Automated texture mapping CityJSON 3D city models from oblique and nadir aerial imagery

Mehmet Buyukdemircioglu[1], Sander Oude Elberink[1]

[1] Faculty of Geoinformation Science and Earth Observation (ITC), University of Twente, Enschede, The Netherlands
(m.buyukdemircioglu, s.j.oudeelberink)@utwente.nl

Keywords: 3D city models, Digital Twin, CityJSON, Texture mapping, Photogrammetry

Abstract

The incorporation of detailed textures in 3D city models is crucial for enhancing their realism, as it adds depth and authenticity to the visual representation, thereby closely mimicking the surfaces and materials found in actual urban environments. Existing 3D city models can be enriched with energy-related roof and façade details, such as the material type (such as windows, green façades, bricks) and sunlight reflectance which can be derived from texture information. However, a common limitation of these models is their lack of very high resolution textures, which which reduces their realism and detail. Manually mapping textures onto each surface of a building is an exceptionally time-consuming and labor-intensive process, making it unfeasible for large-scale applications involving thousands of buildings. Therefore, an automated method is essential for texture mapping of 3D city models from aerial imagery. In this paper, we present CityJSON texture mapper – a python-based software tool for automated texture mapping of CityJSON-based 3D city models from oblique and nadir aerial imagery. Experimental results demonstrate the effectiveness of our approach in generating high-quality textured 3D city models, showcasing the potential for broader applications in geospatial analysis and decision-making. This research contributes to the ongoing efforts in enhancing the realism and usability of CityJSON-based 3D city models by enhancing them with their real textures from oblique aerial imagery. Texture mapped model can be explored at https://bit.ly/textured3dbag.

1. Introduction

1.1 Motivation

Textures are crucial for increasing the visual quality and realism of 3D city models. The integration of textures into 3D city models significantly enhances their utility across a broad spectrum of fields by providing a more accurate, realistic, and engaging representation of digital twins. As technology advances and the ability to generate and process high-resolution textures improves, the potential applications of textured 3D city models will continue to expand, offering new opportunities for innovation in urban development.

Ongoing climate change and urbanization pose challenges for societies in terms of environmental quality, energy management, and the health of citizens. Many old cities have a significant share of aged and historical buildings with unique and different street profiles from modern infrastructure, which raises additional challenges in the energy transition because of low energy labels and restrictions to required interventions. HERITAGE (2024) project aims to address these issues by creating an advanced sensing and design system focused on identifying, reducing, and preventing heat stress. This will be achieved through monitoring and design interventions targeting the ageing built environments and buildings in old cities.

The HERITAGE project aims to enhance the existing 3D city models of cities by adding details such as building materials and the sun reflectance/albedo values of roofs and façades. This can be achieved by enriching the CityJSON-based 3D city model with information derived from earth observation data, including very high-resolution images. However, a common limitation of the most 3D city models is their lack of very high resolution textures. The process of manually mapping textures to every surface of a building is extremely time-consuming and labor-intensive task, rendering it impractical for large-scale projects that involve thousands of buildings. Within the HERITAGE project, addressing this challenge requires the development of an automated texture mapping process. This would enable the extraction of sun reflectance or material information from texture images, which can then be utilized to enhance 3D city models.

With the rapid advancement in sensor technology, very high-resolution oblique imagery is gaining popularity and is increasingly being acquired by numerous countries and municipalities for different projects. These imaging capabilities extend beyond their traditional use in producing orthophoto basemaps; they are also becoming invaluable for the texture mapping of 3D city models. High resolution textures not only improves the quality of a 3D city model; but also broadens their potential uses in urban planning, simulation, and visualization. Very high resolution textures can also be used to enrich existing 3D city models by extracting materials for roofs and facades from texture images, followed by the calculation of reflectance/albedo values for each building.

In this paper, we introduce the CityJSON Texture Mapper, a Python-based software tool for automated texture mapping CityJSON-based 3D city models from oblique and nadir aerial imagery. We address problem that the literature lacks a developed solution for automated texture mapping specifically designed for CityJSON 3D city models. Our study tackles a set of complex, interrelated challenges specific to the texture mapping of CityJSON 3D city models on a large scale such as integration of different types of aerial imagery, the emphasis on automation and scalability, and the focus on overcoming practical obstacles like obstructions and dataset misalignments. CityJSON Texture Mapper significantly reduces the time and labor required for manual texture mapping, making it feasible to generate textured 3D city models on a large scale. Our experimental results show that software's capability to produce high-quality, textured 3D city models that offer improved visual realism and depth.

1.2 Related Work

Semantic 3D city models, represented by standards such as CityGML (Gröger and Plümer, 2012) and CityJSON ((Ledoux et al., 2019), serve as the backbone for a various applications. CityJSON is based on JSON (JavaScript Object Notation), a lightweight data-interchange format that is easy to read and write for humans and easy to parse and generate for machines. This simplicity leads to smaller file sizes compared to CityGML, making data storage and transmission more efficient. Semantic 3D city models thrive on interoperability and the ability to integrate diverse data sources. While these models provide a structured and semantically rich representation of city elements, the incorporation of high-resolution textures is pivotal for realizing their full potential.

There are existing solutions proposed in the literature for automated texture mapping of semantic 3D city models. Buyukdemircioglu et al. (2018) demonstrated that large-format stereo nadir images can be used for texturing CityGML-based LoD2 3D city models. Frueh et al. (2004) and Kang et al. (2016) have shown that oblique images are more suitable for texturing building facades in 3D city models. This is due to the improved visibility of facades that oblique images provide, making them a widely used resource for both façade and roof texturing. Geo-referenced terrestrial images also can be used for texture mapping 3D city models (Kada et al., 2005). The use of images captured by Unmanned Aerial Vehicles (UAVs) is becoming increasingly popular and widely adopted. These images can also be effectively utilized for texture mapping in CityGML-based 3D city models (He et al., 2022). CityJSON based 3D city models are gaining popularity; however, the literature currently lacks automated solution for texture mapping CityJSON-based 3D city models. In this research, we tackled the challenge of the absence of software for automated texture mapping of CityJSON-based 3D city models from aerial imagery.

2. Dataset

Texture mapping process was performed by utilizing oblique and nadir aerial images, along with CityJSON-based 3D city model as the input data. A total of 35 oblique and 6 nadir aerial images of Enschede city center were captured using the Leica Citymapper sensor by Cyclomedia Technology BV (2024) in 2019. Oblique images were obtained using four distinct cameras, each positioned at a 45-degree angle relative to the Leica Citymapper sensor. The images were captured from an average height of 1500 meters above the terrain with ground sampling distance (GSD) of 5 centimeters.

In the process of aerial imagery acquisition, the saturation of colors is often overlooked, as the primary application of such imagery is for the development of orthophoto or true orthophoto basemaps. However, for crafting detailed and better looking textures, the combination of image sharpness and brightness, along with the vividness of colors, becomes critically important for better visual presentation of the textured 3D city models. To achieve more realistic and better looking building textures, it is necessarry to apply some color enhancement procedures to images.

These processes are designed to improve the visual quality and clarity of the image for better and more realistic textures. This involves the application of different image processing techniques such as contrast stretching, color enhancement, histogram equalization and bit depth adjustment which aims at refining the natural appearance of color tones in the texture. After enhancement, the images gain enhanced vibrancy, clarity, and realism. An example of aerial image before and after radiometric enhancement is given in Figure 1.

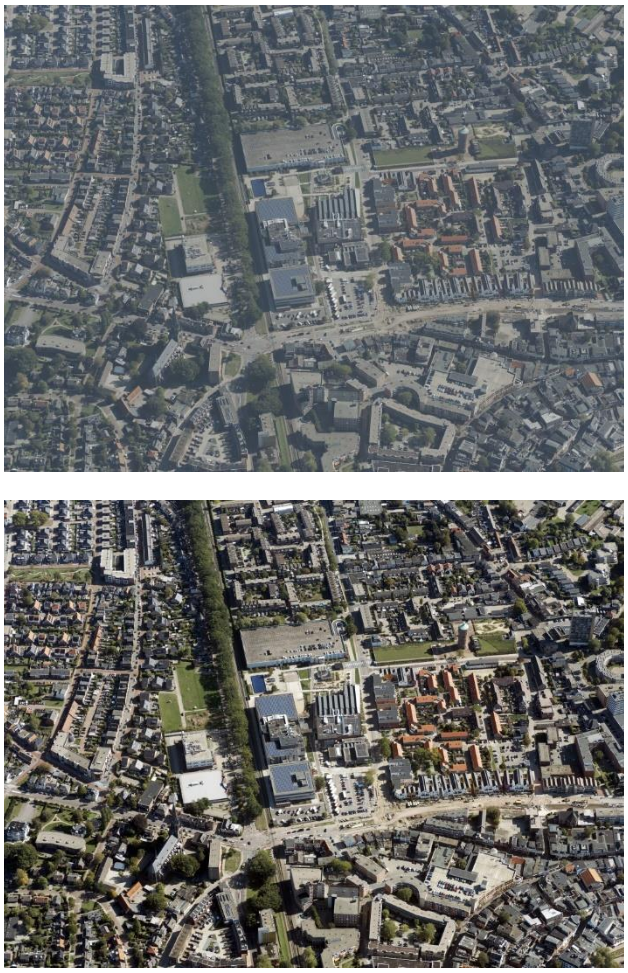


Figure 1. Raw aerial (top) and enhanced aerial image (bottom).

For texture mapping purpose, CityJSON-based 3D city model western European city center with 1346 buildings was selected for performing texture mapping. 3DBAG includes various levels of detail for building models, such as LOD1.1, LOD1.2, and LOD2.2. Due to the absence of roof geometries in the LOD1 family, this study utilizes LOD2.2 building geometries proposed by Biljecki et al. (2016), which also contains roof structures. Incorporating buildings with roof geometries allows for a better understanding of the urban landscape, which is particularly important for applications requiring high precision, such as urban planning, simulation, and for performing analyses on the model. An overview of the study area can be seen in Figure 2.

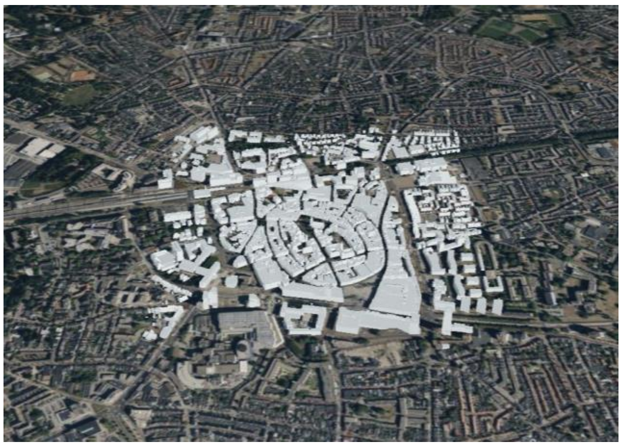


Figure 2. An overview of the study area

3. Method

In this section, we provide a detailed explanation of the implemented workflow. Initially, the process begins with checking image registration of building models on aerial images. This is followed by a thorough explanation of the selection of optimal images, and the detailed procedure for texture mapping.

3.1 Image Orientation Check

The 3D building models in 3DBAG automatically reconstructed using AHN (2024) laser scanning data. This dataset is collected at different time and with different sensor in comparison to aerial imagery. Therefore, the compatibility of these two different data needs to be checked. Before initiating the texture mapping process, it is essential to verify that both the 3D city model and aerial images (extrinsics) are aligned within the same coordinate system. Otherwise, the surfaces of the building models will not project accurately onto the aerial images. If the 3D city model and aerial images are defined in different coordinate systems, they must be converted to the same coordinate system. In the context of this study, given that both the CityJSON 3D city model from 3DBAG, and the external orientation parameters of the imagery are defined in the same coordinate system (EPSG: 7415), reprojection was not necessary for either of the datasets.

As the next step, the surfaces of the 3D building models need to be projected onto the raw aerial images for clipping corresponding textures. It is necessary to compute the pixel coordinates for each vertex of the 3D building models' surfaces, matching them with the aerial images. This process can be executed by leveraging the exterior orientation parameters of the aerial images alongside the interior orientation parameters of the camera. Exterior orientation parameters consist of the three coordinates representing the projection center, which define the camera's translation from the origin to its position at the time of exposure, and the three parameters that describe the rotation, such as the angles of rotation around the three camera axes. Camera interior orientation parameters include camera characteristics like focal length, pixel size and principal point.

Reprojecting a 3D point from the real world into 2D sensor coordinates within an image entails a series of three transformations: from world coordinates to camera coordinates, from camera coordinates to image coordinates, and finally from image coordinates to the sensor coordinate system. It is possible to determine the corresponding pixel coordinates of the building roof and façade surfaces and map them onto the imagery with Direct Linear Transformation (DLT). Equation 1 describes the mathematical representation of a pinhole camera, which is commonly referred to as perspective projection:

$$x = x_0 - c\frac{X - X_0}{Z - Z_0}; y = y_0 - c\frac{Y - Y_0}{Z - Z_0}, \qquad (1)$$

where c = principal distance (focal lenght)
x, y = image coordinates
X_0, Y_0, Z_0 = coordinates of projection centre
X, Y, Z = object coordinates in real world

The formula adjusts the 3D point coordinates by translating them relative to the camera position (X_0, Y_0, Z_0) and then scaling by the principal distance c divided by the depth ($Z - Z_0$) to project the point onto the 2D image plane. Forstner and Wrobel (2016) provide a comprehensive exposition on Direct Linear Transformation (DLT) and the process of transforming 3D world coordinates into 2D pixel coordinates.

Custom Python scripts were developed to visualize projected building surfaces on aerial images and evaluate the compatibility between the datasets. Therefore, a visual inspection was conducted to verify the data compatibility prior to initiating the texture mapping procedure. Figure 3 illustrates the projection of roof geometry onto a raw aerial image.

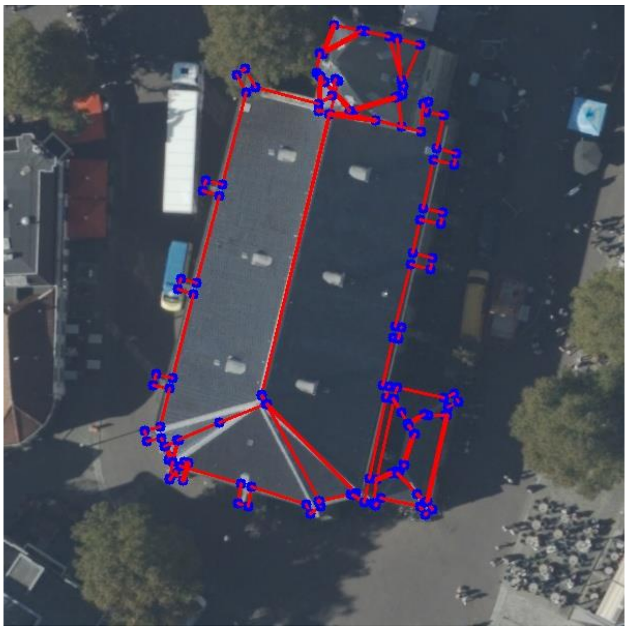


Figure 3. Projection of 3DBAG roof geometry on aerial image

3.2 Optimal Image Selection

A single building could be visible in multiple images, depending on the forward and lateral overlap ratio of aerial images. Similarly, facades or roof surfaces of a building can be visible in multiple images captured from different positions of the aircraft. As an example, during a photogrammetric data collection process where images are captured with an 80% forward overlap and a 60% lateral overlap, some buildings may be visible in up to 15 different images. While not every surface of a building is visible in each image where the building appears, it can still be visible in multiple images. For this purpose, it becomes essential to determine which image will be used for texture mapping each building surface.

A novel approach for selecting the optimal image was implemented as a part of the texture mapping process. Every surface of the building, whether it be the roof or façade that is selected for texture mapping, is processed individually for texture mapping process. CityJSON texture mapper software provides functionality for texture mapping different semantic surfaces of the building, including options to texture only the roof, only the facades, or to texture all surfaces. As a part of texture mapping process, semantic surfaces of the building are seperated, categorizing them into façade and roof surfaces for individual texture mapping.

The texture mapping process operates on a per-building basis, iterating through each building within the dataset. At first stage, the algorithm processes through the entire dataset of images for selected building, filtering out the images where the building is not fully visible. This approach helps to avoid unnecessarry data processing, especially for large projects with hundreds of aerial images. Roof and façade surfaces of the building are seperately processed for texture mapping. The visibility of the building in an image does not guarantee that all of its surfaces are visible. Even the building is visible in aerial image, facade surface visibility depends on the location of the plane at the time of image acquisition. As a consequence, only the facades facing the direction of the plane are visible in each image, while façade surfaces facing in the opposite direction are not visible.

The initial step involves checking whether the selected surface is visible in images where the building is visible. This determination relies on both the position of the aircraft and the surface of the building that is being texture mapped. As a next step, selected surface is projected onto all aerial images where it is visible, and the texture area of the projected surface on each image is calculated. In the final step, both datasets are considered together for each surface, and the image with the largest texture area among the photographs where that surface is visible is selected as the optimal image for texture mapping. This iterative process is applied to all surfaces of the building for texture mapping. If the surface is not visible in any image, it is excluded from the texture mapping process, and a "Null" value is assigned. An example of façade surfaces projected on selected optimal images for texture mapping is given in Figure 4.

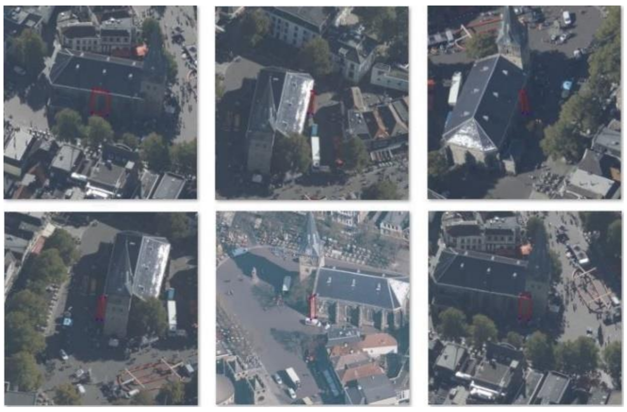
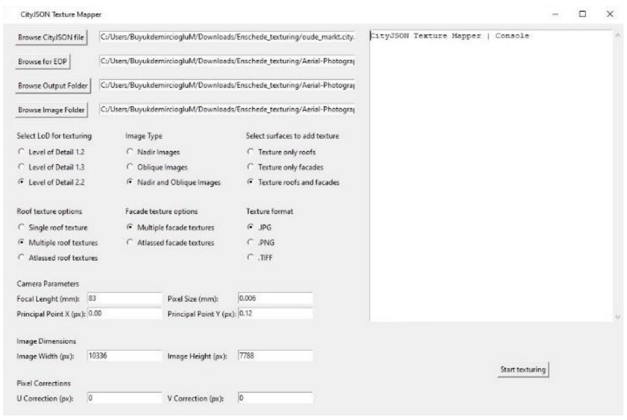


Figure 4. Optimal image selection for facade texture mapping

3.3 Texture mapping process

After the software completes the selection of optimal images, the subsequent step is to extract the textures of the building surfaces from these selected images and map them to the respective surface. The algorithm divides the building into sub-components, such as RoofSurface and WallSurface, and categorizes each into separate classes. Next, every surface of the roof and facade is mapped onto the chosen optimal image, and the pixel coordinates of the surface vertices on that image are calculated. Using the surface vertice coordinates derived from the calculated pixel coordinates, the portion of the image corresponding to that surface is clipped from the selected optimal image. The textures extracted from each aerial image are unique and exclusively allocated to one facade of a single building, guaranteeing that there is no overlap in their usage.

The software supports exporting textures in various formats. User has the option to choose the texture format based on the quality of texture needed. In scenarios where fast scene rendering and enhanced performance are critical, image formats such as JPEG are favored due to its reduced file size. Although image formats like Portable Network Graphics (PNG) offer lossless compression, they require more storage space and can lead to slower scene loading times. They are preferred in specific scenarios where the need for higher quality textures is essential. The graphical user interface of the CityJSON texture mapper software and processing parameters that can be defined by the user is given in Figure 5.



CityJSON Texture Mapper

Browse CityJSON file: C:/Users/BuyukdemirciogluM/Downloads/Enschede_texturing/oude_markt.city.
Browse for EOP: C:/Users/BuyukdemirciogluM/Downloads/Enschede_texturing/Aerial-Photogra|
Browse Output Folder: C:/Users/BuyukdemirciogluM/Downloads/Enschede_texturing/Aerial-Photogra|
Browse Image Folder: C:/Users/BuyukdemirciogluM/Downloads/Enschede_texturing/Aerial-Photogra|

Select LoD for texturing: Level of Detail 1.2 / Level of Detail 1.3 / [✓ Level of Detail 2.2]
Image Type: Nadir Images / Oblique Images / [✓ Nadir and Oblique Images]
Select surfaces to add texture: Texture only roofs / Texture only facades / [✓ Texture roofs and facades]

Roof texture options: Single roof texture / [✓ Multiple roof textures] / Atlassed roof textures
Facade texture options: [✓ Multiple facade textures] / Atlassed facade textures
Texture format: [✓ .JPG] / .PNG / .TIFF

Camera Parameters
Focal Lenght (mm): 83
Pixel Size (mm): 0.006
Principal Point X (px): 0.00
Principal Point Y (px): 0.12

Image Dimensions
Image Width (px): 10336
Image Height (px): 7788

Pixel Corrections
U Correction (px): 0
V Correction (px): 0

CityJSON Texture Mapper | Console

Start texturing

Figure 5. CityJSON texture mapper GUI

3D building models in 3DBAG feature 3 different level of detail for buildings, including LOD 1.2, LOD 1.3, and LOD 2.2. The software provides the functionality to apply texture mapping to the selected level of detail for buildings. In this study, building models with Level of Detail 2.2 (LOD2.2) is selected for texture mapping purposes due to included roof geometries. The roof geometry of the building can be texture mapped in two distinct options, as a single texture extracted from a single nadir image, typically the one most perpendicular to the building roof, resulting in all roof geometries being mapped from a unified roof texture. Alternatively, it can be texture mapped from multiple textures clipped from optimal oblique or nadir aerial images from the dataset, where each optimal image provides the most suitable texture for its corresponding roof surface. Texture mapping the roof from a single texture offers a more homogenous appearance, yet in many instances, texture mapping the vertical surfaces of the roof proves challenging due to the nadir viewing angle. The process of utilizing multiple textures for the roof starts by identifying the optimal image for each roof surface, followed by individually clipping the texture for each roof surface from the corresponding optimal aerial image. The software also provides the option to exclusively texture map specific surface types, such as roof surfaces, facades or roof and facades. An illustration showcasing examples of roof-only texture mapping, façade-only texture mapping, and fully texture mapped building is given in Figure 6.

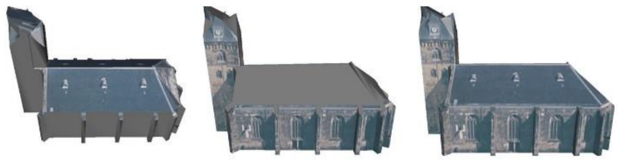


Figure 6. Roof, façade and fully texture mapped building

Following the texture clipping process, the file path of the texture and its corresponding texture coordinates for each textured surface must be written in the CityJSON file. The pixel coordinates from the images are converted into u,v coordinates for each surface and then integrated into the CityJSON file. Additionally, the texture format and file location for each surface are included in the CityJSON file. A CityJSON Geometry Object can contain a member named "texture" to store surface textures. This member comprises key-value pairs, where the key denotes the theme of the textures, and the value is a JSON object featuring a "values" member. This "values" member consists of a hierarchy of integer arrays. Each array represents a ring of vertices on a surface, with the first value indicating its position within the "textures" member of the "appearance" section in the CityJSON object. Subsequent indices denote the UV positions of corresponding vertices, drawn from the "boundaries" member of the geometry. Therefore, each array includes one additional value compared to the number of vertices in the ring. The array's depth aligns with the CityJSON Geometry object's depth and corresponds to that of the "boundary" array. The user has the flexibility to determine the precision of texture coordinates, including the number of decimal places. Increasing precision of the texture coordinates will result in a more precise fit to the surface; however, this will also lead to an increase in the size of the CityJSON file.

A total of 48,497 textures is clipped and written into the CityJSON file for 1,346 buildings. The texturing process was executed on a Dell Precision 3561 (i7 11800H – 16GB RAM) and required roughly 3 minutes and 16 seconds to complete. On average, 247 surfaces were texture mapped per second. This performance indicates that the CityJSON texture mapper executes the texturing process significantly faster than commercial software alternatives. During the texture mapping process, a copy of the original CityJSON file is created, and all modifications are applied to this file. The modified version is then saved as a separate CityJSON file, along with the textures, ensuring that all original attributes of the buildings remain intact.

After completing the texture mapping process, the textured 3D city model becomes ready for visualization. The texture-mapped 3D city model is visualized in CesiumJS (2024). As of now, there isn't an open-source solution that supports for visualizing texture-mapped CityJSON 3D city models. Given that CesiumJS does not natively support CityJSON, the textured CityJSON file is converted to CityGML format using citygml-tools (2024) for visualization in CesiumJS. This conversion step is necessary to bridge the compatibility gap between the CityJSON format and the visualization capabilities of CesiumJS. A view of the original and texture mapped 3D city model is given in Figure 7.

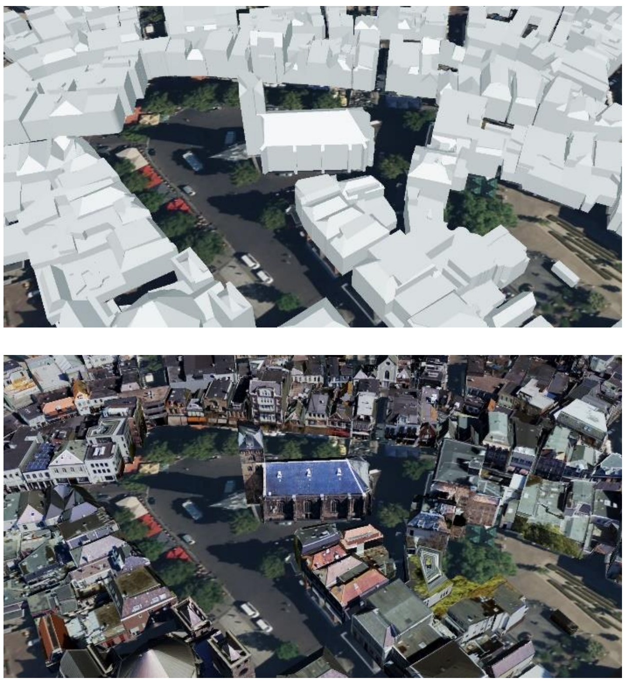


Figure 7. 3D city model before texture mapping (top) and after texture mapping (bottom)

4. Discussion

Several factors significantly influence the quality and accuracy of texture mapping. Among these, the precision of the exterior orientation parameters stands out as crucial. Inadequate accuracy in these parameters leads to improper projection of building geometries onto the images, resulting in a misalignment between the building's geometry and its texture. Orientation angles and center coordinates significantly influence façade texturing. Texturing quality improves near the photograph's outer edges.

There is a slight misalignment between aerial images and 3D building geometries in the 3DBAG model, likely because they are derived from different data sources. Nowadays, graph neural networks have demonstrated successful results in delineating roof structures from aerial photographs (Zhao et al., 2022). Since that the models in our study are automatically reconstructed from aerial laser scanning point clouds and 2D building footprints, integrating roof lines extracted from aerial imagery via graph neural networks may address misalignments between the two datasets. This integration could lead to the reconstruction of more precise roof geometries and aligned textures with the building surfaces.

Another significant challenge in texture mapping arises from obstructions like trees that obscure roofs and facades, reducing the visual quality of the texture mapping. Objects surrounding or atop buildings can obscure the structure, leading to poor texture application. Likewise, buildings occluded by adjacent structures often suffer from incorrect texture mapping, where the obscured surface might be inaccurately covered with the texture of the obstructing building. To mitigate this issue, a visibility analysis should be conducted prior to texture mapping to ensure accurate representation. The visibility analysis should take into account the buildings surrounding the texture-mapped building, and, if feasible, other objects like trees, considering their actual height and width. If the surface is partially obscured in the image, exploring existing deep learning techniques to eliminate objects from the texture image could be beneficial.

Furthermore, the timing of image acquisition throughout the day affects the presence of shadows on various parts of the building and within the image itself. The literature offers shadow removal techniques based on deep learning, which are effective in eliminating shadows from images, which could be a future work. Optimal image capture times are when the sun is directly overhead, minimizing shadow effects and enhancing the quality of the texture mapping. Taking these factors into account will optimize the texture mapping process and enhance the texture quality, resulting in an improved visual representation of textured 3D city models.

5. Conclusion and Future Works

In this study, we introduce CityJSON Texture Mapper, a Python-based software for automated texture mapping CityJSON-based 3D city models from oblique aerial imagery. The absence of existing automated solutions for texture mapping CityJSON 3D city models from aerial imagery in literature highlights the novelty of the study. Developed tool processes CityJSON 3D city models and aerial imagery together, then automatically maps textures to building surfaces such as roofs and walls. It offers various customizable options, including the level of detail for texture application, choice of surfaces for texturing (roofs, facades, or both), and texture quality. A live demo of the texture mapped 3D city model can be explored at https://bit.ly/textured3dbag (2024).

For future developments, we aim to implement advanced visibility analysis that considers surrounding buildings on texture mapping visibility. This enhancement will have the ability to detect situations where a shorter building is blocked from view by a taller one nearby, thus preventing the need to employ obstructed images for texturing those concealed areas. Additionally, the application of deep learning, particularly Generative Adversarial Networks (GANs), could further enhance texture resolution. GANs have been successfully employed in several studies to significantly improve image resolution, which could be beneficial for creating high-quality building textures.

Another potential direction is to automate the extraction and modeling of building facade details, such as windows and doors, from the textures, allowing for detailed 3D models even in the absence of explicit texture information. This could make untextured 3D city models more informative and visually rich, or can be used to upgrade LOD2 family buildings to LOD3. The use of 3D city models in energy-related applications, like urban energy simulations and solar potential assessments, is also growing. Textures play a crucial role in these applications by providing critical data for identifying roof and facade materials, allowing for the assignment of accurate albedo values to enhance building models for energy studies. Future research will focus on extracting reflectance values and identifying material types of buildings to enhance existing 3D city models from textures for applications in energy efficiency.

Significant improvements could be achieved through texture packaging, which consolidates all texture images of a building into a single image. This method not only minimizes the city model's file size by reducing duplicate texture coordinates and file paths but also accelerates scene loading times, as renderers would need to load a single packaged texture instead of multiple images for each surface.

Given that the 3D city model is reconstructed from the AHN point cloud and textures are derived from oblique aerial images captured by a different sensor at a different time, such misalignment between these two datasets is expected. However, as a future work is planned for implementing a correction mechanism to detect and rectify misalignments and changes between aerial images and the 3D city model. Implementing this approach will result in improved texture mapping on building surfaces, subsequently enhancing the extraction of information for building roofs and facades, such as sunlight reflectance and material properties for energy-related applications.

Acknowledgements

This paper is part of the 4TU-programme HERITAGE (HEat Robustness In relation To AGEing cities). HERITAGE is funded by the 4TU-programme High Tech for a Sustainable Future (HTSF). 4TU is the federation of the four technical universities in The Netherlands (Delft University of Technology, DUT; Eindhoven University of Technology, TU/e; University of Twente, UT and Wageningen University and Research, WUR).

References

3DBAG. 2021. 3D Geoinformation, Delft University of Technology. https://www.3dbag.nl (22 May 2024)

AHN. 2024. Actueel Hoogtebestand Nederland. https://www.ahn.nl. (22 May 2024)

Biljecki, F., Ledoux, H., & Stoter, J., 2016. An improved LOD specification for 3D building models. Computers, Environment and Urban Systems, 59, 25-37. https://doi.org/10.1016/j.compenvurbsys.2016.04.005

Buyukdemircioglu, M., Kocaman, S., Isikdag, U., 2018. Semi-automatic 3D city model generation from large-format aerial images. ISPRS International Journal of Geo-Information, 7(9), 339. https://doi.org/10.3390/ijgi7090339

CesiumJS - 3D geospatial visualization for the web. 2024. https://cesium.com/platform/cesiumjs/ (22 May 2024)

citygml-tools. 2024. https://github.com/citygml4j/citygml-tools (22 May 2024)

Cyclomedia Technology BV. 2024. https://www.cyclomedia.com/en (22 May 2024)

Förstner, W., Wrobel, B., 2016: *Photogrammetric Computer Vision*. Springer Nature, Cham. https://doi.org/10.1007/978-3-319-11550-4

Frueh, C., Sammon, R., Zakhor, A., 2004. Automated texture mapping of 3D city models with oblique aerial imagery. Proceedings. 2nd International Symposium on 3D Data Processing, Visualization and Transmission, 2004. 3DPVT 2004. pp. 396-403. https://doi.org/10.1109/TDPVT.2004.1335266

Gröger, G., & Plümer, L., 2012. CityGML–Interoperable semantic 3D city models. ISPRS Journal of Photogrammetry and Remote Sensing, 71, 12-33. https://doi.org/10.1016/j.isprsjprs.2012.04.004

He H., Yu J., Cheng P., Wang Y., Zhu Y., Lin T., Dai G., 2022. Automatic, Multiview, Coplanar Extraction for CityGML Building Model Texture Mapping. Remote Sensing, 14(1), 50. https://doi.org/10.3390/rs14010050

Kada, M., Klinec, D., Haala, N., 2005. Facade Texturing for rendering 3D city models. ASPRS 2005 (pp. 78-85).

Kang, J., Deng, F., Li, X., & Wan, F., 2016. Automatic texture reconstruction of 3d city model from oblique images. The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, 41, 341-347. https://doi.org/10.5194/isprs-archives-XLI-B1-341-2016

Ledoux, H., Arroyo Ohori, K., Kumar, K., Dukai, B., Labetski, A., & Vitalis, S., 2019. CityJSON: A compact and easy-to-use encoding of the CityGML data model. Open Geospatial Data, Software and Standards, 4(1), 1-12. https://doi.org/10.1186/s40965-019-0064-0

Textured 3DBAG. 2024. https://bit.ly/textured3dbag (22 May 2024)

The HEat Robustness In relation To AGEing cities (HERITAGE) Program. 2024. https://www.4tu.nl/heritage/ (22 May 2024)