




Implementing integrated digital twin modelling and representation into the Snap4City platform for smart city solutions

L. Adreani^{1,2} · P. Bellini^{1,2} · C. Colombo^{2,3} · M. Fanfani^{1,2,3} · P. Nesi^{1,2}  · G. Pantaleo^{1,2} · R. Pisanu^{2,3}

Received: 2 August 2022 / Revised: 24 August 2023 / Accepted: 31 August 2023 /
Published online: 5 October 2023
© The Author(s) 2023

Abstract

Recently, Digital Twins solutions have attracted a growing interest as a fundamental paradigm for managing data driven processes on smart cities. They are complex modelling that should include 3D interactive representations of buildings and infrastructures, integrated with a wide range of data for Smart City cyber-physical ecosystem monitoring and controlling. This paper presents a framework for modelling, generating and distributing Digital Twin representations with 3D models from a various set of data, as well as its integration into the open-source Smart City framework, where many kinds of real time and historical data are available. The proposed solution offers a method for creating integrated data rendering of 3D city entities coupled with Smart City data (e.g., IoT Devices with time-series and historical data, heatmaps, geometries and shapes related to traffic flows, bus routes/stops, cycling paths). The solution for generating 3D representation is based on a number of computer vision and machine learning solutions, thus shortening the activities of passing from raw data (i.e., Lidar, shapes, patterns, etc.) to 3D representations. Implementation has been enforced into the quite widespread open-source Snap4City Smart City platform and has been validated by using hundreds of buildings in Florence city central area, Italy, plus hundreds of thousands of data as points of interest, IoT Devices, traffic flows, dynamic heatmaps, etc.

Keywords Digital Twin · 3D City model · Smart City Modelling

1 Introduction

Smart Cities are complex infrastructures integrating multiple linked data sources, Internet of Things (IoT) devices and applications, involving many different data and stakeholders. In this context, spatial data information may act as enablers for smart applications and decision support systems, provided that they are interoperable with legacy and future solutions [1]. Recently, aspects related to digital 3D city modelling and Digital Twin (DT) have gained a growing interest, since they allow to create a more realistic context where decision

makers can perform analyses, simulations, demonstrations, planning and monitoring in several different domains and application areas (e.g., urban planning, energy management, traffic and mobility, disaster management, air pollution monitoring). As pointed out in [2], the most popular interpretation of DTs in a smart city context has been the geospatial mapping of urban areas to 3D models combined with associated data from existing city datasets, together with real-time data from smart city deployments. Many technological standards and approaches have been proposed in literature, such as: CityGML, CityJSON, the combination of Building Information Modelling (BIM) and Geographic Information System (GIS), which provides a City Information Modelling (CIM) [3]. An efficient city DT framework should include the integration of BIM and GIS-based city information with the smart city data asset [4], also exploiting sensor technologies (Internet of Things—IoT, 5G, etc.), data analytic algorithms (Artificial Intelligence, Machine and Deep Learning, etc.) to finally provide an in-depth and real-time insight of large and complex physical systems, which would otherwise be almost impossible to obtain [5]. However, a typical issue that such systems have to manage deals with limitations in data accessibility, shareability and interoperability [6], especially in complex domains such as smart cities.

In past years, much research has been made in the field of 3D city modelling, to recreate realistic visualizations. However, due to the typical city size, handling all the data and their processing is a challenging task that is still unsolved [7]. One critical aspect of developing diffuse 3D city models is to find the correct generative approach and format for the data to be rendered by a graphic interface on web browser. For this purpose, a set of requirements has been proposed by CityGML, according to different Levels of Detail (LoD) addressed by the model. According to [8], there are five levels: LoD0 is represented by models having only a 2D map with 3D terrain; LoD1 adds buildings as simple boxes; LoD2 adds rooftops details to LoD1; LoD3 presents also external facade structures; LoD4 adds building interiors. LoD4 was introduced in CityGML 2.0, but it was removed in the latest version of CityGML 3.0. Both CityGML and CityJSON have defined a format to represent geometry and topology for 3D buildings, using respectively XML and JSON. CityGML 3.0 integrates a BIM standard, alongside the GIS format, from Industrial Foundation Class (IFC) [9]. Some integrations of CityGML have been proposed in real cases, like the city of Helsinki, where a LoD3 city model was implemented and made publicly available [10]. However, this system does not provide integration with IoT data or other kinds of city data. Another similar integration was made by the city of Rotterdam [11], recreating a LoD2 type of buildings, with no integration of decoration elements, nor elevation of terrain (this is a relevant aspect for non-flat cities). An attempt of making a LoD3 3D city model was made by ETH Zurich with the VarCity [12]. However, the provided semantic information is generally limited to a small number of semantic classes. The 3dcitydb implements a 3D model for the city of Berlin [13], providing a pickable model of LoD2 buildings, supporting also WMS (Web Map Service) layers (typical of GIS solutions providing maps, heatmaps and orthomaps) and terrain layer. However, no heatmaps are provided, only some different ground map representations. The city of Stockholm [14] has implemented many aspects of Digital Twin concept, such as POI (point of interest), LoD3 type buildings, either with 3D tiles or modelled one, and others 3D entities. However, this solution does not implement any WMS heatmap.

In the context of 3D city data collection, advancements in Digital Surface Model (DSM) creation from Light Detection And Ranging (LiDAR) or aerial/satellite imaging technology has allowed the modelling of urban topography at a spatial resolution and granularity which were not achievable before this technology [15, 16]. In [17], a method to create a city model from a point cloud generated by LiDAR technology is presented.

Despite its efficiency, this approach cannot process asymmetrical objects and is somewhat geometrically inaccurate. Creating 3D models of buildings with accurate details (e.g., the shape of the rooftop) is a more complex task. Some solutions have addressed this problem using a model-driven approach [18, 19], where parameterized primitives (i.e., precomputed roof models collected in a library) are fitted on point cloud data. However, complex buildings like the ones than can be found in historical urban areas cannot be correctly modelled using a standard set of roof primitives. Hence, in complex contexts, data-driven approaches that model three-dimensional shapes by means of computer vision and computational geometry [20, 21], appear to be more adequate. These methods typically start by extracting geometrical entities such as outer borders, step lines, planar surfaces, etc., and then proceed to reconstruct the 3D building model. Since DSM data is frequently corrupted by noise, robust estimation and regularization techniques can be used to improve results. A more realistic 3D city representations can be obtained by enhancing building shapes with textures extracted from RGB images. In particular, rooftops textures can be obtained from orthomaps or satellite images. However, this is not an easy task: at first, rooftops have to be detected in the RGB images; then, the segmented patches must be carefully aligned with the top-view of the 3D map. Indeed, even if geolocalization information is typically available, errors occur due to uncertainties [22] and an accurate multi-modal registration is required (e.g., between the RGB images and the 3D structure) [23]. In literature, several works have addressed these topics using both computer vision standard and learning-based solutions. In [24], handcrafted features and a hierarchical segmentation approach have been used to identify buildings in rural areas. SVMs (support Vector Machines) [25] and Random Forests [26] have also been used to address this task. For example, in [27] authors proposed a three-steps method based on colour-based clustering, roof detection using an SVM and a final false negative recovery. Slightly different, in [28] a pair-wise exploitation of satellite images has been used to reconstruct a 3D model that could then be employed to identify rooftop regions. However, such solutions, not only have some limitations when working on areas with dense buildings, but also require a successive registration on the 3D map. More recently, deep learning-based solutions appeared for remote sensed image processing [29]. In [30], a Mask R-CNN (region based convolutional neural network) [31] was used to detect rooftops from aerial images. Differently, in [32, 33] a U-Net architecture [34] has been preferred. Moreover, these last two solutions have provided not only rooftop segmentation, but also the registration on 3D data.

Façade texturing is even more complex, also due to lack of data. If rooftop textures can be extracted from orthomaps covering large portions of a city, façades require indeed more dedicated acquisitions, since a single image can only include one or few buildings. Moreover, in addition to the segmentation and registration problems, façade images need to be rectified by removing any distortion introduced during image acquisition.

In this paper we are presenting a 3D City Modelling Framework for Smart City Digital Twin, covering more data with respect to those addressed in the LoD classification and including terrain elevation, roads, building planimetries, maps, orthomaps, heatmaps, buildings, high value buildings with meshes, building extruded from their plant shape, roof reproduction from LiDAR data, pattern extraction and positioning for roofs and facades, traffic flow data, IOT, traffic flows, bus routes, cycling paths, etc. In this paper, the main focus is on:

- (i) the production process from raw data to 3D DT elements. This process is very complex, since a large number of heterogeneous data need to be collected and referred

- one another. Due to their volume and complexity the activity has to be performed automatically and special algorithms and tools have to be created.
- (ii) the integrated model for DT representation and distribution on web. The modelling of a DT implies the capability of creating a model that can be easily deployed and distributed progressively from a server to a browser with the limitation of that platform.
 - (iii) the satisfaction of a large range of requirements to allow both modelling and interaction with 3D structures. The DT model usage has to provide a number of features to allow its practical usage so as to grant support to decision makers when interacting with city structures and observing the details of a scene.
 - (iv) performance in production and distribution of the resulted integrated DT model. According to progressive web distribution of 3D DT, the approach is totally different from what would be deployed on game applications to prevent from requiring huge amount of memory and dedicated CPU/GPU.

The 3D representation is enriched with: 3D representation of crests for traffic flows, 3D shapes and dynamic PINs which can manifest data values for real time IoT data representation, heatmaps and animations, picking functionality for building and data elements, and interactivity with all the elements from dashboards. The 3D DT has been developed in the context of the open-source Snap4City platform and framework [35, 36], to provide an environment where smart applications can be easily created also by exploiting 3D DT model visualization with all extended smart city data, as mentioned above. The work presented in this paper follows the same research line of [37] which presented early results in mapping orthomaps on roofs only, addressing less requirements, which are now fully considered and are coped with in the next section, thus enabling the performance of a less limited validation on those techniques. Snap4City (<https://www.snap4city.org>) is an open-source smart city IoT platform which can be used to collect a large range of smart city / IoT data with the aim of: making data aggregation; computing data analytics as predictions, anomaly detection, indicators, etc., [38, 39]; and showing data on 2D dashboards with maps, time series and a large range of data representation graphics [36].

The paper is organized as follows: in Section 2, the identified requirements of Digital Twin for smart city solutions are presented, comparing them with those proposed in the state of the art. In Section 3, model architecture and data process are described, highlighting the complexity of the process and the main enabling elements which are discussed in deep hereafter. The 3D model representation of the Digital Twin is detailed in Section 4, stressing the mechanism for its assembly and distribution. In Section 5, details of the main production processes for 3D representation are described. They include the construction and texturing processing for buildings. Section 6 focusses on computational costs and performance for 3D model production and its scalability. The same section also includes some examples of the distribution performance. Conclusions are drawn in Section 7.

2 Requirements analysis

With the aim of creating a DT in the context of smart cities, the 3D representation of buildings in the city plays a relevant role. To this end, a set of specific requirements has been identified and is reported in this section. In the past, a similar approach has been proposed by CityGML which defined different levels of detail (LoD) for the models [8].

The CityGML approach was mainly on visual represented, and it is actually not detailed enough to describe the needs of full DT models in the Smart City solutions for decision makers. *Therefore, a more complete set of requirements and an assessment model for Web delivering of 3D representations of DTs at disposal of decision support systems are presented in this section.* Most requirements are related to the 3D representation and to the integration of 3D data with the massive data infrastructure in back, which actually supports decision makers, while all other aspects are addressed by former tools based on Snap4City [35, 40, 41]. In particular, the DT solution for smart city has to provide support to represent in a 3D context, what follows:

R1. Buildings of the city structure, roads, gardens, etc. The single building should be represented with details in terms of shape (facades, roofs, towers, cupolas, etc.), and patterns on facades and roofs. To this end, different techniques can be adopted to model buildings. For example, (i) extruding the bounding box of buildings as obtained from the perimeter up to the heights of eaves, (ii) creating meshes describing details of the physical structure. Building perimeters (i.e., planimetry) can be obtained from open services like those offered by OpenStreetMap (<https://www.openstreetmap.org/>), however such data show a lack of information regarding the buildings' height or their roof structure. The heights of eaves can be obtained from manual measures or exploiting aerial LiDAR measures or via stereo images to be converted into DSM. The RGB images required to extract the façade patterns are not easily accessible. On the other hand, roof patterns could be obtained by orthomaps, even if an accurate texture extraction is not straight forward, as described hereafter. Shapes of roads and gardens are usually accessible on GIS for their ground coverage, and rarely for 3D garden details.

R2. Ground information such as road shapes and names, names of squares and localities, etc., exploiting the so called Orthomaps, with possible real aerial view patterns. They are typically provided in terms of multi resolution tiled images from GIS systems using WMS protocol.

R3. One or more heatmaps superimposed (with transparency) on ground level information without overlapping buildings. For example, to represent some information, such as: heatmap of temperature, traffic flow, pollutant distribution, people flow, etc. Also in this case, they are typically provided in term of multi resolution geolocated tiled images, provided by GIS using WMS protocol. (ii) heatmaps should be located over the terrain model of R6. (iii) in some cases, a time sequence of heatmap is available to show the distribution evolution over time. This aspect adds complexity to the model.

R4. Paths and areas which can be (i) super-imposed on the ground and on heatmap levels without overlapping buildings, for example those needed to describe garden perimeters, cycling paths, trajectories, border of gov areas, traffic/people flow or density/velocity along paths, etc., (ii) elevated in the 3D form as crest. This piece of information is quite specific and has to be produced via some data analytics. The 2D version (i) can be distributed by using GIS in WFS/WMS protocols, while the (ii) version has to be distributed as 3D objects.

R5. Pin marking the position of services, IoT Devices, Point of Interest (POI), Key Performance Indicator (KPI), etc., and providing clickable information according to (i) some data model, (ii) Time Series of IoT Devices to show historical and real time data, according to (iii) shapes coming from GIS, etc. This information is quite specific and can be produced on the basis of the information recovered from Private and/or Open Data.

R6. Terrain information and elevation, so that the city skyline may include the shape of possible mountains around the city, and the correct elevation of the city ground. This also means that buildings and Orthomaps should be placed according to the terrain elevation (R3.ii). The terrain model can be recovered from LiDAR or flight scanning and may be distributed in files via institutional open data sites. It is typically called DTM (Digital Terrain Model), also provided in WMS from GIS.

R7. Additional 3D entities to add more realism to the scenario, such as: trees, benches, fountains, semaphores, digital signages, and any other city furniture, etc.

R8. Virtual 3D structures as dynamic PIN changing colour and/or size/shape according to some data value, OD flows with jumps/arcs, etc. Dynamic pins as SVG shape and colour by changing with some real time value. Dynamic pins as solid 3D changing colour or size according to some real time value.

In addition, a framework including the DT solution has to be capable to provide some interactivity on the above-mentioned 3D data structures, in particular it should be able to depict the 3D scene:

RA. according to the point of view, providing capabilities for changing it by means of: zooming, rotating, tilting, and panning the scene and also changing light or time of the day/night (this should lead to produce shadows projected by buildings on ground and other building, and a different illumination from direct to indirect exposition to daylight), etc.

RB. with the sky, maybe with different sky conditions according to the actual day, light condition, weather, or weather forecast.

RC. providing access to the information associated with augmenting PINs: POI, KPI, etc., and maybe to real time data, and time series associated with possible IoT Devices located on the 3D scene.

RD. providing the possibility of selecting each single building or PIN to: (i) shift to a more detailed information associated with the building, or (ii) go into a BIM view of the building, with the possibility of navigating into the building structure, and again to access the internal data associate to PINs into the building. You may also disable the building view to see only the city 3D without buildings, but with PINs.

RE. providing the possibility of selecting an element (3D, PIN, ground, heatmap) to cause a call back into a business logic tool for provoking events and actions in the systems, where developers may associate intelligence activities, analytics, other views, etc.

RF. providing the possibility of inspecting ground terrain and see detailed 3D elements placed in the underground, such as water pipes, or located in the ground as benches, luminaries, red lights, etc.

According to the identified requirements, in the following Table 1, an assessment of the most relevant solutions is reported.

3 Model architecture and data process

According to above-described requirements, a solution for Smart City Digital Twin, **SCDT**, has to cope with them, by implementing an integrated solution covering three main areas:

Table 1 Comparison of 3D representation platforms for Digital Twins vs Smart City

	CityGML [8]	Helsinki [10]	Rotterdam [11]	Berlin [13]	Stockholm [14]
R.1.i	Yes (LoD1)	No (only available in higher detail)	No (only available in higher detail)	No (only available in higher detail)	No (only available in higher detail)
R.1.ii	Yes (LoD3)	Yes (either with object or 3D tiles)	Yes (LoD2)	Yes (LoD2)	Yes (LoD3)
R.2	No	Yes	Yes (C)	Yes (C)	Yes (but with a fixed Orthomap)
R.3.i	No	No	No	Yes (does not include Wrms)	No
R.3.ii	No	No	No	No	No
R.3.iii	No	No	No	No	No
R.4.i	No	Yes (C)	Yes (C)	No (x)	Yes
R.4.ii	No	No	No	No	No
R.5.i	No	No	No	No	Yes
R.5.ii	No	No	No	No	No
R.5.iii	No	No	No	No	No
R.6	Yes	Yes (with 3D tiles)	No	No	Yes
R.7	Yes (LoD2)	Yes (with 3D tiles)	No	No	Yes (3d tiles and single entity)
R.8	No	No	No	No	Yes
RA	No(*)	Yes	Yes	Yes	Yes
RB	No(*)	No	No	No	No
RC	No(*)	No	No	No	Yes
RD.i	not clear (may be)	Yes (when models are loaded as object, not 3D)	Yes	Yes	No
RD.ii	No	No	No	No	No
RE	No(*)	No	No	No	Yes
RF	No(*)	Yes (**)	Yes (**)	No	No

Where: (*) defines only the building model, (**) functionality implemented in Cesium but without any model placed underground, (x) use Cesium, it could be possible, (C) based on Cesium

- (a) **3D model** representation of information in an integrated manner,
- (b) **software architecture for distribution, thus** providing access to 3D representation via a suitable user interface presenting the 3D model including the interactivity features and on demand data facilities, and
- (c) **production process** of 3D models by starting from multiple information which have to be recovered from accessible resources or produced/acquired, processed and integrated into the 3D model, and made available for distribution on demand.

Above-described requirements from R1 to R8 mainly impact on (a) and (b) for the resulting performance to distribute and reproduce DT representation in real time on browser, thus providing support for users to interact with 3D representation in real time. The system presents challenging aspects due to large amount of data to be processed on client side on the basis of the point of view. This impacts especially when several details are provided at the same time in the same view; e.g., textures/patterns, detailed heatmaps, complex terrain shapes, which implies to compute several projections to avoid overlaps, high valuable building in meshes, 3D shapes of each building, etc. In these cases, the issue is typically mitigated at the expense of a lower resolution of textures.

On the other hand, features from RA to RF have to be mainly satisfied by production process (c) of the data model to be distributed according to (a) and (b). In fact, the model can be composed by several elements: 3D representation, meshes, patterns, shapes, heatmaps, etc. The process of shifting from images and data into the integrated 3D model is not a trivial one and it is partially described in this paper as to some of its aspects. Such production process to produce the **SCDT** model is depicted in Fig. 1. This production process highlights data sources: GIS, raw images, building shapes, heatmaps, PINs, POI, IoT devices, DTM, LIDAR, etc.

According to Fig. 1, this production process for a 3D model’s creation requires a set of sub-processes:

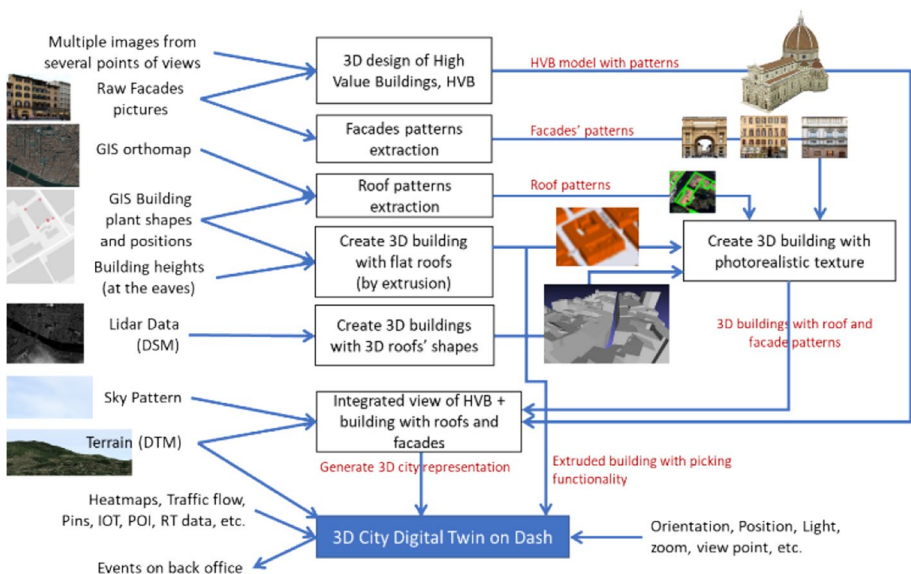


Fig. 1 Data flow of the production process for creating a Digital Twin for smart cities

- **3D design of High Value Buildings, HVBs:** in order to produce accurate representation of HVBs, a manual 3D design or automatic computer vision techniques (such as Structure from Motion) can be used. Precise measurements or specific image/video acquisition campaign and overlapped textures are required. Additionally, geo-localization information must be provided. The resulting textured 3D models can be exported as geo-localized glTF files.
- **Facades pattern extraction:** façade texturing requires a specific acquisition campaign. Acquired RGB images must be processed to remove radial and projective distortions. As to rooftops, obtained textures can be provided as PNG or JPEG files.
- **Roof pattern extraction:** photorealistic textures of building rooftops can be obtained from orthomaps. Since orthomaps are typically roughly geo-localized, a careful registration w.r.t. building shapes is required. After that, textures can be extracted and provided as PNG or JPEG files.
- **Create 3D buildings with flat roof (by extrusion):** given building shapes plus their height, typically measured at their eaves, simple 3D models with flat rooftop can be obtained. The resulting data format is a GeoJSON file with a height/elevation attribute to compute the building extrusion from the ground at run time. This is the model used to implement the 3D bounding box to enable the functionality of picking.
- **Create 3D building with 3D roof:** when a DSM is available, either obtained from LiDAR data or other acquisition modality, accurate 3D roof shapes can be obtained to build a more realistic SCDT. Building 3D models can be provided as glTF (GL Transmission Format) files, with geo-localization information.
- **Create 3D building with photorealistic textures:** 3D buildings obtained by extrusion or exploiting a DSM can be enhanced with photorealistic rooftop and façade patterns by applying textures extracted from RGB images. Textured building models are saved in glTF files, with geo-localization information.
- **Integrated view of HVBs + buildings with roof and facades:** building 3D models and HVB models are finally placed into a unique 3D representation exploiting their geo-localization information, thus obtaining the complete 3D representation for the SCDT.

General architecture for distributing SCDT includes a set of data integrating 3D models, meshes, with DTM. In addition, heatmap, traffic flow, Pins, IOT, POI, etc., can be dynamically loaded on demand as described in the paper.

For the distribution of the data:

- **3D representation File in GeoJSON via HTTPS:** it describes the 3D structure of the city and all information related to it. It is used to represent the city model in extruded mode and to retrieve buildings information or other BIM data for the picking functionality.
- **3D representation File in glTF/GLB (GLB is the binary version of glTF) via HTTPS:** it describes the city 3D structure in terms of buildings and their relationships with the other graphic elements: facades, meshes of HVB, textures and materials.
- **Pattern files via HTTPS:** pattern images for facades, roofs, DTM files in PNG format, Sky texture, etc.
- **orthomaps, maps, heatmaps, animated heatmaps, traffic flows, animated traffic flows,** etc., are provided limited to the portion of the map shown in the window frame from a GIS server via WMS over https (i.e., via the GeoServer integrated into the Snap4City platform).

- **semantic details in JSON on demand such as:** roads graph, POI, IoT data, Pins, cycling paths, vectorial traffic flows, etc., on the basis of the portion of the map shown in the window **via smart city API HTTPS from SuperService Map of Snap4City platform** [35, 41, 42].

4 3D model representation of the digital twin

The model for creating 3D representations, which allows to provide all above-mentioned information, is based on a hierarchical layered structure depicted in Fig. 2 and described in this section.

The layered solution has been implemented via WebGL API, in order to process all data in parallel, thanks to the GPU passthrough: to this end, the open-source library called Deck.gl has been used. All the layers needed for the representation of the Snap4City platform data types have been implemented and they are loaded at runtime on user demand. Thanks to the multi-layer structure of deck.gl, layers have been implemented individually with their own safe context, to avoid interferences one another. Every layer has its own scope, managing its own data type. Therefore, in the following we are introducing the implemented layers to describe data the types provided in the Snap4City 3D representation (see Fig. 3 for an example).

First, the base deck application has been realized by using a custom implementation and management of the viewState object, where all geographical information for the map (such as latitude, longitude, zoom, etc.), are defined. We have also implemented a custom rendering in order to add features like SkyBox that need direct access to the WebGL context. Starting from the first layer. The elevation of the terrain has been modelled by implementing a composite layer called *TileLayer*, which is used to divide maps in multiple tiles with their own sublayer: for each tile a sublayer called *TerrainLayer* has been created. Thus, the elevation map, in the form of DTM files, has been used to create the *TerrainLayer* 3D model from the map, while the background orthomap has been used as a texture of terrain objects. Once the DTM image has been retrieved, Martini tessellation is used to create the

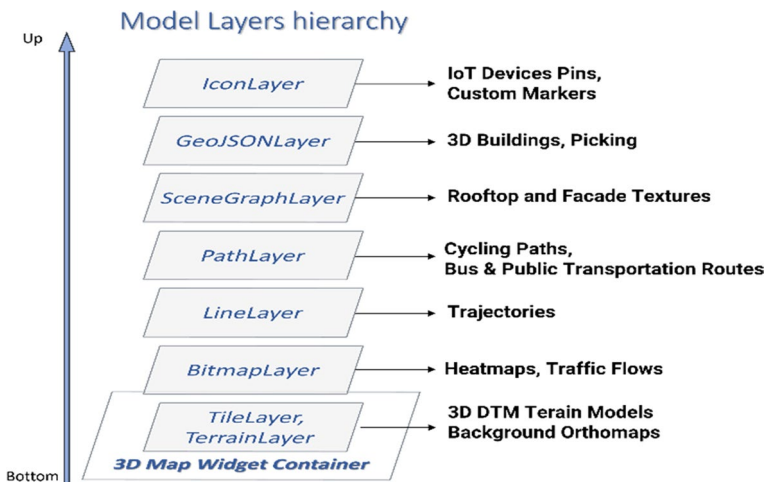


Fig. 2 Hierarchical layers structure of the model

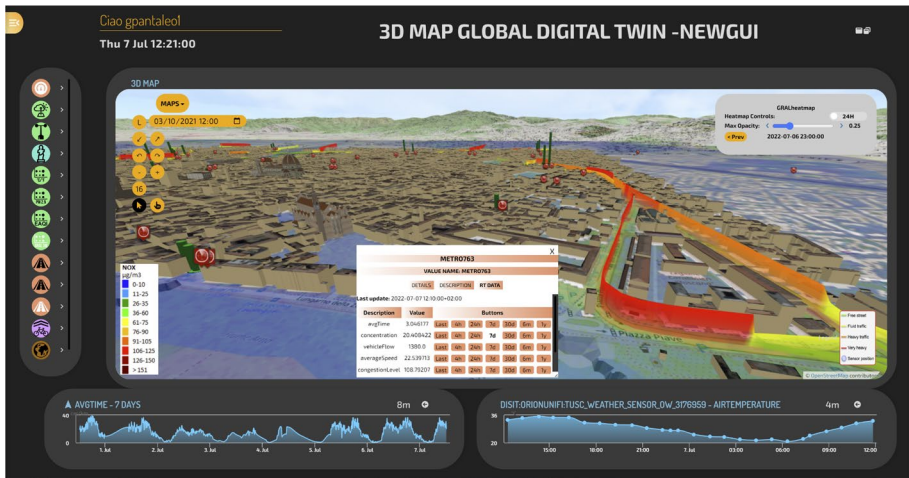


Fig. 3 Multi Data Map of Snap4City with addition of textures and HVB (e.g.: the Florence dome, Santa Croce basilica) [29, 31]. <https://www.snap4city.org/dashboardSmartCity/view/index.php?iddashboard=MzI5Mw> = accessible to all

mesh to replace any flat tile background. The result is a 3D representation of terrain with texture to represent better the territory.

The background orthomaps have been also implemented through a `TileLayer`. In this case, `BitmapLayer` to display an image in the map has been used. This method has been also used to represent heatmaps, which are essentials to provide a fast access / representation to large amounts of data. In order to implement heatmap visualization in `deck.gl`, we have used a composite layer which automatically retrieves heatmaps from a dedicated geo-server (through several formats, including WMS) and displays them as an image. Heatmaps can be static or animated; static heatmaps are viewed as single PNG images, while animated ones are sent by the geo-server in GIF format, and they are later divided into multiple images and rendered sequentially with a customizable delay time.

For the implementation of data coming from different sources like IoT devices, trajectories, cycling paths, etc., various layers with a specific JSON mapping have been implemented. To display paths and geometries, different layers have been used depending on the geometry type to be displayed, i.e., `LineLayer` for trajectories, `PathLayer` for the cycling path. IoT devices are also displayed as pickable markers on the map. When a user selects one of them, a popup with the sensor information (static attributes, as well as real-time data, if available) is shown. Whenever the sensor provides real-time data, they can be displayed on dedicated widgets, such as time trends, upon request.

3D representations of buildings are provided in two manners: Extruded and Realistic (meshes, HVB). Extruded buildings are implemented by using a GeoJSON file, in order to have a faster loading time, and this is required because this type of buildings are loaded, even when the realistic ones are loaded. The extruded building may have flat roof of realistic roof based on LiDAR data. In both cases, the pattern taken from orthomaps has been mapped as described in the following.

Realistic buildings HVB (presenting photorealistic details, or other building with meshes) can be loaded as both `SceneGraph` and 3D tiles. In order to implement the picking functionality, we need also to render the extruded buildings underneath. The

Extruded type is totally described in a single GeoJSON file, where the following elements are defined for each building: the base polygon, the height, and various other attributes and pieces of information. The GeoJSON file is loaded in a layer called GeoJSONLayer and it is responsible to take all the features in the file and display them on the map, with the base polygon extruded by its height. In the case of Realistic building data type, we use the glTF and GLB formats to describe the scene, and they are loaded by the SceneGraphLayer. This type of integration works well to achieve impressive visualization without impacting too much on the application performances. 3D buildings can also be individually picked on map, in order to see all information on that building, not to mention any linking to dedicated BIM representations or other details, if available. When we need to retrieve information about a particular building with the picking functionality, the glTF/GLB format is not enough since it does not contain this kind of information. Therefore, all information about each and every building is retrieved on the 3D building GeoJSON file and dynamically loaded upon request.

A simulation of the sun position has been also implemented, to create the light and shadow in the scene; this can be useful to simulate when and where a particular zone is in the shadow in a particular hour of the day. In the future it may be also used to study natural heat distribution in the city, which may contribute to mitigate the problem of urban heat island effect, an issue all cities have to cope with.

Features working in a 2D environment need to be revised or completely modified when we are working in a rich and more complex 3D environment. For instance, a major requirement is to improve the visualization of shapes such as in traffic flow representations. Usually, traffic flows are represented by coloured lines in a 2D map, but those lines are then usually hidden in a 3D map by buildings or terrain. Therefore, in order to improve the visualization of traffic flows density in a 3D fashion, a new layer called CrestLayer has been created. In this layer, traffic density is displayed as raised crests whose amplitude are proportional to the computed traffic density values. The area under the crest is coloured using the four different colours of the traffic density palette (green, yellow, orange and red). Due to the problem nature, we need to work with data that can be fragmented or incomplete, so before initializing a CrestLayer we pre-process data in order to have a smoother representation. Every crest segment is composed by three parts: a middle one, where the value is the traffic density of that specific road segment, and two external ones, where densities are the average of all density values of roads connected to that road segment.

Another implemented feature is the visualization of dynamic pins. Dynamic pins allow to graphically represent sensor markers, changing shape and/or colour depending on the value of the metric they are associated to. In this way, dynamic pins enable a fast and immersive data-driven and event-driven visualization of data coming from physical or virtual sensors. Multiple views for different types of sensors can be exploited, ranging from dynamic SVG creation to 3D column representation of real-time data values. When a SVG is selected for pin visualization, the dynamic pin is created dynamically in the backend; then it is retrieved and displayed by using the IconLayer. In the event of a 3D column, a composite layer has been generated to recreate a thermometer effect, displaying the value of any sensors with a 3D cylinder shape whose height is proportional to the displayed metric value. The colour of the column typically represents the category of those sensors, and all these elements can be customized by the user.

5 Main processes for 3D representation production

In this section, details regarding the production process are reported: (A) roof pattern extraction, (B) façade pattern extraction, (C) creation of 3D buildings (with flat or 3D rooftops) and photo-realistic textures, and (D) integration of HVB and 3D buildings into a unique 3D representation.

A. Roof pattern extraction

Orthomaps of the city of Florence, kindly provided by the “*Sistema Informativo Territoriale ed Ambientale*” (GIS Local) of Tuscany Region, were used to obtain roof textures. These RGB photos are tiles with a resolution of 8200×6200 pixels, with partial overlap and rough geo-localization in the EPSG 3003 (Monte Mario / Italy zone 1) coordinate system.

As a first step aerial images and 2D GIS building shapes (expressed in the EPSG 4326 coordinate system – Geodetic Parameter Dataset, originally created by European Petroleum Survey Group) were converted into a common coordinate reference system. Multiple orthomap tiles describing the considered area were fused into a single mosaic image using the Geospatial Data Abstraction Library, GDAL (<https://gdal.org/>). Then, we down-sampled the mosaic image by a factor of $\frac{1}{4}$, to obtain a relevant speed-up in the successive steps, without losing accuracy, as the chosen image resolution allows both rooftop detection and alignment deep net (see hereafter) to operate optimally.

To detect the rooftops from the orthomaps and align them with the building shapes, we used the method presented in [33], based on a double U-Net architecture exploiting multi-resolution [42] and multi-task learning. The net takes as input an RGB orthomap and the corresponding cadastral map (represented as a binary image), and outputs a list of multi-polygons aligned with the RGB image. To obtain the cadastral map, the 2D shapes of the buildings were converted into a raster binary image. The output multi-polygons, up-scaled to consider the image down-sampling previously done, were then exploited to both extract rooftop textures (from the full resolution mosaic) and align them with the 2D building shapes. An affine transformation to warp the mosaic orthomaps and register it with respect to 2D building shapes was computed. However, using a single transformation for all the multi-polygons would have given rise to local inaccuracies. For this reason, we decided to compute a dedicated transformation for each multi-polygon and locally warped the image to obtain a better registration. Specifically, given vertexes of an aligned multi-polygon V_A and vertexes of a corresponding 2D shape V_S an affine transformation T was estimated such as

$$V_S = TV_A \quad (1)$$

Then, according to the estimated T , the orthomap was warped and the considered rooftop was extracted. After repeating this process for all multi-polygons, a complete warped orthomap (including only the rooftops) was obtained and exported as a JPEG file. Note that, while exporting the texture image, different resolution can be used to obtain smaller weights and faster visualization.

B. Façades patten extraction

In order to properly texture the façades of 3D building models, some image acquisition campaigns were carried out in some of the most representative locations of Florence. We did not use a wide-angle lens to avoid introducing strong radial distortions, therefore,

when a building was too large to be captured in a single frame, we employed photo-mosaic [43] to obtain a full façade image. Residual slight distortion effects were removed in post-production using camera calibration data.

In order to create textures to be applied to the 3D model building façade, the acquired images must be rectified, i.e., they must be reduced to orthographic views, where any projective distortion is removed. This was accomplished by warping the images with specialized homography transformations computed between the vertices of each façade and a square pattern of 300×300 pixels. Note that, even if this solution alters the original aspect ratio of the façade, this was not a problem, since when applying the texture on the 3D model the correct proportions were restored. In Fig. 4 both original image and rectified façade are shown.

C. Creation of 3D model with photorealistic textures

Building models with flat roofs By exploiting the height attributes included for example in a GeoJSON file, it is possible to obtain 3D models of buildings with flat rooftops. Such result can be simply obtained by extruding 2D building shapes using the *BlenderGIS* library according to the given height attribute.

Building models with 3D roofs The case of 3D rooftops is much more complex. Indeed, if DSM data are available, image processing and computational geometry algorithms must be devised and employed to obtain an accurate representation of buildings. DSM data used in this work were also kindly provided by the “*Sistema Informativo Territoriale ed Ambientale*” (GIS local system) of Tuscany Region. They were obtained from a LiDAR survey and are composed by several tiles covering the city of Florence, with a resolution of 1 square meter per pixel. Moreover, given the complexity and density of buildings in Florence, in our solution we started by cropping the DSM using 2D building shapes to select the DSM

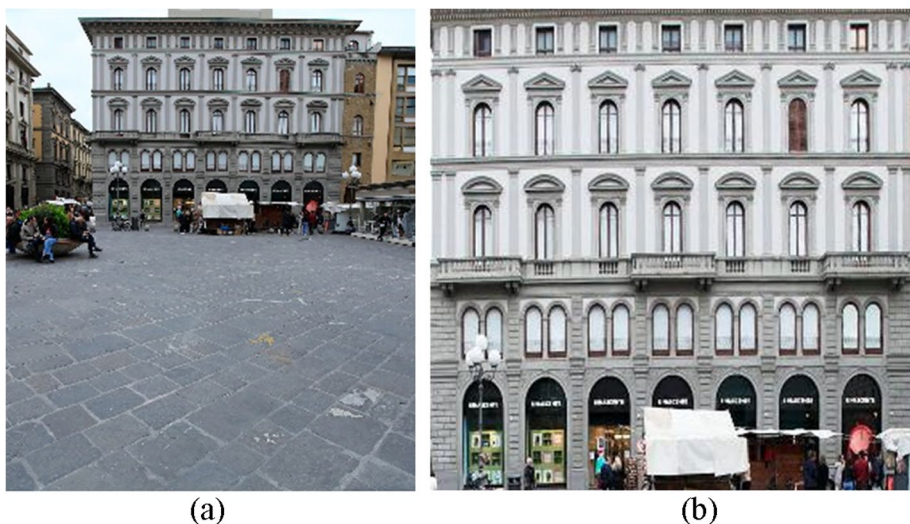


Fig. 4 Façade rectification. In **a** the original image, in **b** the rectified façade

pixels of a single building. The cropped DSM image was then smoothed using a bilateral filter [44] to attenuate the noise and preserve the content (see Fig. 5a).

A region growing algorithm [45] was used to cluster different portions of the building according to their elevation that henceforth will be referred to as *height-clusters*. Starting from the highest point, neighbours in a 3×3 window, centred on the source point, were evaluated and associated with the same cluster, if their height difference with respect to the central point was inferior to a user defined threshold (set at 0.25 m in our experiments). The process was iterated on all newly associated points until no more points could be included. Then, new clusters were defined in a similar way on remaining points. As can be seen in Fig. 5b, region growing tends to produce an over-segmentation of the rooftop, so smaller clusters (with less than 10 points) were removed and progressively associated with other bigger clusters. Points on the edge between two adjacent height-clusters were selected and, after applying morphological thinning to remove redundant points, they were used to regress step-lines with a linear estimation method embedded into a RANSAC framework [46] to reject outliers. Note that, since the edge between two height-clusters can be composed by multiple line segments, a solution inspired by the J-Linkage algorithm [47] was exploited. Specifically, given a set of edge points, all possible combinations of point pairs were used to estimate candidate line segments – this combinatorial strategy is viable in this context, since the number of edge points was found to be always relatively small in our experiments. Each line candidate was then scored by considering the number of points (i.e., the inliers) close to the estimated line according to a threshold, set to 0.5 pixels, and the sparsity of those points along the line. Starting from the candidate with the highest number of inliers and with denser points, line segments were selected. Since the great majority of rooftops is of quasi-rectangular shape, only candidates that were near orthogonal or near parallel to the first selected candidate were retained. Finally, similar

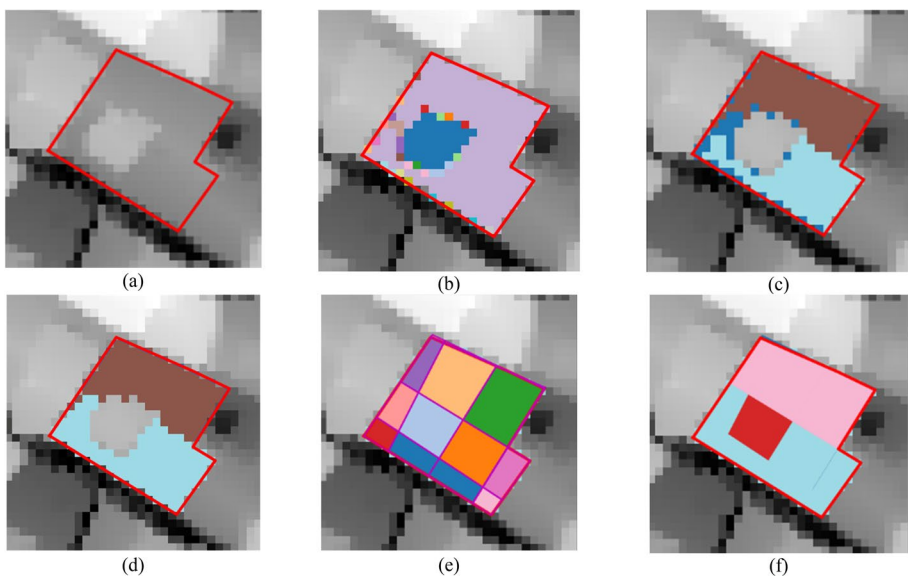


Fig. 5 Computational steps of pipeline to obtain building model with 3D roof from LiDAR based DSM data. **a** input DSM with the building shape polygon superimposed in red, **b** initial output of the region growing clustering, **c** an intermediate step of the plane-cluster expansion, **d** final plane-clusters, **e** rooftop planar patches, **f** planar roof segments obtained after fusion of planar patches

candidates (in terms of position and direction) were grouped together and used to estimate refined step-lines.

A second clustering phase was carried out into each of the previously identified height-clusters, so as to possibly split it into different sub-clusters (referred to as *plane-clusters*) sharing a common plane orientation. To achieve that, for each point of the height-cluster 3D normal vectors were estimated, together with gradient magnitude and direction. The HDBSCAN algorithm [48] was then used leveraging on a custom weight matrix $W = \{w_{ij}\}$ modelling pairwise distances between each pair of points. W was built as follows: For each pair of points i, j the weight w_{ij} is defined according to the angular difference between the normal vectors N_{ij} , the difference on gradient magnitude M_{ij} and direction D_{ij} , and the sum of the L1 distances on the X and Y coordinates L_{ij} . All metrics are independently normalized in the interval $[0,1]$. Finally, w_{ij} is then obtained as

$$w_{ij} = \begin{cases} \infty, & \text{if } L_{ij} > 6 \\ w_N N_{ij} + w_M M_{ij} + w_D D_{ij} + w_L L_{ij}, & \text{otherwise} \end{cases}$$

where (w_N, w_M, w_D, w_L) are weights, respectively set to $(10, 0.1, 5, 1)$. Note that, if $w_{ij} = \infty$, HDBSCAN ignores this pairwise relationship and the points i, j will fall in the same cluster, provided that there is a path between the two points containing only finite distances. Using the weight matrix, closed points with similar normal vectors and gradient are grouped together in plane-clusters. However, several points remain not associated to any plane-cluster. To solve this, we have proceeded as follows.

First, we used all points in a plane-cluster to robustly estimate the corresponding 3D plane. Then, non-associated points were progressively included to one of the plane-clusters, according to their proximity to the cluster and their distance with respect to its 3D plane (see Fig. 5c-d). In a similar way to what occurred with height-clusters, edge points on adjacent plane-clusters were used to compute hip-lines (i.e., lines that separate two roof parts with different plane orientation): However, in this case, since two planes can only intersect in a single line, a standard RANSAC algorithm was used.

Finally, borderlines obtained from the 2D shape polygon of the building were also estimated. We noticed that, sometimes, borderlines were very similar if not identical to the step-lines previously estimated. We devised a heuristic to assess the similarity between borderlines and step-lines, by considering their behaviour wrt nearby roof. If a step-line was found to be equal to a borderline, we discarded the estimated step-line, keeping only the borderline, since this was probably more accurate being obtained from the shape polygon.

Borderlines, step-lines, and hip-lines were then used to split the whole rooftop into planar patches by evaluating the intersection of lines and recovering all resulting closed polygons (see Fig. 5e). In this way we were certain that each of these polygons included points belonging to the same 3D plane. Then, each planar patch associated to polygons was labelled according to height and plane-clusters, and patches with the same label were finally fused to obtain the minimal number of planar roof segments (expressed as concave or convex polygons) covering the whole rooftop, see Fig. 5f.

For each roof segment, points falling into it were used to refine the estimate of common 3D plane. A triangulation step was carried out to obtain triangular meshes of segments as planar straight-line graph, with a particular care to deal with concave polygons. Then, elevations (i.e., the Z-coordinate) of segment vertexes were computed exploiting estimated 3D planes.

Once the rooftop modelling has been completed, building walls were constructed by computing all points at the foot of the perpendicular for each roof vertex. Note that DTM

information was exploited to consider the ground elevation. In Fig. 6, the final 3D model is shown, as an example coherent with data of Fig. 5.

Both flat-roof and 3D-roof models are integrated into glTF/GLB files.

3D model texturing At this moment the texturing phase was carried out using Blender. As to rooftops, the warped orthomap, obtained with the method described in Section 5. A was used to texture the polygonal faces corresponding to rooftops using the *Python Blender API*. Note that, when using flat-roof models, roof faces can be easily retrieved by selecting those having a vertical normal vector. Differently, when using 3D roof computed from the DSM, the selection process was carried out beforehand. When creating the building model, we used special attention to split roofs from the building's walls, thus avoiding any problem during the texturing phase.

Façade textures were instead manually applied on lateral surfaces of buildings (i.e., their walls) by selecting meshes and using the *project_from_view* Blender function, that automatically adapts the squared textures obtained in Sect. 5. B on the façade surface.

In order to show how much effective is our strategy to build 3D model with photo-realistic textures, we present in Fig. 7 the reconstruction of the area between Costa San Giorgio and *Costa dei Magnoli* in Florence (they are accessible from the 3D view of DT of Florence reported in Fig. 3 and related accessible URL). From an inspection of the figure, building models with 3D rooftops and elevated according to the DTM data are shown. Moreover, roof and façade textures were applied to increase realism of such 3D map. Notwithstanding slight modelling errors, the reconstructed buildings are accurate enough to provide a high-fidelity 3D map, ready to be exploited as a Smart City digital twin.

D. HVB integration

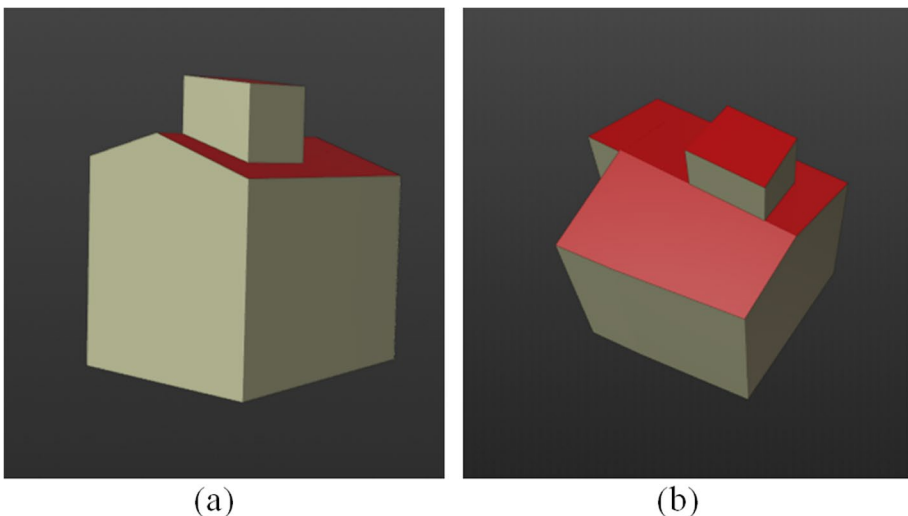


Fig. 6 Different view of the obtained building model with 3D roofs

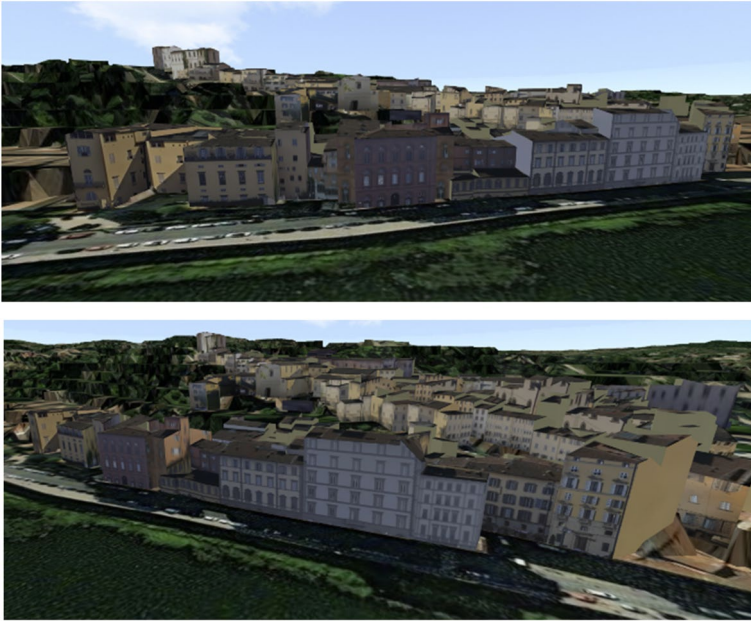


Fig. 7 Two views of the 3D reconstruction of the area between Costa San Giorgio and Costa dei Magnoli in Florence, featuring building models with 3D rooftops elevated according to the DTM data, and roof and façade photorealistic textures. The ground layer is textured using RGB data sampled from orthomaps

Using *Blender*, we were also able to include and geo-locate in the map the HVB 3D models. For example, as shown in Fig. 8, an accurate 3D reconstruction of Santa Maria del Fiore Cathedral (Florence Dome) was placed into the 3D representation, thus achieving a nicer final result. The final 3D map textured model with also HVB models was exported in **gITF/GLB** format (including 3D geometries, textures, and coordinates) ready to be deployed in the Snap4City platform using the *SceneGraphLayer of the deck.gl framework* (<https://deck.gl/>).

Fig. 8 An example of integration of a HVB into the 3D map (in this case Santa Maria del Fiore Cathedral in Florence)



6 3D digital twin representation performance

In this section, the computational costs and performance for 3D model production and its scalability to a whole city, are discussed. The same section also includes some issues regarding performance in distributing a possible representation. It was not possible to compare the performance of our complex process with respect to state-of-the-art performance, since other solutions are mainly based on manual operations and are not published.

Before discussing performance aspects, the framework context where performance has been assessed is briefly described.

Snap4City is a 100% open-source platform developed at DISIT Lab, University of Florence (<https://www.snap4city.org/>), [40, 49]. The platform manages heterogeneous data sources, such as: IoT devices (city sensors and actuators, as well as private devices, supporting a large variety of brokers and protocols), open data, external services. For each different kind of data, static attributes (such as geographical information and other metadata) and real-time data are collected. Device data are semantically indexed in an RDF Knowledge Base, thus they can be retrieved by dedicated APIs and exploited by Data Analytics processes and IoT applications to perform analyses, simulations, forecasts, etc. This allows users to produce new knowledge on data, which can be shown on user interface through Dashboards and a wide range of widgets (showing data both in pull and push modalities). The integration of the 3D city model into the Snap4City platform was realized using a Multi-Data Map Widget which can be instantiated into any Dashboard. It presents an interactive 3D environment of the city, and it grants the possibility of inspecting different kinds of city entities and data, such as: IoT devices, Points of Interests (POI), heatmaps, geometries related to bus routes, cycle paths, traffic flows, etc. In this way, the Snap4City platform allows to exploit a complete open-source framework that can collect, process, and manage all the data needed to obtain a high-fidelity Smart City Digital Twin, SCDT. Currently, Snap4City is used in several cities and multiple installations. Each single installation can be multitenant and thus multiple cities or areas can share the same deploy and features.

Therefore, to integrate the 3D representations in the Multi-Data Map Widget of Snap4City platform (thus creating a 3D Multi-Data Map Widget), the `deck.gl` open-source library has been used and extended, as described in Section 5. By exploiting the multi-layer structure of `deck.gl`, we have implemented distinct layers for each type of data supported by the Snap4City platform. All layers can be viewed and removed dynamically upon the user's choice via a dedicated menu on 3D Multi-Data Map Widget. Figure 3 represents the 3D Multi-Data Map Widget of the city with 3D models and textures obtained using the method described in Section 5, and the whole architecture of Section 3. The tool is freely accessible on web and includes heatmaps, traffic flow sensors, traffic flow data, animations, PINs for IOT and POI, etc. On this regard, an `IconLayer` was implemented in `deck.gl` to represent all IoT devices managed by the platform. In Snap4City, IoT devices are ingested and stored in a semantic Knowledge Base, and they are classified by semantic categories [42, 49]. Therefore, a pool with different icons for each type of device category is used to represent device markers on map. The user can access to all information given by a specific sensor and city element by simply clicking on the device PIN; in this way, a popup is shown presenting static attributes and, when available, real-time and historical data can be selected and viewed on dedicated time-trend and single-content widgets.

A. Roof pattern extraction performance assessment

To obtain a quantitative validation of any rooftop extraction results on our data (process described in Sect. 5. A), we manually have created a set of ground-truth multi-polygon for 200 buildings scattered uniformly on the covered area. Then, we have evaluated the Intersection over Union (IoU) between the ground-truth and the input (non-aligned) and the output (aligned) multi-polygons.

In Fig. 9, a bar plot showing the IoU score obtained for each considered building is reported. As can be seen, for almost all test cases (only in four cases the input multi-polygons have higher IoU), the IoU has increased when using the output multi-polygons, thus confirming the effectiveness of our approach. In average we could obtain an IoU score of 71% for the input multi-polygons, and 88.54% for the output multi-polygons after aligning them by using the deep network, with an increase of almost 17.5 points which is a percentage increment 19.8%.

As to computational times, two main phases must be considered: the alignment obtained through the deep net, and the orthomap warping and segmentation. The alignment net run on a NVIDIA TITAN Xp GPU, with 12 GB of RAM. In our experiment we noticed that the required execution time does not directly depend on the number of multi-polygons (i.e., buildings) considered, but instead on the dimension of the orthomap given in input. For example, using an orthomap covering the 127 buildings in the area between *Costa San Giorgio* and *Costa dei Magnoli* in Florence (shown in Fig. 7) with a dimension of 447×444 pixels, the alignment took 17.573 s. On the other hand, to get the alignment for all buildings in the Florence downtown (around 20,000 buildings), an orthomap of dimension 4518×5430 pixels was required. In this case the process completed in 594.396 s – note that we measured an equal time using the same orthomap and limiting the alignment to 100 buildings. To cover a whole city, bigger orthomaps will be required, however, the alignment should take less than 10 min. On the other hand, the orthomap warping and segmentation to extract the roof texture is carried out individually for each building: our Python code, running on a PC with an Intel Core i7-8700 CPU@3.20 GHz, with 32 GB of RAM, took in average 0.006 s per

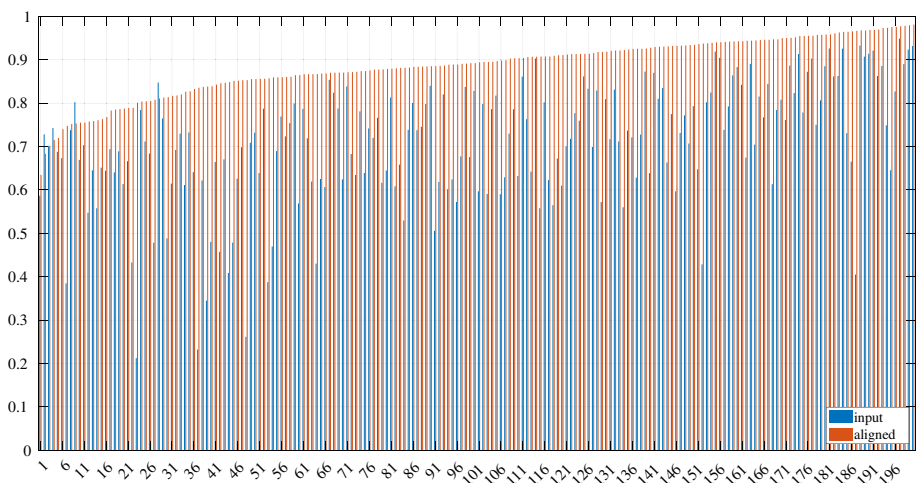


Fig. 9 A bar plot illustrating the IoU score obtained for 200 buildings between the ground-truth and the input non-aligned multi-polygons (in blue), and the aligned ones (in red). Note that the bars are ordered with respect to the aligned scores for better reading

building. Therefore, these operations for all buildings in Florence downtown require around 2.2 min. The code has been developed in Python and execution time has been estimated on the same workstation.

B. Computational times for façade texture extraction

To obtain the façade texture (process described in Section 5. B), after having selected corners, the image rectification is completed in 0.004 s in average for each façade. Just few minutes should be taken to rectify all façades for the entire city.

C. Computational times for constructing building models with 3D roofs

In this case, we are referring to the process described in Section 5. C, where LiDAR data and other information are used to reconstruct roof surfaces minimizing their number and reciprocal connections and placement with respect to facades over the building shape extruded from the plane, up to the reconstructed roof surfaces. To this end, computational times for the above mentioned 127 buildings of the area between *Costa San Giorgio* and *Costa dei Magnoli* in Florence (shown in Fig. 7), took an average of 8.696 s to complete the reconstruction of each single building. The main computational demanding effort is due to the construction of the 3D modelling (i.e., the computation of the roof and wall 3D vertexes and the triangulation of the surface meshes), and the estimation of the plane-clusters (i.e., the HDBSCAN clustering and the successive cluster expansion), respectively requiring 4.345 and 1.309 s in average. If considering that a full DT for a city like Florence will encompass more than 20,000 buildings, to obtain all 3D models about 2 days are required. However, since each building model construction is an independent process, parallel execution on multiple cores or even on different workstations/servers could be exploited to significantly reduce the total time. Please note that, the 3D building construction, as well as the roof and façade texturing are offline operations that need to be carried out sporadically.

D. Distribution performance

A performance analysis has been also carried out for the distribution of 3D model files when loading the 3D city model visualization in a Snap4City dashboard on some web browsers. We evaluated and compared different configurations for the distribution of 3D model components.

Therefore, different configurations for the 3D model distribution via a set of files have been used, as reported in the following list. For their description and meaning, please refer to the list of 3D model components reported to the end of Section 3. In this section, the focus is only on the 3D model construction, since orthomaps, PINs, traffic, etc. are less computationally intensive and may be requested on demand.

Therefore, we compare a number of 3D model configurations approaches for distribution, in particular:

- **GLB Complete:** all buildings' models and textures included in a single binary file expressed in GLB format. The binary file size resulted to be of 109 MB.
- **GLB Split:** buildings' models have been split into multiple GLB files, one for the auto-generated buildings, and additional 5 files representing HVB (i.e., "Ponte Vecchio",

“Loggia dei Lanzi”, “Santa Croce”, “San Lorenzo”, “Santa Maria Novella”. The total size of the involved files is 109 MB. One large file of about 60 MB and other smaller for the HVB.

- **Gltf Multi:** all buildings’ models and texture mapping included in a single gltf file, associated with a binary.bin file including different types of buffers, to optimize the GPU loading times. In this case, we used this file to implement the buffer for geometry, while textures have been loaded as multiple image files from different sites to parallelize download. Total size of the involved files has been of 113 MB.

Therefore, we set up a total of four configurations for 3D Model distribution. Loading times have been measured, by considering the time interval between: (i) the instant in which the request to download the 3D City model is triggered on dashboard, and (ii) the instant in which the last downloading file (among the ones which are part of the 3D city model) is completed. To this purpose, the web browser development tools have been exploited (on Google Chrome v103.0.5060.134) on a computer with an Intel core i7-1165G7@ 2.80 GHz, 32 GB of RAM and a NVIDIA GeForce MX450 graphic card. For each configuration, multiple measures have been performed, and average values have been calculated as results. Moreover, we repeated the same measures for three different network connection bandwidths in download: 135 Mb/s, 65 Mb/s and 7 Mb/s. Results are shown in Table 2. From these results, we can notice that the **GLB Complete** configuration achieves the lowest loading times for the first access to the 3D model, a part for the case in which the bandwidth is very low. When browser cache is forced to be used the general performances are improved. The distribution approach controls some of the cache parameters to accelerate the successive accesses after the first.

7 Conclusions

In this paper, a solution for modelling and construction 3D representation for Digital Twin Smart City has been presented, providing photorealistic texture and integration into Snap4City Smart City framework. Our main contributions on the paper have been on (i) the production process to pass from raw data to 3D Digital Twin elements, (ii) the integration of a complex 3D model for DT representation and its distribution approach to be shown on any browser, (iii) the satisfaction of a larger number of requirements which are needed to actually use the DT, and on (iv) the performance assessment for producing the 3D model and its distribution. The proposed solution follows a deep learning approach based on U-Net to detect the rooftops from aerial images and align them with the 3D map buildings, which are obtained by extrusion from GeoJSON data. The solution is implemented in the open-source Snap4City platform as a multi-layer 3D map, which can be used by users as

Table 2 Comparison of loading times for the distribution of different 3D Model configurations and bandwidths

3D model configurations	Bandwidth configurations		
	135 Mb/s	65 Mb/s	7 Mb/s
GLB Complete	6.92 s	13.41 s	132.00 s
GLB Split	8.23 s	14.81 s	129.60 s
Gltf Multi	13.58 s	19.05 s	143.40 s

Number in bold remark the best values

a widget on dashboards to visualize a full 3D city environment and a large variety of data, including IoT devices (city sensors and actuators, as well as private devices), POI, heat-maps, geometries and polylines related to cycling paths, bus routes, traffic flow etc. In other words, users have the possibility to pick on map single city elements and device markers and inspect their data and attributes. In this way, the proposed solution aims at providing an easy and smart navigation of the global digital twin of the city and its related data. The paper has demonstrated that it is possible and computationally viable to create 3D representation of cities with moderated effort, thus confirming the validity of our approach. Also, its distribution is possible, which makes viable the creation of sophisticated realistic 3D scenario for city analysis, strategies and what-if analysis for decision makers. The developed code of the open source Snap4City is available on GitHub as linked from <https://www.snap4city.org/>, which is also distributed in Appliance Virtual Machine, as well as in Docker based configurations (see also <https://digitaltwin.snap4city.org>).

Future direction of this research activity includes: modelling of city details such are trees, semaphore, sidewalks, etc. to make the scene more realistic when it is observed and navigated at the maximum zoom; further improvement of performance since the addition of more 3D structures will requires a new round of optimization (for entity modelling, management and distribution algorithms); the modelling of additional complex data such as scenarios, routing, etc.

Acknowledgements Authors would like to thank the HeritData Interreg project. Snap4City (<https://www.snap4city.org>) is an open technology and research by DISIT Lab, University of Florence, Italy.

Funding Open access funding provided by Università degli Studi di Firenze within the CRUI-CARE Agreement.

Data availability Data supporting the findings of this study are available from <https://www.snap4city.org/> and restrictions apply to the availability of the original source data, which were used under license for the current study, and thus they are not publicly available. On the other hand, resulting data of the produced 3D models are free on distribution from the Dashboards.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Chaturvedi K, Matheus A, Nguyen SH, Kolbe TH (2019) Securing spatial data infrastructures for distributed Smart City applications and services. *Future Gener Comput Syst* 101:723–736
2. Mylonas G, Kalogers A, Kkalogeras G, Anagnostopoulos C, Alexakos C, Muñoz L (2021) Digital twins from smart manufacturing to smart cities: a survey. *IEEE Access* 9:143222–143249. <https://doi.org/10.1109/ACCESS.2021.3120843>
3. Lafioune N, St-Jacque M (2020) Towards the creation of a searchable 3D smart city model. *Innov Manag Rev* 17(3):285–305


4. Deng T, Zhang K, Shen Z-J (2021) A systematic review of a digital twin city: a new pattern of urban governance toward smart cities. *J Manag Sci Eng* 6(2):125–134. <https://doi.org/10.1016/j.jmse.2021.03.003>
5. Rasheed A, San O, Kvamsdal T (2020) Digital twin: values, challenges and enablers from a modeling perspective. *IEEE Access* 8:21980–22012
6. Caprari G (2022) Digital twin for urban planning in the green deal era: a state of the art and future perspectives. *Sustainability* 14(10). <https://doi.org/10.3390/su14106263>
7. Shahat E, Hyun CT, Yeom C (2021) City digital twin potentials: a review and research agenda. *MDPI* 13(6):3386
8. Gröger G, Plümer L (2012) CityGML interoperable semantic 3D city models. *ISPRS J Photogramm Remote Sens*:16–21
9. Jovanovic D, Milovanov S, Ruskovski I, Govedarica M, Sladic D, Radulovic A, Pajic V (2020) Building virtual 3D city model for smart cities applications: a case study on campus area of the University of Novi Sad. *ISPRS Int J Geo-Inf*:16–21
10. Helsinki 3D city model. Available online: <https://kartta.hel.fi/3d/#/>. Accessed 10-06-2022
11. Rotterdam 3D. Available online: <https://www.3drotterdam.nl>. Accessed 10-06-2022
12. ETH Zurich VarCity project. Available online: <http://www.varcity.ethz.ch/>. Accessed 10-06-2022
13. Berlin 3D, 3dcitydb. Available online: https://www.3dcitydb.org/3dcitydb-web-map/1.7/3dwebclient/index.html?title=Berlin_Demo&batchSize=1&latitude=52.517479728958044&longitude=13.411141287558161&height=534.3099172951087&heading=345.2992773976952&pitch=-44.26228062802528&roll=359.933888621294&layer_0=url%3Dhttps%253A%252F%252Fwww.3dcitydb.org%252F3dcitydb%252Ffileadmin%252Fmydata%252FBerlin_Demo%252FBerlin_Buildings_rgbTexture_ScaleFactor_0.3%252FBerlin_Buildings_rgbTexture_collada_MasterJSON.json%26name%3DBerlin_Buildings_rgbTexture%26active%3Dtrue%26spreadsheetUrl%3Dhttps%253A%252F%252Fwww.google.com%252Ffusiontables%252FDataSource%253Fdocid%253D19cuelDgIHMqRQyBwLEztMLEzGzP831BWFtKQA3B%2526pli%253D1%2523rows%253Aid%253D1%26cityobjectsJsonUrl%3D%26minLodPixels%3D100%26maxLodPixels%3D1.7976931348623157e%252B308%26maxSizeOfCachedTiles%3D200%26maxCountOfVisibleTiles%3D200. Accessed 10-06-2022
14. Stockholm Opencities Planner. Available online: <https://eu.opencitiesplanner.bentley.com/stockholm/stockholmvoxer>. Accessed 10-06-2022
15. Bonczak B, Kontokosta CE (2019) Large-scale parameterization of 3D building morphology in complex urban landscapes using aerial LiDAR and city administrative data. *Comput Environ Urban Syst* 73:126–142
16. Fissore E, Pirotti, F (2019) DSM and DTM for extracting 3D building models: advantages and limitations. *Int Arch Photogramm Remote Sens Spatial Inf Sci XLII-2/W13*:1539–1544. <https://doi.org/10.5194/isprs-archives-XLII-2-W13-1539-2019>
17. Xue F, Lu W, Chen Z, Webster CJ (2020) From LiDAR point cloud towards digital twin city: clustering city objects based on Gestalt principles. *ISPRS J Photogramm Remote Sens* 167:418–431
18. Gui S, Qin R (2021) Automated LoD-2 model reconstruction from very-high-resolution satellite-derived digital surface model and orthophoto. *ISPRS J Photogramm Remote Sens* 181:1–19. <https://doi.org/10.1016/j.isprsjprs.2021.08.025>. (ISSN 0924-2716)
19. Building Reconstruction CityGML builder. <https://vc.systems/en/products/building-reconstruction/>. Accessed 10-06-2022
20. Awrangjeb M, Gilani SAN, Siddiqui FU (2018) An effective data-driven method for 3-D building roof reconstruction and robust change detection. *Remote Sens* 10:1512. <https://doi.org/10.3390/rs10101511>
21. Wang Y, Zorzi S, Bittner K (2021) Machine-learned 3D building vectorization from satellite imagery. In: Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition (CVPR) workshops
22. Thompson JA, Bell JC, Butler CA (2001) Digital elevation model resolution: effects on terrain attribute calculation and quantitative soil-landscape modeling. *Geoderma* 100:67–89
23. Ye Y, Shan J, Bruzzone L, Shen L (2017) Robust registration of multimodal remote sensing images based on structural similarity. *IEEE Trans Geosci Remote Sens* 55:2941–2958
24. Izadi M, Saeedi P (2010) Automatic building detection in aerial images using a hierarchical feature based image segmentation. In: 2010 20th International Conference on Pattern Recognition
25. Mountrakis G, Im J, Ogole C (2011) Support vector machines in remote sensing: a review. *ISPRS J Photogramm Remote Sens* 66:247–259
26. Belgiu M, Drăguț L (2016) Random Forest in remote sensing: a review of applications and future directions. *ISPRS J Photogramm Remote Sens* 114:24–31
27. Baluyan H, Joshi B, Hinai A, Woon W (2013) Novel approach for rooftop detection using support vector machine. *ISRN Mach Vis* 2013:11

28. Bosch M, Kurtz Z, Hagstrom S, Brown M (2016) A multiple view stereo benchmark for satellite imagery. In: 2016 IEEE Applied Imagery Pattern Recognition Workshop (AIPR)
29. Ma L, Liu Y, Zhang X, Ye Y, Yin G, Johnson BA (2019) Deep learning in remote sensing applications: a meta-analysis and review. *ISPRS J Photogramm Remote Sens* 152:166–177
30. Chen M, Li J (2019) Deep convolutional neural network application on rooftop detection for aerial image. *ArXiv*, vol. abs/1910.13509
31. He K, Gkioxari G, Dollar P, Girshick R (2017) Mask R-CNN. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*
32. Castello R, Walch A, Attias R, Cadei R, Jiang S, Scartezzini J-L (2021) Quantification of the suitable rooftop area for solar panel installation from overhead imagery using convolutional neural networks. *J Phys Conf Ser* 2042:012002
33. Girard N, Charpiat G, Tarabalka Y (2018) Aligning and updating cadaster maps with aerial images by multi-task, multi-resolution deep learning. In: *ACCV*
34. Ronneberger O, Fischer P, Brox T (2015) U-Net: convolutional networks for biomedical image segmentation. In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Cham
35. Han Q, Nesi P, Pantaleo G, Paoli I (2020) Smart City dashboards: design, development and evaluation. In: *Proceedings of the IEEE ICHMS 2020, International Conference on Human Machine Systems*. <http://ichms.dimes.unical.it/>. Accessed 10-06-2022
36. Garau C, Nesi P, Paoli I, Paolucci M, Zamperlin P (2020) A big data platform for smart and sustainable cities: environmental monitoring case studies in Europe. In: *Proceedings of International Conference on Computational Science and its Applications. ICCSA 2020, Cagliari*. <http://www.iccsa.org/> https://link.springer.com/chapter/10.1007%2F978-3-030-58820-5_30 . Accessed 10-06-2022
37. Adreani L, Colombo C, Fanfani M, Nesi P, Pantaleo G, Pisanu R (2022) Digital twin framework for Smart City solutions. *DMSVIVA 2022, The 28th International DMS Conference on Visualization and Visual Languages, KSIR Virtual Conference Center, Pittsburgh*. <http://ksiresearch.org/seke/dmsviva22.html>. Accessed 10-06-2022
38. Bilotta S, Collini E, Nesi P, Pantaleo G (2022) Short-Term prediction of city vehicle flow via convolutional deep learning. *IEEE Access*. <https://doi.org/10.1109/ACCESS.2022.3217240>. <https://ieeexplore.ieee.org/document/9930774> ISSN: 2169–3536
39. Collini E, Ipsaro Palesi LA, Nesi P, Pantaleo G, Nocentini N, Rosi A (2022) Predicting and understanding landslide events with explainable AI. *IEEE Access*. <https://doi.org/10.1109/ACCESS.2022.3158328>. <https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9732490>. Accessed 10-06-2022
40. Bellini E, Bellini P, Cenni D, Nesi P, Pantaleo G, Paoli I, Paolucci M (2021) An IoE and Big Multimedia Data approach for Urban Transport System resilience management in Smart City. *Sensors, MDPI*. <https://www.mdpi.com/1424-8220/21/2/435/pdf>
41. Badii C, Bellini P, Difino A, Nesi P (2019) Sii-mobility: an IOT/IOE architecture to enhance smart city services of mobility and transportation. *Sensors, MDPI*. <https://doi.org/10.3390/s19010001>. <https://www.mdpi.com/1424-8220/19/1/1/pdf>. Accessed 10-06-2022
42. Zampieri A, Charpiat G, Tarabalka Y (2018) Coarse to fine non-rigid registration: a chain of scale-specific neural networks for multimodal image alignment with application to remote sensing. *ArXiv*, vol. abs/1802.09816
43. Baldi G, Colombo C, Del Bimbo A (1999) A compact and retrieval-oriented video representation using mosaics. In: *Proceedings 3rd International Conference on Visual Information Systems VISual99*. Springer LNCS 1999, Amsterdam, p 171–178
44. Tomasi C, Manduchi R (1998) Bilateral filtering for gray and color images (PDF). *Sixth International Conference on Computer Vision*. Bombay, p 839–846. <https://doi.org/10.1109/ICCV.1998.710815>
45. Pal NR, Pal SK (1993) A review on image segmentation techniques. *Pattern Recogn* 26(9):1277–1278. [https://doi.org/10.1016/0031-3203\(93\)90135-J](https://doi.org/10.1016/0031-3203(93)90135-J)
46. Fischler MA, Bolles RC (1981) “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography” (PDF). *Comm ACM* 24(6):381–395. <https://doi.org/10.1145/358669.358692>
47. Toldo R, Fusiello A (2008) Robust multiple structures estimation with J-linkage. In: *Forsyth D, Torr P, Zisserman A (eds) Computer Vision – ECCV 2008*. ECCV 2008. *Lecture Notes in Computer Science*, vol 5302. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-88682-2_41
48. Campello RJGB, Moulavi D, Sander J (2013) Density-based clustering based on hierarchical density estimates. In: *Pei J, Tseng VS, Cao L, Motoda H, Xu G (eds) Advances in Knowledge Discovery and Data Mining. PAKDD 2013*. *Lecture Notes in Computer Science*, vol 7819. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-37456-2_14
49. Bellini P, Bugli F, Nesi P, Pantaleo G, Paolucci M, Zaza I (2019) Data flow management and visual analytic for big data Smart City/IOT. *19th IEEE Int. Conf. on Scalable Computing and Communication*. *IEEE*

SCALCOM, Leicester. <https://www.slideshare.net/paolonesi/data-flow-management-and-visual-analytic-for-big-data-smart-cityiot>. Accessed 10-06-2022

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Authors and Affiliations

L. Adreani^{1,2} · P. Bellini^{1,2} · C. Colombo^{2,3} · M. Fanfani^{1,2,3} · P. Nesi^{1,2}  · G. Pantaleo^{1,2} · R. Pisanu^{2,3}

✉ P. Nesi
Paolo.Nesi@unifi.it
<https://www.disit.org>
<https://www.snap4city.org>

L. Adreani
Lorenzo.Adreani@unifi.it
<https://www.disit.org>
<https://www.snap4city.org>

P. Bellini
Pierfrancesco.Bellini@unifi.it
<https://www.disit.org>
<https://www.snap4city.org>

C. Colombo
Carlo.Colombo@unifi.it
<http://cvg.dsi.unifi.it/cvg/>

M. Fanfani
Marco.Fanfani@unifi.it
<https://www.disit.org>
<https://www.snap4city.org>
<http://cvg.dsi.unifi.it/cvg/>

G. Pantaleo
Gianni.Pantaleo@unifi.it
<https://www.disit.org>
<https://www.snap4city.org>

R. Pisanu
Riccerdo.Pisanu@unifi.it
<http://cvg.dsi.unifi.it/cvg/>

¹ DISIT Lab, Florence, Italy

² University of Florence, Florence, Italy

³ Computational Vision Group, Florence, Italy