frame rates. In contrast, while unfiltered shadow mapping delivers proper quality on flat surfaces given a large enough depth map, as soon as terrain is added to the scene, heavy moiré occurs (aliasing). While this cannot be depicted here, temporal artifacts (e.g., occurring during camera pans or zooms, as well as shadow changes due to time adjustments) are significant. *PCF* effectively filters them out by taking 3 x 3 samples among a shadow border but still shows slight pixelation in the penumbra. *VSM* was only recently added to *Three.js* and also not rigorously tested with the prototype yet. Thus, it might profit from further parameter tweaking according to the prototype's scenery and light situation. However, out of the box, the method has a recognizable negative impact on the frame rate while not providing better shadows: In fact, the technique generates wrong and deformed shadows that depart significantly from the other approaches.



(a) Unfiltered (b) PCF

(c) PCSS (d) VSM

Figure 4.15: Shadow mapping variants in a city context. Depth map resolution = 512 x 512. The low resolution is easily recognizable at the unfiltered variant (a), which also produces very bad shimmering as shadows move. PCF (b) drastically improves the situation while still showing artifacts, which are completely smoothed out by PCSS (c). VSM (d) behaves unpredictably: The tower's shadow on the top left part of the image is gone (assumingly since its structure was too narrow for the low depth map resolution). Also, the zoomed-in shadow on the bottom right shows a sudden switch from a hard shadow border into a blurred one — and a wrong shape in general. Image sources: Prototype screenshots



(a) Unfiltered (b) PCF

(c) PCSS (d) VSM

Figure 4.16: Shadow mapping variants in a terrain scenario. Depth map resolution = 2048 x 2048. (a) shows strong aliasing artifacts due to “shadow acne”, which becomes even more obtrusive during light or camera changes (temporal artifacts). Similar to the city scenario, PCSS (c) offers the smoothed-out version of PCF (b), which already improves aliasing a lot compared to (a). VSM (d) creates almost vector-like shadow borders, but their shape differs from all others and is, in general, too narrow. What all shadow mapping variants have in common, are issues with self-shadowing on slopes, as sampling bias causes areas, that actually are in shadowed terrain, to wrongly brighten up. Image sources: Prototype screenshots

Shadow frustum optimization

For shadow mapping, the scene is rendered from the light's perspective in order to retrieve depth values, which are then stored in a depth map (refer to Section 2.1.2 for details). As this texture is subsequently sampled, shadow quality benefits from a higher resolution of this depth map. On the other hand, a higher resolution results in a slower rendering, thus, reduced FPS. Furthermore, the maximum possible texture size varies among different hardware.

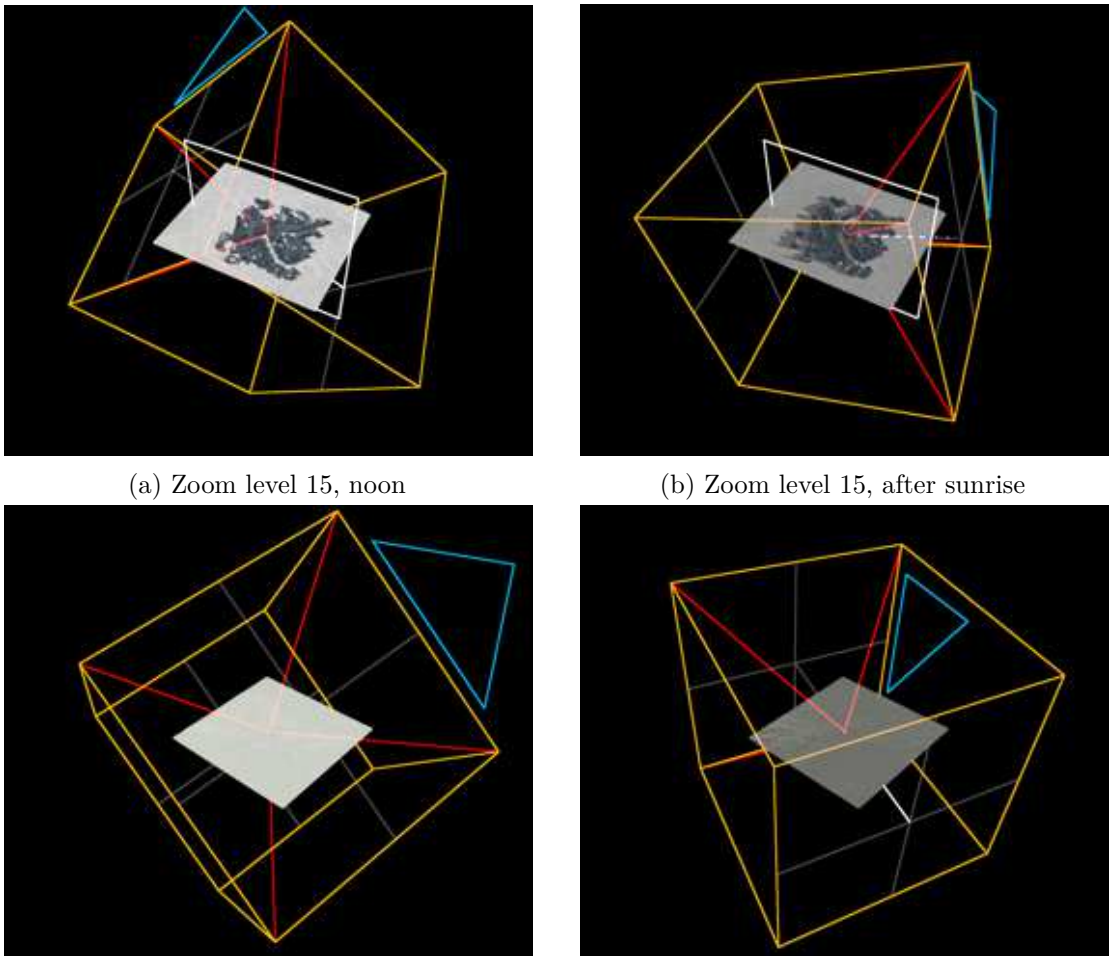
A strategy to optimally use a depth map's space is to *adapt* the size of the light's camera view frustum (shadow frustum): As described in Section 2.1.2, Martin and Tan (2004) do this sophisticatedly by approximating the observer's view frustum with a trapezoid, as seen from the light source. A naive approach with a similar aim, that is also implemented in the presented prototype, is described as follows:

The Sun is simulated by a directional light source. Its camera's view frustum, required for the depth map creation (i.e., every object that is inside this frustum can cause or retrieve shadow — while everything outside, cannot) is defined by a cube, with an edge length currently 3.4 times as long as the distance between the observer camera and the scene's center on the surface (= camera target). A smaller cube, while improving shadow quality, would obviously limit the range of where shadows can occur. This becomes an issue during strong tilting (i.e., low inclination angle) of the observer's camera, as the range of vision reaches further than the volume where shadows are actually rendered. During adjustments of time, as the Sun moves, the shadow frustum cube rotates accordingly, keeping the scene within its scope. Figure 4.17 illustrates this cube at different zoom levels and points in time.
In general, this approach utilizes one of shadow mapping's biggest strengths: Since it is agnostic to the scene's content, it does not matter whether shadows are computed for the whole city or just a single building: Performance does not suffer from higher scene complexity. Moreover, since the shadow frustum is adapted dynamically depending on observer distance, perceived shadow quality remains constant as well.

Time-integration

The algorithm behind a naive *Time-integration* approach was already presented in Section 3.4.2. In contrast to the advanced method by Miranda et al. (2019), the implemented shadow map based variant lacks the linear interpolation between discrete time steps, thus, discloses their respective shadows (as seen in Figure 3.13), which becomes especially recognizable for narrow structures. The quality of the result is dependent on three variables: 1) The number of samples (the higher, the better), 2) covered time range (the shorter, the better), and 3) narrowness of the occluder structure (the wider, the better).

Performance suffers from a higher sampling count, as more individual light sources need to be added to the scene, being illuminated by all of them simultaneously. Depending on available hardware, the impact of the implemented approach varies: On a 2016 MacBook



(a) Zoom level 15, noon

(b) Zoom level 15, after sunrise

(c) Zoom level 7, noon

(d) Zoom level 7, close to sunset

Figure 4.17: The yellow cube depicts the Sun's light source's view frustum. It is rotated according to the Sun's position as time changes, and scaled up and down as the zoom level (i.e., camera distance towards its target) is adjusted. The aim is to keep the visible scene, as seen from the observer, within this cube to maintain shadowing of relevant objects. All while avoiding waste of depth texture resolution on objects that are not visible in the first place. Image sources: Prototype screenshots

Pro with a dedicated GPU (Radeon Pro 455 with 2 GB RAM), six light sources can be visualized at almost interactive frame rates (17 ± 3 FPS) and nine lights at around 10 FPS — while an iPhone XS (released in 2018) supports five lights at most (also at interactive frame rates), and an iPhone 7 (2016) supports four and crashes with five.

The current web-based implementation of time-integrated shadow maps is far from reliable and needs optimization, as well as enhancements by considering new underlying strategies: One — not further investigated — idea is to *bake* temporal shadows in a sidelined process, and limit/disallow time-related user interaction consequently. Hence, after baking, a user would not be able to adjust the time without losing the previously generated temporal shadows. An approach could work and behave as follows:

- N time steps' shadow maps are merged ("baked") into a single texture map, which, as a result, becomes independent from all light sources.
- All but the $(n/2)$-nth light (i.e., the light averaging the considered time frame) are removed from the scene to accelerate rendering.
- Shadow mapping is disabled globally.
- The UI reflects the current state by disabling time adjustment and discouraging from panning outside the considered map area.
- The previously generated temporal shadow texture is projected onto the scene.

This is, however, part of future research and work.

Discussion

Research question *1./a)* asks, among others, whether a "visually appealing" visualization of solar shadows is possible within a web-context (refer to Section 1.2). It has been shown that especially *Three.js*' built-in implementation of *Percentage-Closer Soft Shadows (PCSS)* is visually capable. Nevertheless, a real evaluation of visual appeal would require a user study, which was not within the scope of this thesis. It is, however, considered for future work.

4.2.4 User interface

The user interface is designed with the primary aim to be familiar for users who are used to online map services, such as *Google Maps*, *Apple Maps*, *OSM*, and similar. This means that the dominant element of the UI is the map itself (see Figure 4.18 for screenshots of the web application running on a desktop browser, Figure 4.19 for a smartphone browser), which follows a "slippy map" paradigm (refer to Section 3.1.2) — a pannable, zoomable map. An addition to this is the ability to *tilt* the map to allow inspection of building facades or other angled or vertical structures.

Figure 4.18: The overall user interface of the implemented prototype. Image source: Prototype screenshot

Map interaction

Tilting is achieved by dragging with the right mouse button right in the scene — while the left one is reserved for panning. Zooming is done by scrolling the mouse wheel or applying equivalent gestures on laptop's trackpads, like pinching or swiping with two fingers.

On touch devices, a typical gesture among online maps is to use one-finger movement for panning and two-finger synchronous motion for tilting, while zooming is also achieved through pinching gestures. The library used in the current prototype to handle camera movement — `OrbitControls.js` (threejs.org/docs, 2020) — however, switches this pattern and links one-finger motion to tilting and two fingers to panning. This is undesired behavior, as panning is the far more relevant gesture. Since, unfortunately, the two gestures are interwoven within the provided code, they could not be separated and adapted accordingly yet. This task is, nevertheless, subject of future work.

Side panel

In addition to map interactions, a side panel is provided that takes over a) location querying and b) time adjustments (see Figure 4.20). The location search input field on

(a) Portrait mode

(b) Landscape mode, automatically hiding any browser UI

Figure 4.19: Screenshots of the prototype running on an iPhone XS. Image sources: Prototype screenshots

Figure 4.20: Side panel for location search and time-related functionality

the top of the panel is currently linked to *algolia Places* (algolia.com, 2020), a geocoder that also provides autocompletion for user inputs.

Below the location search, the date/time-related UI elements are aligned: On the top, the *current time*, followed by a slider that allows its adjustment, equipped with the following features:

- Its range reaches from 0:00 to 23:59, whereas the first 10 % of the slider's range cover the time from 0:00 to *sunrise*. The last 10 % of its range cover, analogously, the time from *sunset* to 23:59. This allows for a finer granularity for the relevant period of time where the Sun is actually above the horizon, actually affecting shadow rendering.
- The same time span is illustrated by a yellow background along with the slider's path.
- Time of sunrise and sunset is placed below the slider at the 10 % and 90 % mark, respectively.
- These labels are dynamically updated as the map is panned and times of sunrise and sunset change.

As previously mentioned, consideration of time zones is not implemented yet. This causes faulty behavior since the *current time*, as well as the times for sunrise and sunset, are relative to the time zone the current location resides in. Therefore, it is planned to respect time zones by dynamically updating all related UI elements accordingly, and also list the current time zone (and allow switching it as well).

Conclusively, there is a button at the side panel's bottom, showing the current *date*. By clicking or tapping on it, a calendar-like date chooser pops up, allowing to change the day of the year.

URL

Most of the current state the application resides in, is reflected within the browser's Uniform Resource Locator (URL). Vienna, at zoom level 15 and at 12:00 noon, for example, is represented by `https://[server_address]/?lng=16.37432&lat=48.21004&zoom=15&time=1588327200000&numSunlights=1`.

The *GET* parameters have the following meaning:

- `lng`: Longitude of the location as float number.
- `lat`: Latitude of the location as float number.
- `zoom`: Zoom level as integer (float representation is currently in development but not fully tested).
- `time`: Current time in milliseconds since January 1st, 1970, 00:00:00.000 GMT.
- `numSunlights`: This allows the activation of time-integration by increasing the number of light sources (= time steps). Right now, the considered time span is hardcoded at 60 minutes, as no other appropriate UI is implemented yet to handle temporal shadows.

4.2.5 Performance

In order to test the performance of the implemented prototype, and answer the research question of whether it is *"technologically possible to interactively visualize solar shadows caused by terrain, vegetation and buildings with sufficient precision in a web-based context"* (refer to Section 1.2), the application was tested on four different devices in three different main scenarios. The resulting FPS are then averaged to give an overall impression of the device's performance. As defined in Section 1.1.6, the aim of the presented implementation is to reach at least 24 FPS as a custom set benchmark for interactive visualization.

An additional test scenario covers the early implementation of time-integrated shadows. Since this feature is still in an experimental state, the test shall provide a first insight into the potential and is, therefore, separated from the overall results.

Test devices

The four tested devices include one laptop computer and three smartphones from two different manufacturers:

1. **Apple MacBook Pro, 15-inch:**
   - Released in 2016
   - 16 GB RAM, 2.7 GHz Intel Core i7 (4 cores)

- Radeon Pro 455 GPU with 2 GB VRAM
- Running Google Chrome Version 79.0.3945 (Official Build) (64-Bit) at a resolution of 1,920 x 1,080 pixels

2. **Apple iPhone XS** (wikipedia.org, A12):
   - Released in 2018
   - 4GB RAM, A12 Bionic CPU (2 + 4 cores)
   - Integrated GPU (4 cores)
   - Running Safari on iOS version 13.3.1 at a resolution of 2,436 x 1,125 pixels
3. **Samsung Galaxy Note 8** (wikipedia.org, Note 8):
   - Released in 2017
   - 6GB RAM, Exynos 8895 CPU (8 cores)
   - Exynos Mali-G71 MP20 GPU (20 cores)
   - Running Chrome version 79.0.3945 on Android version 9 at a resolution of 2,960 x 1,440 pixels
4. **Apple iPhone 7** (wikipedia.org, A10):
   - Released in 2016
   - 2GB RAM, A10 Fusion CPU (2 + 2 cores)
   - PowerVR GT7600 Plus GPU (6 cores)
   - Running Safari on iOS version 13.3 at a resolution of 1,334 x 750 pixels

The hardware of the *iPhone XS* is — excluding RAM and display — almost identical to the tablets *iPad Mini* (5th generation, 2019) and *iPad Air* (3rd generation, 2019) (wikipedia.org, iPad). Hence, besides their higher screen resolution due to larger display sizes and slightly reduced RAM (3 GB instead of 4 GB, ibid.), they should be able to achieve comparable performance.

Test scenarios

All devices were tested among the following four different (3 + 1) scenarios, which were exposed to 4 + 1 user interactions in order to cover a realistic space of practical use. This also includes the differentiation between a top-down perspective (i.e., normal to the surface) and a tilted variant at the maximum possible angle for the given zoom level. The latter drastically increases the number of loaded and displayed objects.

- **Scenarios:**

1. **Vienna at zoom level 15:** This scenario is characterized by a large amount of LOD2 3D city models on 2.5D terrain (Figure 4.21). While the terrain is mostly flat, it is still tessellated, thus increasing polygon count.
2. **Vienna at zoom level 18:** Very similar to the first scenario, however, the closer zoom reduces visibility range, including the number of LOD2 3D city models being visualized (Figure 4.22).
3. **Terrain at zoom level 12:** Main characteristic is the lack of 3D buildings, as this scenario is reduced to the rendering of the Austrian Alps (Figure 4.23).
4. **Temporal shadows with n=4:** Bonus test for the experimental feature of time-integrated shadows with four simultaneous light sources, as the tested *iPhone 7* is not able to handle a higher number. The rest of the setting is taken from scenario 2, including zoom level 18 (Figure 4.24).

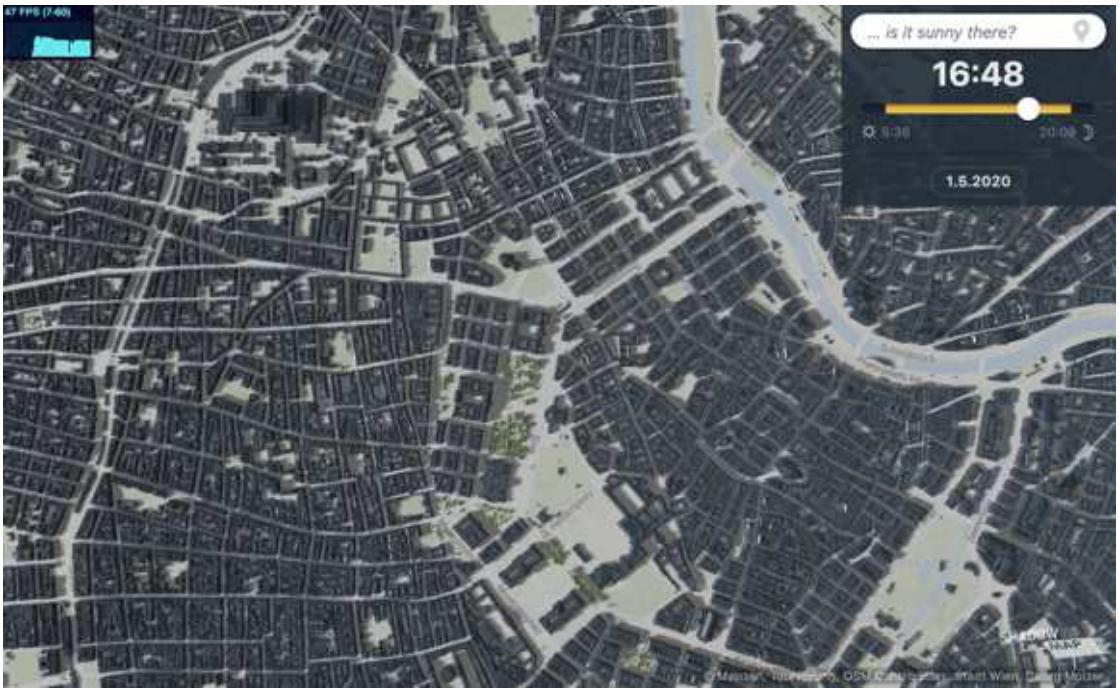
- **User interactions:**

1. **Idle (90°):** No actual interaction, just a still scene as seen from the top at 90° towards the surface. The frame rate is measured after an initialization and loading phase.
2. **Idle (tilted):** The *tilted* variant needs to handle a significantly wider visible area, including buildings, terrain tiles, and vegetation, if available. There is also no actual interaction, and the frame rate is measured after an initialization and loading phase as well.
3. **Rotation:** The scene is manually and randomly rotated while staying at the same location. Since the frame rate varies depending on the viewing angle, a mean value is considered.
4. **Pan (tilted):** At a tilted angle, the scene is panned manually. Thus, new objects come into view and might even need to be dynamically loaded during the process. The framerate, measured during the process, is averaged as well.
5. **Time Adj.:** This is to measure the performance impact of adjusting time within the current visible scene, thus, triggering updates of the shadow visualization. The performance impact is listed in percent, averaging the two idle states (90° and tilted).

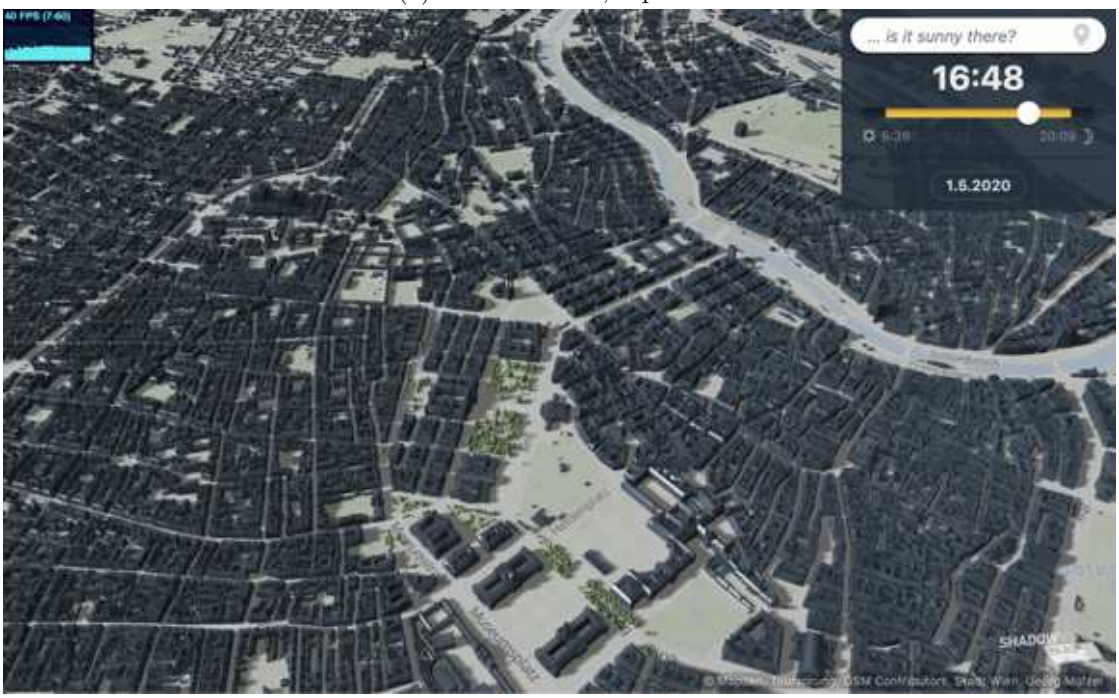
Results

1) MacBook Pro, 15-inch:

Table 4.1 lists the results for test device 1, the *15-inch MacBook Pro* laptop computer. What is easy to observe is the cost of LOD2 buildings. While terrain alone runs at full FPS almost throughout, Vienna at zoom level 18 (= fewer buildings visible than on level 15) suffers from a significant performance drop during interaction. This gets worse for zoom level 15 where many hundreds, if not thousands of 3D buildings are



(a) Zoom level 15, top view



(b) Zoom level 15, tilted view

Figure 4.21: Performance test scenario 1: Vienna at zoom level 15. Image sources: Prototype screenshots

(a) Zoom level 18, top view

(b) Zoom level 18, tilted view

Figure 4.22: Performance test scenario 2: Vienna at zoom level 18. Image sources: Prototype screenshots

(a) Zoom level 12, top view

(b) Zoom level 12, tilted view

Figure 4.23: Performance test scenario 3: Terrain at zoom level 12. Image sources: Prototype screenshots

Figure 4.24: Bonus scenario 4: Time-integrated shadows for four time steps at zoom level 18 in the city. Image source: Prototype screenshot

MacBook Pro (2016)	1920 x 1080					
Scenario	Idle (90°) *fps*	Idle (tilted) *fps*	Rotation *fps*	Pan (tilted) *fps*	Mean *fps*	*Time Adj.*
1) Vienna Zoom=15	48	38	42	23	37,8	19 %
2) Vienna Zoom=18	60	42	51	29	45,5	10 %
3) Terrain Zoom=12	60	60	60	50	57,5	5 %
Mean	56,0	46,7	51,0	34,0	46,9	11,3 %
4) Temporal Shadows n=4	37	23	30	15	26,25	10 %

Table 4.1: Performance test on device 1: MacBook Pro (2016)

rendered simultaneously. The tilted perspective ("Idle (tilted)") increases visible range, thus, bringing more buildings into the view and, therefore, further reducing FPS. This is amplified by the transparency of buildings. During panning, as this is done with a tilted camera as well, not only are there already many buildings in the view plane, but new ones are loaded and instanced at the same time, causing a further reduction of achieved FPS.

As the zoom level increases in the city context, rendering speed accelerates, since fewer buildings need to be considered. Terrain, which has no buildings at all, runs at the

highest possible frame rates, which only drop a little as the map is panned and new terrain is loaded and generated. All in all, for the relevant scenarios and interactions tested, FPS average at 46.9, which can be considered as *interactive*.

The impact of time adjustments (last column) is relatively minor, especially in terrain. In scenario 1, however, a 20 % FPS reduction can be measured, which, in fact, initially contradicts the assumption of shadow mapping being agnostic to the scene's content. The real reason for this drop — compared to scenarios with less geometry — needs further investigation. Regarding temporal shadows with four simultaneous light sources, performance is actually, ranging from 23 to 37 FPS in the non-panning states.

2) iPhone XS:

iPhone XS	2436 x 1125					
Scenario	Idle (90°) *fps*	Idle (tilted) *fps*	Rotation *fps*	Pan (tilted) *fps*	Mean *fps*	Time Adj.
1) Vienna Zoom=15	30	16	23	17	21,5	15 %
2) Vienna Zoom=18	60	32	45	31	42,0	19 %
3) Terrain Zoom=12	60	60	60	53	58,3	11 %
Mean	50,0	36,0	42,7	33,7	40,6	15,0 %
4) Temporal Shadows n=4	28	16	19	12	18,75	20 %

Table 4.2: Performance test on device 2: iPhone XS (2018)

The second device tested, the *iPhone XS* smartphone, shows a similar picture in regards to the cost of rendering large amounts of LOD2 buildings (see Table 4.2). Averaging at 40.6 FPS, the device seems slower than the previously tested *MacBook Pro* — however, it needs to render 2,436 x 1,125 pixels compared to the 1,920 x 1080 of the laptop. This means a 32 % higher resolution compared to the 16 % higher average FPS of the laptop (at a significantly lower resolution).

Leaving resolution aside, interestingly, the *iPhone XS* performs consequently lower in all tested interaction scenarios, but one: Panning. Here it is almost ex aequo with the laptop computer. A reasonable argument for this fact might be that the *iPhone XS* is actually faster at 3D mesh decompression and instantiation (which is relevant during panning) than the laptop, which compensates the otherwise slower rendering speed.

The overall 40.6 FPS are impressive for a handheld device and can be considered as *interactive* as well. Even temporal shadows work relatively fast at 19-28 FPS, leaving the panning state aside.

3) Samsung Galaxy Note 8:

The only Android device tested, a *Samsung Galaxy Note 8*, is one year older than the *iPhone XS*, and its performance running the implemented prototype is drastically worse. While it drives a higher resolution screen (2,960 x 1,440, thus 56 % more pixels than on

Galaxy Note 8	2960 x 1440					
Scenario	Idle (90°) *fps*	Idle (tilted) *fps*	Rotation *fps*	Pan (tilted) *fps*	Mean *fps*	Time Adj.
1) Vienna Zoom=15	12	7	7	5	7,8	57 %
2) Vienna Zoom=18	25	12	14	7	14,5	31 %
3) Terrain Zoom=12	28	16	19	9	18,0	10 %
Mean	21,7	11,7	13,3	7,0	13,4	32,7 %
4) Temporal Shadows n=4	1	1	1	1	1	50 %

Table 4.3: Performance test on device 3: Samsung Galaxy Note 8 (2017)

the *iPhone XS*), the browser's UI blocks 20 % of the screen in landscape mode, where iOS' Safari hides it completely. Nevertheless, the smartphone had problems with high numbers of buildings and also preferred the terrain-only scenario. Time, thus, shadow adjustments had further significant impact on the performance — even more in the city context where it peaked in a 57 % performance reduction (see Table 4.3).

Averaging all scenarios, the *Galaxy Note 8* reached only 13.4 FPS, which is below the defined benchmark of *interactivity*. Given the raw hardware specs (i.e., more CPU and GPU cores), this comes as a surprise. Furthermore, temporal shadows were practically unusable and never exceeded 1 FPS.

4) Apple iPhone 7:

IPhone 7	1334 x 750					
Scenario	Idle (90°) *fps*	Idle (tilted) *fps*	Rotation *fps*	Pan (tilted) *fps*	Mean *fps*	Time Adj.
1) Vienna Zoom=15	25	crash	crash	crash	-	39 %
2) Vienna Zoom=18	60	crash	crash	crash	-	24 %
3) Terrain Zoom=12	60	60	60	32	53,0	16 %
Mean	48,3	-	-	-	-	26,3 %
4) Temporal Shadows n=4	28	crash	crash	crash	-	56 %

Table 4.4: Performance test on device 4: iPhone 7 (2016)

The last device in the list, the *iPhone 7* is also the oldest. Released in 2016, it still delivers an impressive performance, which is not reflected in the results table (Table 4.4), as the prototype crashed reproducibly. This occurs as soon as the camera view is tilted and more building tiles need to be loaded, naturally occurring during the *Idle (tilted), Rotation, and Pan (tilted)* states within the *Vienna* scenario. During the time before the application crashes, however, frame rates are rather impressive and at least 2-3 x faster than the *Galaxy Note 8*. Since the performance was that good, further investigation on why the application crashed, are part of future work. A first hypothesis is that

the reduced amount of RAM — compared to the other contestants — causes troubles during the decompression and instantiation phase of the 3D LOD2 buildings. As this is happening under *web worker* parallelization, a strategy would be to limit the maximum amount of threads.

The terrain scenario runs exceptionally well: Being two years older than the *iPhone XS*, it reaches 91 % of its performance. However, time adjustments cause a significantly higher impact, averaging at 26.3 %. As mentioned before, regarding time-integrated shadows, the *iPhone 7* supports four light sources at maximum — actually reaching 28 FPS in the *Idle (*90°*)* state — while, again, crashing during user interaction.

Discussion

Depending on available hardware, *interactively visualization of a 3D solar shadow map within a web-context* is indeed achievable (refer to the *first* research question in Section 1.2). Two of the four tested devices exceed the defined threshold of 24 FPS significantly. One older smartphone, on the other hand, crashed reproducibly in the city/3D buildings scenarios. The Android device performed underwhelmingly — thus, the question arises, how more recent Android smartphones behave. During development, the prototype was randomly tested on Android hardware at various stages, and the performance was consequently notably inferior to Apple hardware. Whether this is caused by the underlying GPU, CPU, or browser needs further investigation and, even more, a standardized testing scheme. On that note, it is essential to mention that the described test scenarios heavily relied on manual user input and were not fully automized, which introduces a bias to the results. Their primary aim, however, was to provide an idea about the prototype's interactivity.

What all tested hardware has in common, is a notable performance drop as soon as 3D LOD2 buildings are added to the scene. This identifies a scope that deserves special attention: Right now, buildings are rendered semi-transparently, with the aim to partially see the basemap texture below them to improve navigational context (something that terrain does not need as it *is* the basemap — which also manifests in much higher FPS). This, however, means that culling of invisible objects, as they are occluded by structures in the front, is practically removed from potential scene optimizations since transparent buildings inherently cannot occlude anything behind them. The situation gets worse at tilted viewing angles as more and more buildings, aligned behind each other, appear in the scene. First tests with opaque buildings dramatically improve FPS, however, at the cost of basemap visibility. Therefore, adaptive adjustment of this feature, based on hardware capabilities, seems reasonable.

This also concerns other visual effects, such as the employed *Phong Shading* of the whole scene. It is used to give a subtle impression of Sun angle and inclination by visualizing reflections on roofs and surfaces. However, the effect is not crucial to a shadow visualization and more of a *nice to have*. The same is valid for full resolution on "high-DPI"

devices (which all of the tested hardware were): Shadows could be communicated with half the provided resolution; they, however, would not *look as pleasant*.

A final aspect in regards to 3D buildings is its current lack of LOD: By linking the buildings' LOD to the observer distance, polygon count, thus, complexity of the visible scene, could be drastically reduced. By implementing such an approach, test scenario 1 at zoom level 15 might even perform faster than scenario 2 at zoom level 18 — since scenario 1 only requires LOD1, while scenario 2 uses LOD2.

A very positive aspect of the implementation is the asynchronous loading and instantiation of geometry: The use of modern JavaScript features in these regards (*async/await* as well as *Web Workers*) practically eliminates UI blocking or stuttering, allowing smooth interaction during the tested scenarios.

Given the still immature phase the prototype still resides in, its performance is promising — both visually and in regards to its speed. There is still room for improvement, however, which will first be tackled by further investigating performance bottlenecks as well as crashes on older devices.

4.3 Qualitative evaluation

This section is about comparing scenes, as they are simulated by the implemented prototype with actual reality, represented by photographs. If the prototype is precise enough, visualized shadows should resemble their real counterparts. For this, only photographs where location and time are known should be considered, whereas the latter is usually contained within the EXIF metadata of an image file.

4.3.1 Facades

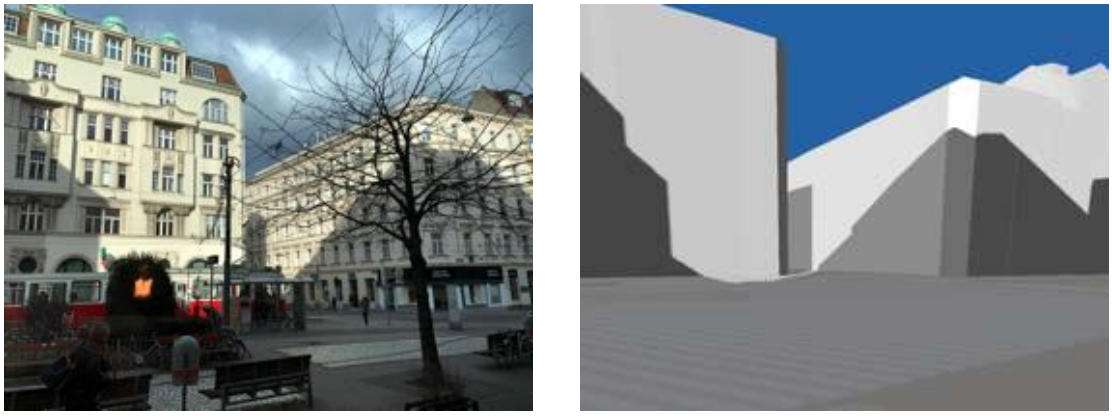
This section is about shadow play on building facades in Vienna. For shadows to be correctly visualized, the 3D city model needs to be accurate enough — both for the occluding structure as well as the shadow receiving one. Figures 4.25, 4.26, 4.27, and 4.28 show collected results.

4.3.2 Terrain

It was hard to find real terrain photography with a) the exact date and time available, b) discernable shadows, and — even more — c) the precise actual location. Aerial photography of unknown terrain makes correct positioning of the camera within the 3D scene difficult or even impossible. Figure 4.29 is, therefore, the only available comparison in this field, based on a picture the author took in 2014 out of a moving car.

4.3.3 Bird's eye view of Vienna

Two aerial photographs of Vienna are compared with their respective simulation. Both were taken with a much higher focal length; thus, the visualization was adapted accordingly



(a) Reality (b) Simulation (prototype screenshot)

Figure 4.25: Siebensternplatz, Vienna, on February 5th, 2020, at 14:01, as seen from Café 7stern. The simulated shadows resemble reality. Shadow details, like the multiple angles cast onto the left building, as well as the little bend at the front edge of the right building, are visible. What is also observable, is the lack of simulated gable structures on the left building, as they are not part of the LOD2 3D city model

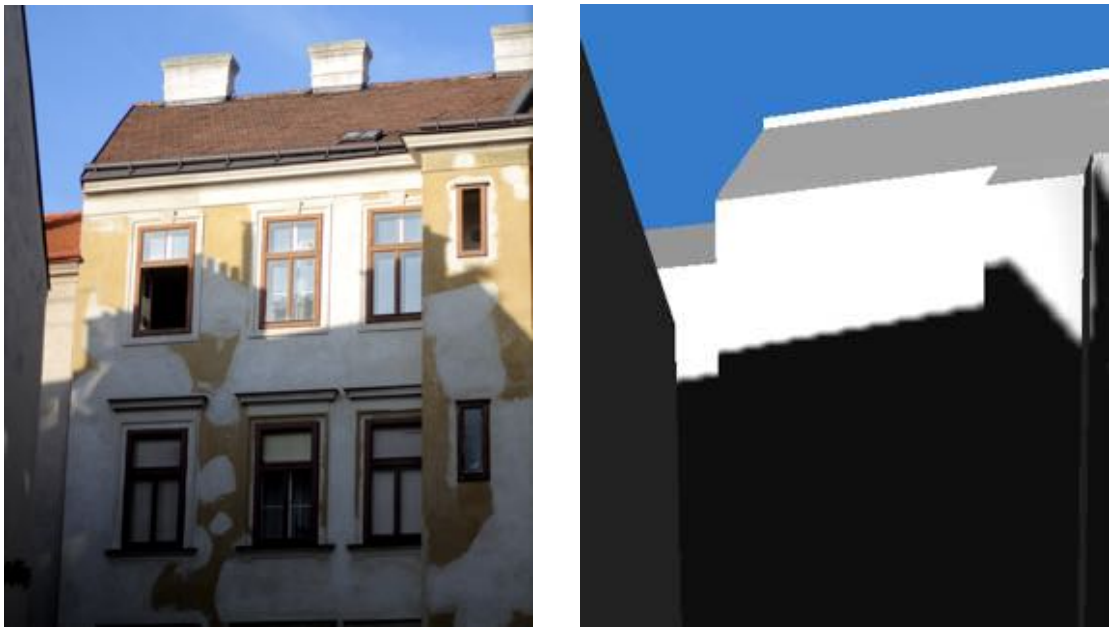
(Figure 4.30 and 4.31).

4.4 Discussion

Due to the visual evidence presented in the previous section, research question *1./b)*, regarding *"how close do simulated urban solar shadow scenarios resemble their real-life counterparts?"*, can be answered with: Close, at least.

There are limitations, nevertheless: As previously mentioned, Vienna's trees are not fully integrated yet. Furthermore, trees in private property are not provided by Vienna's OGD tree cadastre, and tree species, as well as their winter/summer appearance, are not considered yet. Also, their z-position is incorrect since the displacement mapped terrain cannot be ray cast; thus, another approach is needed, which is not implemented yet.

Leaving vegetation aside and focussing on terrain and buildings, the results are promising, and this includes the quality of available data: Even the LOD2 version of Vienna is already close to reality, only minor structures, such as chimneys and gables, are missing, which obviously make a difference in the visualization. However, due to their relatively low volume, those impacts are bearable and could, in the future, be compensated by a higher detailed LOD3 version. In regards to terrain, there are the aforementioned issues, where sometimes, depending on the data provider, the data is not what it is supposed to be. I.e., the elevation data by *Mapbox* is actually a surface model in some areas, providing wrong terrain data for a few tested cities, including Vienna. Outside of cities, however, data allows realistic scenery, as it is shown in Figure 4.29b. The terrain sample size



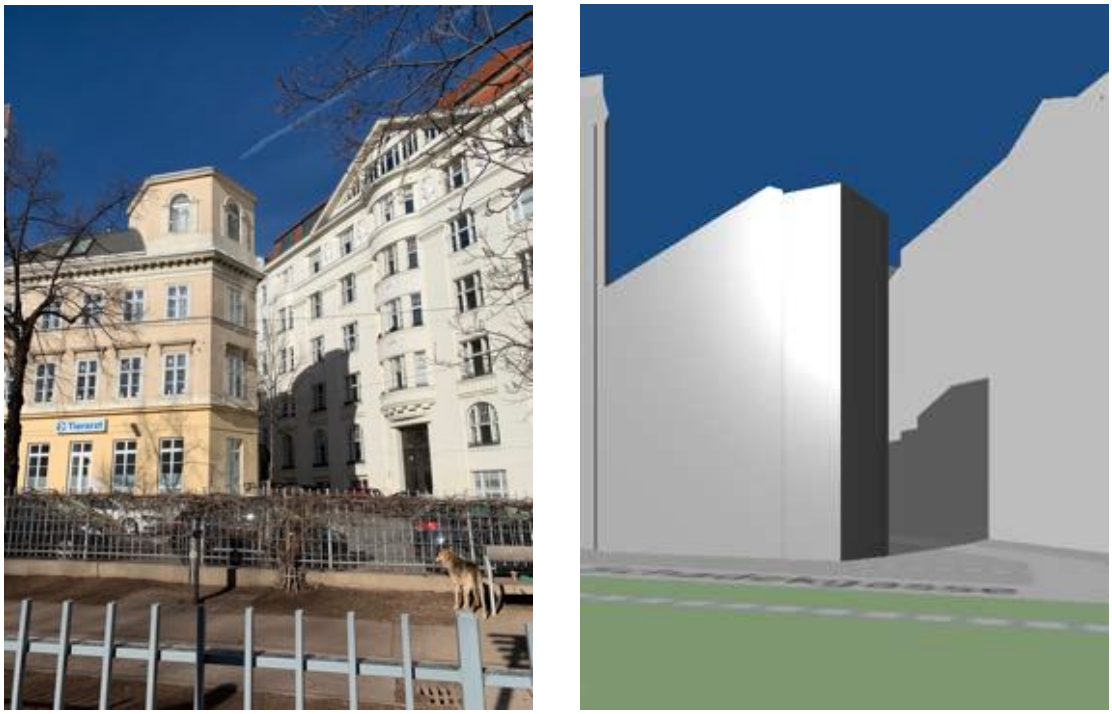
(a) Reality (b) Simulation (prototype screenshot)

Figure 4.26: Inner courtyard, 7th district, Vienna, on September 19th, 2019, at 8:10, as seen from an apartment. The rendering is done at a different field of view (= focal length) since the prototype's camera operation is not as flexible yet. Shadow projection is generally valid, nevertheless, lacking the chimneys as shadow casters. They are also not visible on the building's roof in the back. They should, however, be part of the LOD3 model, the Viennese OGD provides and might be integrated in the future

for the qualitative review is nevertheless admittedly small and needs further evaluation; however, aerial photography with precise metadata is challenging to collect. Conclusively, the generated results are better than initially expected — and even run at interactive frame rates, provided recent hardware.

In regards to data retrieval and processing, the current pipeline still requires significant manual interaction during its subsequent stages. An improved version would check data sources for updates and integrate them automatically into the application. Achieved compression rates of the LOD2 building model are impressive.

For the prototype's primary aim of *interactive visualization*, the decision to employ *shadow mapping* as the primary shadow visualization technique seems appropriate. From a quality and performance standpoint, the results presented in this section confirm this. There are nevertheless drawbacks like visible artifacts on tilted slopes that need further attention. Also, shadow mapping does not allow for precise analysis of insolation, which can be relevant for specific use cases. Another shortcoming concerns the lack of LOD among city models. This unnecessarily limits the potential performance and UX, as, for example, camera tilting is artificially constrained to reduce viewing range, thus,

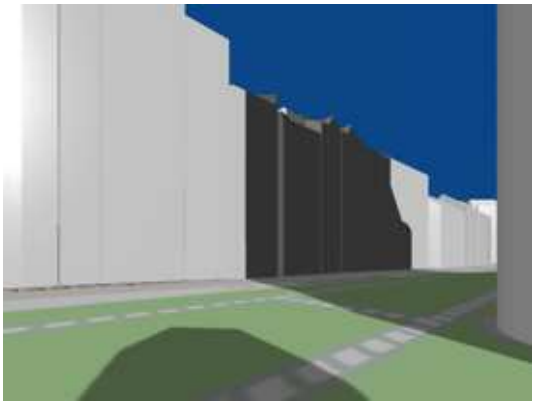


(a) Reality (b) Simulation (prototype screenshot)

Figure 4.27: Schadekgasse crossing Nelkengasse in Vienna's 7th district, on February 1st, 2020, at 13:28 as seen from Esterházypark on the other side of the street. Again, the main visible shadow renders well; the occluding roof structure on the yellow building to the left, however, is much more rounded in reality and weirdly abstracted in the LOD2 version. This causes wrong curvature of the cast shadow, as it can be seen in (b)

scene complexity. While the presented *3D tile flood fill* works robust and fast, it still lacks proper interpolation of height values and face-normals along tile borders, thus, causing seams. Furthermore, it does not allow ray casting based height retrieval; thus, an alternative method for situations like vertical positioning of trees is required. Solving these drawbacks is the subject of future work.

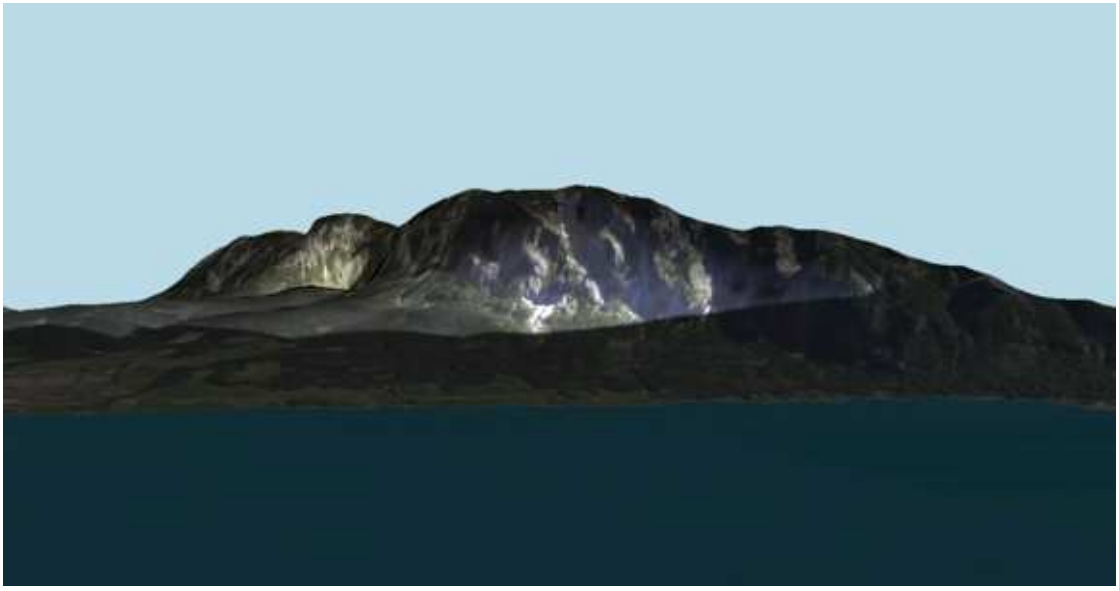
(a) Reality



(b) Simulation (prototype screenshot)

Figure 4.28: Same location and time as Figure 4.27, a little further to the right. Here, the simulated shadow (cast by the "Haus des Meeres" flak tower) starts too far off to the left, and it is unclear, why. Furthermore, on the right end, the simulated shadow suggests a tilted occluding structure. This can actually be confirmed as a recent constructional add-on: A new vertical structure at the "Haus des Meeres" now occludes a previously exposed diagonal roof. This change has not been reflected in the LOD2 model yet (refer to Section 4.1.1 about the model's update frequency). Another aspect concerns vegetation; as the shadow in the very front of the simulation is caused by a tree, and there are two problems with it: 1) The tree is stuck in the ground since vegetation cannot be placed at their correct vertical position yet (refer to Section 4.1.5 for more details). 2) Even though it is winter and reality refuses deciduous trees to maintain their leaves, the simulation does not reflect this fact yet

(a) Reality



(b) Simulation (prototype screenshot)

Figure 4.29: The "Brennerin" at lake Attersee in Upper Austria, Austria on January 6th, 2014, during sunset (15:56) as seen from the other side of the lake. For this simulation shot, the basemap was switched to Mapbox' provided satellite material as well as its elevation data, which provides great quality outside of Austrian cities. The silhouette of the mountain is well resembled, and the simulation of the shadow, covering the lower third of the mountain, is remarkably precise. However, the real version of that shadow has a blurred edge due to the Sun's nature of being an area light — something the simulation does not consider yet (see Section 2.1.5 for more details)

(a) Reality. Photo: "Vienna aerial MQ Ring 2aug14 - 1" by *Andrew Nash (andynash)*, flickr.com, CC BY-SA 2.0

(b) Simulation (prototype screenshot)

Figure 4.30: (a) shows Vienna's center districts, shot on August 2nd, 2014, at 18:17. Thus, the Sun is already closer to the horizon (sunset on that day was at 20:31), casting shadows in the whole complex scene. The simulation (b), due to its lack of building textures, and also far less trees (again, only 1,999 trees are currently loaded in the prototype), allows a clearer perception of the shadow situation. Comparing it to (a), it is recognizable that — besides obviously missing most trees — streets, facades, and squares, however, appear to match their real counterpart

(a) Reality. Photo: “MQ Luftaufnahme” by *MuseumsQuartier Wien* (museumsquartierwien), © Peter Korrak, flickr.com, CC BY-NC-ND 2.0 (cropped from original)

(b) Simulation (prototype screenshot)

Figure 4.31: For this picture, no EXIF date and time were available. Instead of proofing the correct Sun position for a given time in the simulated counterpart (which was proven by the previous comparisons already), the aim here is to compare shadow details and utilize the good picture quality of the photograph after adjusting the simulation time accordingly. The narrow shadows made a summer noon obvious, and simulated shadows are approximating reality

CHAPTER 5

Summary & Future Work

"One of the great challenges in life is knowing enough to think you're right but not enough to know you're wrong"

— Neil deGrasse Tyson

This thesis is about interactive solar shadow maps. Its title alone touches many disciplines, which are reflected throughout the presented work. The driving idea was to collect and transform available research and technology into something that reaches a broad, interdisciplinary audience, demonstrating that knowledge about the Sun, and the shadow situation it causes here on Earth, does not need to be reserved for experts, but can be made accessible to everyone with a web browser. Gained insight and knowledge through this access might be fruitful among many different application scenarios, and become more relevant in upcoming decades of climate change.

5.1 Summary

Motivation and aim of the work

Chapter 1 started with formulating the motivation behind the work, identifying potential use-cases, and discussing the shadow situation in dense urban areas, such as New York City, home of at least one street that does not see the Sun throughout the whole year. It was argued that, even though most — if not all — required data to visualize solar shadows is, in fact, available, an accessible tool is surprisingly lacking. A historical geoscientific perspective of the Sun's impact on various systems underlined the relevance of such a tool. The article on "the right to light" discussed how increased urbanization and accompanying taller building structures reduce human exposure to sunlight, which can even be linked to health issues. Therefore, municipal efforts and legislation were presented that try to counteract this trend, ensuring access to sunlight even in crowded cities. Use-cases spanning over various disciplines were identified that could benefit from a deeper understanding of solar shadows, adding predictability and plannability to

sunlight affected aspects. Among them were fields like photography, tourism, gastronomy, or real estate — but also questions of common private life, like: "Where should I plant my tomatoes?" or "Which parking spot is the shadiest for the next 2 hours?". Early attempts on solar shadow maps were presented, which, ironically, mostly had drinking in the Sun in their mind and did not survive as usable products.

After a first attempt to identify the aims and challenges of a potential solution, the problem statement and related research questions were formulated. It was postulated that: *"Due to the relevance of the Sun on (human) life on Earth, it can be argued that every human should be able to effortlessly and interactively explore and understand solar shadows in cities and around the world in an accessible, practical and visually appealing manner."* – thus, the rest of the work revolves around the problem space, spanned up by this formulation, with the eventual aim to implement a prototype that proves technological feasibility, as well as qualitative sufficiency. Collected related literature, methods, and learnings were to be documented and discussed.

Literature and related work

State of the art literature in Chapter 2 provided insight into historical approaches on solar shadow maps, which — in one case — went as far as to build a physical machine that used the real Sun on plastic relief maps to simulate shadow situations. Other approaches began to employ computers to integrate more sophisticated features into insolation calculations. Computer graphics related literature covered different approaches on shadow rendering methods, from their origins to later stage evolvements and improvements. It was reasoned, why for a global solar shadow map, *shadow mapping* is probably the most fitting approach to use. Since shadow rendering in computer graphics goes hand in hand with an underlying three-dimensional scene, data sources, their dimensionality and representation were evaluated. With increased observer distance to the currently visible three-dimensional scene, the number of visible objects rises; thus, increasing complexity. Applied *LOD* counteracts this by reducing model complexity according to longer viewing distance, therefore, maintaining the interactivity of the application.

As the Sun plays a critical role in this thesis, related astronomical research introduced facts about its size, distance to Earth, orbital planes, seasonal changes, and other influential factors. It was discussed how a legit computer graphical approximation could look like, and how the Sun's position in the sky for arbitrary dates and locations on Earth can be calculated and retrieved.

In regards to existing applications, the outstanding work on "Shadow accrual maps" was presented, which allows semi-interactive to interactive three-dimensional temporal shadow visualization and even analysis, spanning arbitrary time ranges. While the accompanying implementation undoubtedly demonstrates what can be achieved in the context of solar shadow maps, its hardware demands are beyond what current web-contexts can provide. Other presented work — an iPhone app targeted towards photographers ("PhotoPills") and Vienna's "Stadtplan3D" — offer distinct features on their own; nevertheless, none of

the existing approaches integrated *all* the required aspects, as they were formulated in the problem statement.

Methodology towards a prototype

Chapter 3 presented the methodology towards an actual prototype implementation. It was split between overall design considerations, data retrieval, and processing, as well as potential means of visualization. A juxtaposition of the pros and cons of 2D and 3D visualization concluded that the 3D variant provides greater freedom of use and potentially decreases required data storage and transmission. The design pattern of a 2D "slippy map" was, therefore, extended to work in a 3D space, including handling of basemap tiles and related coordinate system conversions, which are necessary for planar mapping of geodesic data. In regards to data, all the identified required variants (i.e., buildings, terrain, vegetation, basemap, and the Sun itself) were discussed separately, covering their actual sources and characteristics such as dimensionality, LOD, decoding, and data quality. After data related aspects were researched and formulated, the question arose on how to visualize and put it into use: Hence, existing map rendering engines were investigated and compared, as the initial strategy was to build upon them and implement extensions and modifications to support solar shadows. Since none of them was thoroughly convincing and lacking in at least one necessary aspect, the decision was made to go one level down and use a web-based 3D engine (*Three.js*) as the foundation for a more customized implementation.

Prototype implementation

The actual implementation of a solar shadow map prototype, built with *Three.js* as a foundation, was covered in Chapter 4. Previously presented theoretical methodology was transformed into practical actions and related source code. Arguments that led to implementation decisions were reflected, shortcomings were identified, and potential future improvements presented. Again, the topics of data and visualization were separated as a reference to the previous chapter. Scripts, which were written to retrieve *Open Data*, triggered a discussion on how available interfaces could be designed to improve the efficiency and flexibility of the retrieval process. Impressive achievements in regards to 3D model compression were presented, and procedural generation of trees, based on OGD metadata, was described.

For the scope of visualization, camera and Sun positioning were examined, whereas the former required handling of numerical instabilities due to the vast — Earth-spanning — coverage of the map. After various shadow rendering methods were evaluated in Chapter 2, they were eventually applied and compared within urban and rural contexts. The performance was — besides issues with tilted slopes — convincing among all of them; however, *Percentage-Closer Soft Shadows (PCSS)* delivered the most visually appealing results at an acceptable performance drop, compared to an unfiltered shadow mapping approach. To further improve shadow rendering, it was described how the shadow frustum's volume can be adapted according to the current observer position, in

order to optimize utilization of available depth texture resolution. Furthermore, a naive implementation of time-integrated shadows was presented.

In order to answer the research question, whether a web-based solar shadow map can operate interactively, the implemented prototype underwent a performance analysis, covering various usage scenarios and interactions. They were tested on four different devices (one laptop and three smartphones), which proved that — on more recent devices — interactivity is possible. Besides speed, also precision was reviewed through qualitative evaluation. It could be shown that the simulation, as it was tested among the three categories (i.e., shadows on facades, within terrain and from an aerial perspective), performed at high accuracy. Deviations from reality were minor and, if observable, caused by a lack of detail in the origin data.

Conclusively, the presented work gives an affirmative answer to the main research question formulated in Chapter 1, Section 1.2: *It is, in fact, technologically possible to interactively visualize solar shadows caused by terrain, vegetation, and buildings with sufficient precision in a web-based context. Enabling an accessible, visually appealing map user experience.*

5.2 Future work

This thesis can be seen as the prerequisite for further research and work in different ways. The implemented prototype, for example, represents a solid foundation to build upon; however, currently lacks an actual user study. Hence, it is not proven whether the design decisions made are also approved and understood by humans. Due to the web-based nature of the implementation, allowing distributed access for a large audience, however, execution of representative user studies is simplified. Furthermore, the prototype provides a toolkit to experiment on; variants of visualizations, UI, or allowed interactions can be implemented and deployed, therefore, allowing A/B testing and evaluation. A related idea is to create use-case centric adaptations of the application; e.g., a photographer might need a different UX than a real estate agent or the driver of a car, looking for a shady parking spot. Each of such use-cases could be considered a research area on its own. Web distributed user tests can guide the creation of a favorable design.

Visualization

Visualization, as of now, went the straightforward approach of depicting shadows as they would look like in reality; in essence: Unlit versus lit surfaces. This, however, is not necessarily the best possible strategy. Future work could research radical deviations, away from mere color variants, going as far as to maybe even omit visualization per se and — instead — provide speech based navigation to the next sunny spot, or an API to retrieve shadow related information. This also relates to UI: While, right now, it consists of the most basic interactions, it could be designed in a way that fosters educational aspects, enabling users to learn about the Sun and its impact on Earth — right during use. Moreover, in the current state, decisions for basemap styles and building colors

were made intuitively, while prior research on these attributes could potentially improve comprehensibility.

Improving shortcomings

Considering the implemented prototype, some drawbacks, as well as potential solutions and improvements, were already discussed in Chapter 4. This includes further automation of initial data retrieval, and actual evaluation of the current 3D building reprojection. Also, interpolation of adjacent terrain tile elevation values and normal vectors, and the actual possibility to compute height values in the *Three.js* scene, as the displacement mapping currently happens in the vertex shader, are currently lacking. Furthermore, vegetation data needs proper consideration, demanding structuring and actual coverage of Vienna within the developed prototype. Besides terrain tiles (including their basemap texture), which are inherently sourced from LOD based datasets, other 3D objects such as buildings and trees are currently reduced to a single LOD, ignoring the observer distance. By adding LOD variants, visual quality, as well as rendering speed, could be improved significantly. Furthermore, the presented *3D tile flood fill* would benefit from a proper consideration of the current camera tilt by, i.e., limiting the total of loaded tiles and/or introducing reduced LOD for tiles in the distance.

Performance optimization

Since the application crashed reproducibly on older hardware (2016's iPhone 7), investigation of the underlying reasons is required. While the device is equipped with significantly less RAM than the rest of the test field, the implemented algorithms should detect and adapt to limited resources by slowing down or reducing rendering quality; crashing is unacceptable. This also raises the question, whether *native* implementations of the presented prototype (i.e., running directly on the operating system instead of their browsers) would perform significantly better, and how the more direct GPU access could enable further features, such as the 3D textures required for "Shadow accrual maps". Furthermore, there is vast potential for performance optimizations: Many of them are LOD related, while others concern visual features such as applied shading, rendering resolution, anti-aliasing or post-processing effects. They should be applied adaptively, based on available hardware power.

Data expansion

In regards to data, it is planned to add further cities to reach more people within their actual environment and allow comparison of shadow situations among different urban contexts. As mentioned before, there are additional cities besides Vienna, offering their city model through open data initiatives, waiting to be integrated into a solar shadow map. It can be assumed that their number is going to increase in the future. Data quality is, nevertheless, an important aspect that needs to be considered, as the discussed issue with wrong elevation data in Chapter 3 demonstrated. Thus, simultaneous integration of

various sources and semi-automated selection of the best or most fitting variant for a given area identifies as another research area.

Further ideas

An additional aspect that was consciously left out of this thesis, even though it is one of the most significant sunlight occluding factors of all, is weather. Reasons are its complexity and chaotic structure that contradicts long-term predictability; another one is the already broad scope of the thesis that eventually required this limitation. Given the three-dimensional nature of the implemented prototype, any occluding structure could be, nevertheless, integrated with manageable effort — and this includes clouds. In theory, a 2D cloud layer, sourced by weather data and provided via "cloud tiles", could be added to the scene, analogously as it is already happening with terrain tiles. By using 3D textures, the time component of cloud motion could be reflected within each 3D tile, thus, allowing an animated cloud layer and respective shadows.

Mathematical models for planetary constellations do not only cover Sun/Earth relations: For the implemented prototype, it would only require a minor adaptation to visualize *Moon shadows*, for example, allowing to find locations where the full Moon can be observed. Also, Mars, which might become colonized by humankind within this century, is exposed to the Sun and home to the tallest mountain in the solar system: Olympus Mons, measuring over 22 km in height. Therefore, it stands out as a significant sunlight occluder, being of possible interest in regards to its shadow impact on surrounding areas. There is much to discover, research, and explore — on Earth and beyond. And there is no end in simulating reality. But one for this thesis.

List of Figures

List of Tables

List of Algorithms

Glossary

A-GPS Assisted GPS. Speeds up positioning by augmenting GPS satellite data with cell tower data. The technology, furthermore, enables positioning in areas where there is no direct visual contact to satellites (e.g., tunnels). 18

COLLADA Open standard for an XML-based interchange file format for 3D applications. 8

DEM Digital elevation model. 3D model of a planet's surface or parts of it. Generic term for DTM and DSM. 25

DSM Digital surface model. 3D model of a planet's surface including structures like vegetation or buildings. 7, 25, 50, 68, 69, 72, 73

DTM Digital terrain model. 3D model of a planet's ground surface specifically excluding any artificial structures but also natural vegetation or similar.. 25, 56, 72, 73

insolation The solar power that irradiates on surfaces like terrain, building structures or other objects. 6, 24

LiDAR Light detection and ranging. Method to optically measure distance. Comparable to radar, but uses laser instead of radio beams. 7

Ly Langley (Ly) is the unit for heat transmission: For example, describing the rate of solar insolation received by Earth's surface. 1 Ly = 1 thermochemical calorie per cm^2. 25, 31

megacity Cities with, depending on the source, more than 5-10 mio. citizens. 10

OBJ Wavefront OBJ. Open file format for storing 3D geometry. 92–94

penumbra Combination of the Latin "paene" meaning "almost, nearly" and *umbra* ("shadow"). Compared to the *umbra*, this is the region in which not all of the (area) light source needs to be occluded. It resembles a gradient from unlit to fully lit surface. 33, 41

rasterization Process of turning vector data into raster image or bitmap consisting of pixels. 36

ray tracing A computer graphics rendering approach capable of producing highly realistic images by shooting rays for every pixel of the image into the scene and following their path as they bounce off surfaces (also recursively). Color values of hit surfaces are accumulated and eventually define the color of the pixel. Algorithm adaptations vary in regards to features, visual quality and required computation. 33, 53

responsive design Web programming and design paradigm that allows a single website or application to adapt to different display sizes and aspect ratios. 17

self-shadowing A term used in computer graphics (among other disciplines) describing a concave object casting shadows on itself if lit from suitable directions. 15, 16, 24

stencil buffer Two-dimensional (usually) 8-bit data array generally used to specify the region to be rendered, hence, the analogy to a stencil. Can also be used more creatively within the rendering pipeline, e.g., for rendering of shadow volumes. 36

umbra Latin for "shadow". The total shadow, which is, in contrast to the *penumbra*, completely obstructed from the light source by the occluder(s). 33

WebGL JavaScript graphics API that, foremost, allows hardware accelerated 3D rendering in web browsers. 54, 79, 82

z-buffer Also known as depth buffer. Two-dimensional array which stores depth information (z-values, i.e., distance from the observer) of rendered pixels to help solving visibility problems within a 3D scene. Granularity of the buffer varies between 16 and 32 bit on modern hardware. 38, 41

Acronyms

API Application Programming Interface. 66, 76, 82, 92, 140

BIM Building Information Modeling. 49

CityGML City Geography Markup Language. 7, 8, 46, 68–70, 90, 92, 94

DXF Drawing Exchange Format. 66, 68–70, 90, 92, 94, 95, 97

EPSG European Petroleum Survey Group. 63, 95

EXIF Exchangeable Image File Format. 89, 128, 135

FPS Frames per second. 15, 59, 82, 112, 114, 118, 120, 124–127

GIS Geographic Information System. 7, 80, 95

GPS Global Positioning System. 63

GPU Graphics processing unit. 18, 36, 42, 52, 54, 82, 141

LOD Level of Detail. 23, 43, 46, 61, 62, 87, 107, 128, 130, 138, 139, 141

ODbL Open Database License. 42

OGD Open Government Data. 66, 67, 73, 89, 90, 92, 100, 102, 103, 129, 130, 139

OSM OpenStreetMap. 2–4, 43, 60, 62, 66, 67, 72, 80, 84, 96, 114, 143, 144

SRS Spatial Reference Systems. 63

UI User interface. 21, 56–60, 81, 108, 114, 116–118, 140

URL Uniform Resource Locator. 118

UX User experience. 50, 58–60, 62, 78, 79, 81, 82, 92, 130, 140

WGS World Geodetic System. 63

Bibliography

aa.quae.nl (2018). Astronomy Answers, Position of the Sun. https://www.aa.quae.nl/en/reken/zonpositie.html. Accessed: 2020-01-24.

ABV Zürich (2019). Allgemeine Bauverordnung, Änderung der Schattenwurfregelung – Erläuterungsbericht. https://www.vzgv.ch/sites/vzgv.ch/files/vernehmlassung/erlaeuterungsbericht_zur_vernehmlassung.pdf. Accessed: 2020-01-09.

algolia.com (2020). OrbitControls. https://community.algolia.com/places/documentation.html. Accessed: 2020-02-3.

Allen, C. and Cox, A. (2001). *Allen's Astrophysical Quantities*. Allen's Astrophysical Quantities. Springer New York.

Appel, A. (1968). Some Techniques for Shading Machine Renderings of Solids. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, AFIPS '68 (Spring), page 37–45, New York, NY, USA. Association for Computing Machinery.

arcgis.com (2020). ArcGIS Online. https://www.arcgis.com. Accessed: 2020-01-28.

Autodesk Revit (2020). Autodesk Revit, Perform solar analysis during building design. https://knowledge.autodesk.com/support/revit-products/getting-started/caas/simplecontent/content/building-E2-80-93architecture-E2-80-94handling-solar-analysis-the-design-phase.html. Accessed: 2020-01-18.

Beauchemin, K. M. and Hays, P. (1996). Sunny hospital rooms expedite recovery from severe and refractory depressions. *Journal of affective disorders*, 40(1-2):49–51.

Biljecki, F. and Arroyo Ohori, K. (2015). Automatic Semantic-preserving Conversion Between OBJ and CityGML. In *Eurographics Workshop on Urban Data Modelling and Visualisation 2015*, pages 25–30, Delft, Netherlands.

Biljecki, F., Ledoux, H., and Stoter, J. (2017). Does a Finer Level of Detail of a 3D City Model Bring an Improvement for Estimating Shadows? pages 31–47. ISBN: 0000000213.

Biljecki, F., Stoter, J., Ledoux, H., Zlatanova, S., and Çöltekin, A. (2015). Applications of 3D City Models: State of the Art Review. 4(4):2842–2889.

Biskaborn, B. K., Smith, S. L., Noetzli, J., Matthes, H., Vieira, G., Streletskiy, D. A., Schoeneich, P., Romanovsky, V. E., Lewkowicz, A. G., Abramov, A., Allard, M., Boike, J., Cable, W. L., Christiansen, H. H., Delaloye, R., Diekmann, B., Drozdov, D., Etzelmüller, B., Grosse, G., Guglielmin, M., Ingeman-Nielsen, T., Isaksen, K., Ishikawa, M., Johansson, M., Johannsson, H., Joo, A., Kaverin, D., Kholodov, A., Konstantinov, P., Kröger, T., Lambiel, C., Lanckman, J.-P., Luo, D., Malkova, G., Meiklejohn, I., Moskalenko, N., Oliva, M., Phillips, M., Ramos, M., Sannel, A. B. K., Sergeev, D., Seybold, C., Skryabin, P., Vasiliev, A., Wu, Q., Yoshikawa, K., Zheleznyak, M., and Lantuit, H. (2019). Permafrost is warming at a global scale. *Nature Communications*, 10(1):264.

BKM Vienna LOD1 (2020). Baukörpermodell (LOD1) - Daten beziehen. https://www.wien.gv.at/stadtentwicklung/stadtvermessung/geodaten/bkm/. Accessed: 2020-01-12.

Boubekri, M. (2004). An Argument for Daylighting Legislation Because of Health. 7(2):51–56.

bp.com (2018). Solar Generation - TWH. http://www.bp.com/statisticalreview. Accessed: 2020-01-01.

Bremer, M., Mayr, A., Wichmann, V., Schmidtner, K., and Rutzinger, M. (2016). A new multi-scale 3D-GIS-approach for the assessment and dissemination of solar income of digital city models. 57:144–154.

buildwith.com (2019). Maps Usage Distribution in the Top 1 Million Sites. https://trends.builtwith.com/mapping/maps. Accessed: 2019-12-16.

caniuse.com (2020). Can I use: WebGL 2.0? https://caniuse.com/#feat=webgl2. Accessed: 2020-01-18.

Catita, C., Redweik, P., Pereira, J., and Brito, M. (2014). Extending solar potential analysis in buildings to vertical facades. 66:1–12. Publisher: Pergamon.

cesiumjs.org (2020). CesiumJS. https://cesiumjs.org. Accessed: 2020-01-28.

citygmlwiki.org (2019). Open Data Initiatives. http://citygmlwiki.org/index.php/Open_Data_Initiatives. Accessed: 2020-01-28.

Crow, F. C. (1977). Shadow algorithms for computer graphics. 11(2):242–248. ISBN: 089791029X.

data.gv.at (2019a). Katalog Baumkataster bzw. Bäume Standorte Wien. https://www.data.gv.at/katalog/dataset/stadt-wien_baumkatasterderstadtwien. Accessed: 2020-01-22.

data.gv.at (2019b). Katalog Generalisiertes Dachmodell (LOD2) Dreiecksvermaschung Wien. https://www.data.gv.at/katalog/dataset/generalisiertes-dachmodell-lod2-dreiecksvermaschung-wien. Accessed: 2020-01-29.

data.worldbank.org (2019). Rural population in percent of total population. https://data.worldbank.org/indicator/SP.RUR.TOTL.ZS?end=2016&start=1960&view=chart. Accessed: 2019-12-09.

Dimitrov, R. (2007). Cascaded shadow maps. *Developer Documentation, NVIDIA Corp.*

docs.mapbox.com (2020a). Access elevation data. https://docs.mapbox.com/help/troubleshooting/access-elevation-data/. Accessed: 2020-01-21.

docs.mapbox.com (2020b). Mapbox GL JS. https://docs.mapbox.com/mapbox-gl-js/overview/. Accessed: 2020-01-28.

docs.mapbox.com (2020c). Mapbox GL JS Examples: Add a 3D model. https://docs.mapbox.com/mapbox-gl-js/example/add-3d-model/. Accessed: 2020-01-26.

docs.mapbox.com (2020d). Maps service, Retrieve raster tiles. https://docs.mapbox.com/api/maps/#retrieve-raster-tiles. Accessed: 2020-01-22.

docs.unity3d.com (2019). WebGL Player settings. https://docs.unity3d.com/2019.2/Documentation/Manual/class-PlayerSettingsWebGL.html. Accessed: 2020-01-26.

Donnelly, W. and Lauritzen, A. (2006). Variance shadow maps. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games - SI3D '06*, page 161. ACM Press.

Duffield, C. (1975). Solar Shadow Maps. Master's thesis, The University of Arizona.

Emilio, M., Kuhn, J. R., Bush, R. I., and Scholl, I. F. (2012). Measuring the solar radius from space during the 2003 and 2006 mercury transits. (2000).

Encyclopædia Britannica (2018). Ancient lights. https://www.britannica.com/topic/ancient-lights. Accessed: 2019-12-10.

epsg.io (2020). epsg.io Coordinate Systems Worldwide. https://epsg.io. Accessed: 2020-01-30.

Fernando, R. (2005). Percentage-closer soft shadows. In *SIGGRAPH Sketches*, page 35.

Foley, J., Van, F., Van Dam, A., Feiner, S., Hughes, J., Angel, E., and Hughes, J. (1996). *Computer Graphics: Principles and Practice*. Addison-Wesley systems programming series. Addison-Wesley.

Freitas, S., Catita, C., Redweik, P., and Brito, M. (2014). Modelling solar potential in the urban environment: State-of-the-art review. 41:915–931. Publisher: Pergamon.

Garnett, A. (1935). Insolation, Topography, and Settlement in the Alps. *Geographical Review*, 25(4):601–617.

Garnier, B. J. and Ohmura, A. (1969). Estimating the topographic variations of short-wave radiation income: the example of Barbados. Technical report, Defense Technical Information Center, Fort Belvoir, VA.

gist.github.com/TimvanScherpenzeel (2020). Support table - WebGL and WebGL2 support. https://gist.github.com/TimvanScherpenzeel/f8efeeb1dbed38a5c5dc0c29768a0413. Accessed: 2020-01-18.

github.com (2016). Fixed bug on calculation of astronomical refraction. https://github.com/mourner/suncalc/pull/64. Accessed: 2020-01-24.

github.com (2019). Spherical Mercator math in Javascript. https://github.com/mapbox/sphericalmercator. Accessed: 2020-01-25.

github.com, Corto (2020). Draco 3D Data Compression. https://github.com/cnr-isti-vclab/corto. Accessed: 2020-01-30.

github.com, Draco (2019). Draco 3D Data Compression. https://github.com/google/draco. Accessed: 2020-01-29.

github.com, glTF Pipeline (2020). glTF Pipeline. https://github.com/AnalyticalGraphicsInc/gltf-pipeline. Accessed: 2020-01-30.

github.com/mapbox (2015). Support 3D terrain meshes. https://github.com/mapbox/mapbox-gl-js/issues/1489. Accessed: 2020-01-26.

google.com/get/sunroof (2020). Google Project Sunroof. https://www.google.com/get/sunroof. Accessed: 2020-01-19.

Heidmann, T. (1989). Real shadows real time. *Iris Universe*, 18:28–31.

help.openstreetmap.org (2011). Can you export elevation data from OSM? https://help.openstreetmap.org/questions/3069/elevation-maps. Accessed: 2020-01-22.

Hendrick, R. L., Filgate, B. D., and Adams, W. M. (1971). Application of Environmental Analysis to Watershed Snowmelt. *Journal of Applied Meteorology*, 10(3):418–429.

kickstarter.com (2013). Beer In The Sun. https://www.kickstarter.com/projects/beerinthesun/beer-in-the-sun. Accessed: 2019-12-08.

Knowles, R. L. (1974). *Energy and form: an ecological approach to urban growth*. MIT Pr., Cambridge, Mass. [u.a.].

Kolbe, T., Gröger, G., and Plümer, L. (2005). CityGML - Interoperable access to 3D city models. *Geo-information for Disaster Management*.

Kollewe, J. (2018). London's skyline soars with record 510 tall buildings in pipeline. *The Guardian*. https://www.theguardian.com/business/2018/apr/18/londons-skyline-soars-with-record-510-tall-buildings-in-pipeline. Accessed: 2020-01-09.

Krutzler, D. (2013). Schneegarantie durch 19.000 Kanonen. *Der Standard*. https://www.derstandard.at/story/1361241351220/schneegarantie-durch-19000-kanonen. Accessed: 2020-01-02.

linuxfoundation.org (2019). Mapzen Open Source Data and Software for Real-Time Mapping Applications to Become A Linux Foundation Project. https://www.linuxfoundation.org/press-release/2019/01/mapzen-open-source-data-and-software-for-real-time-mapping-applications-to-become-a-linux-foundation-project/. Accessed: 2020-01-26.

livescience.com (2005). Ice Ages Blamed on Tilted Earth. https://www.livescience.com/6937-ice-ages-blamed-tilted-earth.html. Accessed: 2020-01-23.

mapbox.com (2019). Data. https://www.mapbox.com/data-platform/#mapbox-terrain. Accessed: 2019-02-02.

mapbox.com, Maps for Unity (2018). Mapbox for Unity. https://www.mapbox.com/unity/. Accessed: 2020-01-26.

mapzen.com (2016). Mapzen Terrain Tiles are 1.0 and ready to go. https://www.mapzen.com/blog/terrain-tile-service/. Accessed: 2020-01-22.

mapzen.com (2020). Tangram: Open-Source OpenGL Maps. https://www.mapzen.com/products/tangram/. Accessed: 2020-01-28.

Martin, T. and Tan, T.-S. (2004). Anti-aliasing and Continuity with Trapezoidal Shadow Maps. page 9.

McKean, M. A. (1981). *Environmental Protest and Citizen Politics in Japan*. University of California Press.

Mead, M. N. (2008). Benefits of sunlight: a bright spot for human health. *Environmental health perspectives*, 116(1):A160–7.

Miranda, F., Doraiswamy, H., Lage, M., Wilson, L., Hsieh, M., and Silva, C. T. (2019). Shadow Accrual Maps: Efficient Accumulation of City-Scale Shadows Over Time. *IEEE Transactions on Visualization and Computer Graphics*, 25(3):1559–1574.

Murshed, S. M., Lindsay, A., Picard, S., and Simons, A. (2018). PLANTING: Computing High Spatio-temporal Resolutions of Photovoltaic Potential of 3D City Models. In Mansourian, A., Pilesjö, P., Harrie, L., and van Lammeren, R., editors, *Geospatial Technologies for All*, pages 27–53. Springer International Publishing.

nasa.com (2018). Sun Fact Sheet. https://nssdc.gsfc.nasa.gov/planetary/factsheet/sunfact.html. Accessed: 2020-01-19.

Norbert Lechner (2019). Heliodons. http://webhome.auburn.edu/~lechnnm/heliodon/. Accessed: 2019-12-28.

odyssee-mure.eu (2020). Electricity consumption per dwelling. https://www.odyssee-mure.eu/publications/efficiency-by-sector/households/electricity-consumption-dwelling.html. Accessed: 2020-01-02.

Ohmura, A. (1968). The computation of direct insolation on a slope. *Climatological Bulletin*, 3.

openstreetmap.org (2020). OpenStreetMap. https://openstreetmap.org. Accessed: 2020-01-21.

osmbuildings.org (2020). OSM Buildings. https://osmbuildings.org. Accessed: 2020-01-28.

osmbuildings.org, Solutions (2020). OSM Buildings Solutions. https://osmbuildings.org/solutions/. Accessed: 2020-01-26.

ourworldindata.org (2019). Worldwide renewable energy generation. https://ourworldindata.org/renewable-energy. Accessed: 2020-01-01.

photopills.com (2020). PhotoPills. https://www.photopills.com. Accessed: 2020-01-14.

pintsinthesun.co.uk (2014). Pints In The Sun. http://pintsinthesun.co.uk/. Accessed: 2019-12-08.

planet.openstreetmap.org (2020). Planet OSM. https://planet.openstreetmap.org. Accessed: 2020-01-21.

proj4js.org (2020). Proj4js. http://proj4js.org. Accessed: 2020-01-30.

Proposition K (2020). Prop K - The Sunlight Ordinance. http://sfrecpark.org/wp-content/uploads/Item-2-858-Stanyan_AttachmentA_1989MemoOverview-060618.pdf. Accessed: 2020-01-09.

Quoctrung B., W. J. (2016). Mapping the Shadows of New York City: Every Building, Every Block. *The New York Times*. https://www.nytimes.com/interactive/2016/12/21/upshot/Mapping-the-Shadows-of-New-York-City.html. Accessed: 2019-12-09.

Redway3d Documentation (2019). Shadow mapping detailed. http://www.downloads.redway3d.com/downloads/public/documentation/bk_re_shadow_mapping_detailed.html. Accessed: 2020-01-08.

Reeves, W. T., Salesin, D. H., and Cook, R. L. (1987). Rendering antialiased shadows with depth maps. In *ACM Siggraph Computer Graphics*, volume 21, pages 283–291. ACM.

registry.opendata.aws (2020). Terrain Tiles. https://registry.opendata.aws/terrain-tiles/. Accessed: 2020-01-22.

Royer, F. (2009). The rotation of sun and stars. 21(529):265. ISBN: 9783540237822.

Schnauder, A. and Laufer, N. (2019). Geförderte Kredite in Millionenhöhe für Schneekanonen. *Der Standard*. https://www.derstandard.at/story/2000098088589/staat-schiesst-millionen-fuer-schneekanonen-zu. Accessed: 2020-01-02.

Segal, M., Korobkin, C., van Widenfelt, R., Foran, J., and Haeberli, P. (1992). Fast shadows and lighting effects using texture mapping. 26:249–252. ISBN: 0897914791.

solarsystem.nasa.gov (2019). Earth's Moon. https://solarsystem.nasa.gov/moons/earths-moon/in-depth/#surface_otp. Accessed: 2020-01-01.

Stamminger, M. and Drettakis, G. (2002). Perspective shadow maps. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques - SIGGRAPH '02*, page 557. ACM Press.

Straßer, W. (1974). *Schnelle Kurven- und Flächendarstellung auf grafischen Sichtgeräten.* PhD thesis.

SunCalc (2016). https://github.com/mourner/suncalc. Accessed: 2019-12-08.

threejs.org (2020a). three.js Documentation: DataTexture3D. https://threejs.org/docs/#api/en/textures/DataTexture3D. Accessed: 2020-01-18.

threejs.org (2020b). three.js – JavaScript 3D library. https://threejs.org. Accessed: 2020-01-28.

threejs.org/docs (2020). OrbitControls. https://threejs.org/docs/#examples/en/controls/OrbitControls. Accessed: 2020-02-03.

twitter.com (2014). SunTherapy. https://twitter.com/snips/status/452943384702119936. Accessed: 2019-12-08.

unity3d.com/de/webplayer (2018). Unity Webplayer. https://unity3d.com/de/webplayer. Accessed: 2020-01-26.

unity.com (2020). Unity Real-Time Development Platform. https://unity.com. Accessed: 2020-01-28.

verbund.com (2019). Vienna-Freudenau Run-of-River Plant. https://www.verbund.com/en-at/about-verbund/power-plants/our-power-plants/vienna-freudenau. Accessed: 2020-01-02.

Wai, K.-M., Yu, P. K. N., and Lam, K.-S. (2015). Reduction of Solar UV Radiation Due to Urban High-Rise Buildings – A Coupled Modelling Study. 10(8):1–11.

Wieland, M. and Wendel, J. (2015). Computing Solar Radiation on CityGML Building Data. pages 2–5.

wien.gv.at, LOD2 (2020). Generalisiertes Dachmodell (LOD2) - Produktinformation. https://www.wien.gv.at/stadtentwicklung/stadtvermessung/geodaten/dachmodell/produkt-lod2.html. Accessed: 2020-01-20.

wien.gv.at, LOD3 (2020). Detailliertes Dachmodell (LOD3) - Produktinformation. https://www.wien.gv.at/stadtentwicklung/stadtvermessung/geodaten/dachmodell/produkt.html. Accessed: 2020-01-21.

wien.gv.at/stadtplan3d (2020). Stadtplan3D. https://www.wien.gv.at/stadtplan3d. Accessed: 2020-01-14.

wiki.openstreetmap.org (2018). OSM-4D. https://wiki.openstreetmap.org/wiki/DE:OSM-4D. Accessed: 2020-01-20.

wiki.openstreetmap.org (2019a). Slippy Map.
https://wiki.openstreetmap.org/wiki/Slippy_Map. Accessed: 2020-01-24.

wiki.openstreetmap.org (2019b). Slippy map tilenames.
https://wiki.openstreetmap.org/wiki/Slippy_map_tilenames.
Accessed: 2020-01-24.

wiki.openstreetmap.org (2019c). Zoom levels.
https://wiki.openstreetmap.org/wiki/Zoom_levels. Accessed:
2020-01-24.

wiki.openstreetmap.org (2020). Simple 3D buildings.
https://wiki.openstreetmap.org/wiki/Simple_3D_buildings. Accessed:
2020-01-20.

wikipedia.org (2019a). Axial tilt. https://en.wikipedia.org/wiki/Axial_tilt.
Accessed: 2020-01-23.

wikipedia.org (2019b). Stonehenge.
https://en.wikipedia.org/wiki/Stonehenge. Accessed: 2020-01-01.

wikipedia.org (2020a). Atmospheric refraction.
https://en.wikipedia.org/wiki/Atmospheric_refraction. Accessed:
2020-01-24.

wikipedia.org (2020b). Mariana Trench.
https://en.wikipedia.org/wiki/Mariana_Trench. Accessed: 2020-01-21.

wikipedia.org (2020c). Mount Everest.
https://en.wikipedia.org/wiki/Mount_Everest. Accessed: 2020-01-21.

wikipedia.org (2020d). Web Mercator projection.
https://en.wikipedia.org/wiki/Web_Mercator_projection. Accessed:
2020-01-24.

wikipedia.org (2020e). World Geodetic System.
https://en.wikipedia.org/wiki/World_Geodetic_System. Accessed:
2020-01-24.

wikipedia.org, A10 (2019). Apple A10 Fusion.
https://en.wikipedia.org/wiki/Apple_A10. Accessed: 2020-02-04.

wikipedia.org, A12 (2019). Apple A12.
https://en.wikipedia.org/wiki/Apple_A12. Accessed: 2020-02-04.

wikipedia.org, Interact. Vis. (2019). Interactive visualization.
https://en.wikipedia.org/wiki/Interactive_visualization. Accessed:
2020-02-04.

wikipedia.org, iPad (2020). iPad. https://de.wikipedia.org/wiki/IPad. Accessed: 2020-02-04.

wikipedia.org, Note 8 (2019). Samsung Galaxy Note 8. https://de.wikipedia.org/wiki/Samsung_Galaxy_Note_8. Accessed: 2020-02-04.

wikipedia.org, Real-time (2020). Real-time computing. https://en.wikipedia.org/wiki/Real-time_computing#Soft. Accessed: 2020-02-05.

Williams, L. (1978). Casting Curved Shadows on Curved Surfaces. *SIGGRAPH Comput. Graph.*, 12(3):270–274.

Williams, L., Barry, R., and Andrews, J. (1972). Application of Computed Global Radiation for Areas of High Relief. *Journal of Applied Meteorology*, 11:526–533.

Wimmer, M., Scherzer, D., and Purgathofer, W. (2004). Light space perspective shadow maps. *Rendering Techniques*, 2004:15th.

Wise, D. U. (1969). Pseudo-Radar Topographic Shadowing for Detection of Sub-Continental Sized Fracture Systems. In *Remote Sensing of Environment, VI*, page 603.

wko.at (2018). FACTSHEET – Technische Beschneiung in Österreich. https://www.wko.at/branchen/transport-verkehr/seilbahnen/factsheet-beschneiung.pdf. Accessed: 2020-01-02.

Zhang, F., Sun, H., Xu, L., and Lun, L. K. (2006). Parallel-split shadow maps for large-scale virtual environments. In *Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications*, pages 311–318. ACM.

Zielinska-Dabkowska, K. M. and Xavia, K. (2019). Protect our right to light. 568(7753):451–453.

Zobrist, G. and Sabharwal, C. (1992). *Progress in Computer Graphics*. Number Bd. 1 in Progress in Computer Graphics. Ablex Pub.