Open Data for large-scale geospecific 3D Simulation for Security Applications - A Case Study

Dirk Frommholz

DLR Institute of Space Research, Berlin, Germany - dirk.frommholz@dlr.de

Keywords: Open Data, Building Model, Texture Mapping, 3D Simulation, Serious Gaming, VBS4.

Abstract

This case study details the integration of official large-scale open 2D and 3D geospatial data of the city of Berlin, Germany, into the Virtual Battlespace 4 (VBS4) simulator for security applications. Realistic scenery with elements specific to the target area is obtained from a digital terrain model, true-ortho mosaic, and high-resolution land use/land cover layer rasterized from OpenStreetMap vector primitives. For the central Mitte borough with its government institutions and foreign embassies, almost 20000 buildings are prepared from textured CityGML data in an automatic multi-stage process. This process involves pre-wrapping the texture images, which are referenced by the semantic 3D models using non-canonical coordinates, and the rapid creation of compact atlases to reduce the bitmap count by three orders of magnitude. To ensure that the building meshes blend seamlessly into the terrain, vertical adjustment methods are discussed, and ground extrusion is implemented to approach the model’s base surfaces from below. Data import into VBS4 happens through its Geo interface for the terrain, ortho, and land cover, while the buildings are compiled into an add-on with a custom workflow that involves reprojection, collision component setup, and damage behavior configuration. During interactive convoy training in the virtual environment, a high recognition value compared to the real landscape could be attested visually. Simulation exhibited acceptable frame rates, but required considerable computing resources.

1. Introduction

In recent years, comprehensive geospatial data sets from places around the world have been published under permissive licenses by administrations, companies, and collaborative projects. These include 2D products like digital vector maps, elevation models, orthophotos, and thematic layers such as land use/land cover (LULC) segmentations. A prominent example is the OpenStreetMap (OSM) database (OpenStreetMap, 2025), which has served as a global map source for real-time navigation, nautical charts, humanitarian aid coordination, and gaming, among other applications (Mooney and Minghini, 2017). Beyond two dimensions, point clouds, polygonal meshes and semantic building models with and without textures are increasingly being produced in standard exchange formats (Lei et al., 2023). The 3D data sets mainly cover city areas and have been used for visualization and simulation to address energy, mobility, environmental and security topics (Willenborg et al., 2018).

When geospatial products are released under permissive licenses by their creators, users are entitled to freely access, distribute, exploit and modify this ”open data” for their own purposes. Nevertheless, terms and conditions like crediting the original source may still apply (Khayyat, 2015). The socioeconomic impact of open geospatial data has been widely discussed and also quantified (Hansen and Schrøder, 2019). Free availability, in terms of cost and the absence of usage restrictions, promotes new applications and allows larger areas to be covered. An example is the integration of geographically located 2D and 3D products into modern game engines, i.e., software frameworks that drive entertaining or serious computer games (Vohera et al., 2021). The rationale is to support the rapid creation of extended virtual environments, especially of urban districts or entire cities, with authentically looking, physically sound and precisely positioned landscape elements. Virtual worlds enable simulation and interactive exploration in real-time from first-person or third-person perspectives. This is specifically relevant to public safety organizations who wish to play through emergency scenarios and perform training tasks as close to reality as possible without actually deploying. Also, an accurate digital representation of a scene will facilitate analyses and predictions on the vulnerability and resilience of urban populations.

Regardless of the actual application, the geospatial data capabilities offered by game engines currently lack support for widespread geographic data formats. This is because their structure and complexity conflict with hardware-accelerated visualization techniques and real-time object interaction, especially for large scenes. Thus, a preprocessing workflow must be applied to meet the input requirements of the targeted software system depending on the types of available data, encodings, coordinate reference frames, and other factors.

2. Related Work and Objectives

There are few technical publications on geospatial data preparation for current 3D gaming frameworks and the achieved run time performance. In (Höhl, 2020), two pipelines hinged on commercial and open-source software are discussed that process official survey data to be subsequently used with the cost-free Unreal and Unity engines for augmented/virtual/mixed reality (AR/VR/MR) applications. Both workflows ingest digital terrain text representations, raster orthophotos, CityGML 3D building models (Open Geospatial Consortium, 2012), and path information stored as ESRI shape files (ESRI, 1998). For coordinate system homogenization and format conversion, the commercial toolchain is built around proprietary geographic information systems (GIS) and data translation software, i.e., ESRI ArcGIS with CityMapper, Global Mapper, and FME. The open source variant mainly utilizes the QGIS software to obtain results that can be consumed by the respective engine. Run time of the workflows, feasible scene sizes, and performance on actual AR/VR hardware remain unclear as no evaluation was carried out. In (Würstle et al., 2022), the commercial toolchain is adapted to produce outputs in the open 3D Tiles and I3S streaming formats that provide a level of detail (LOD) mechanism

and in principle allow large scenes to be smoothly displayed. Additional plug-ins are deployed for the once more targeted Unity and Unreal engines to import geospatial data. However, the plug-ins also enable the authors to integrate remote sources of global coverage. First-person view showcases in augmented reality, city development, and mobility are illustrated, but only basic statements are made on the simulation experience. A planetary-scale open visualization platform for digital twin cities, i.e., virtual models of cities simulating their real-world systems, is proposed in (Lee et al., 2020). The platform built on the Unity game engine selectively streams textured terrain and detailed 3D building geometries from the Korean VWorld map service. The data populates surface sectors of the hierarchically subdivided globe depending on the current viewpoint. Although no interactive showcase is demonstrated, it is stated that the described setup can achieve frame rates that exceed the temporal resolution limit of the human visual system. For large urban areas, (MLIT, 2025) describes multiple workflows that process official data sets of the Japanese PLATEAU open data initiative into formats supported by the Unreal/Unity game engines, the Minecraft computer game, and GIS. Specific add-ons have been developed that transform comprehensive CityGML descriptions of urban areas in the country provided either as files or via web services into textured single- or multiresolution geometries. In the process, textures get consolidated into atlases, and different LODs for the data layers can be chosen. The manual has render samples, but omits any performance measures in relation to the scene extent.

This case study exemplifies the preparation of large-scale open data into a collection of related geospatial products, termed ”terrain database”, for the Virtual Battlespace 4 (VBS4) simulator originally developed for defense forces (Bohemia Interactive Simulations, 2022). However, the software also suits civil security applications, and the outlined processing steps likewise will apply to other geo-aware serious gaming engines. The main objective is to demonstrate how publicly available data sets can be transformed into ”geospecific” scenery to give virtual simulation environments a sufficiently realistic, but not necessarily perfect, look and feel that is characteristic for the target area. The anticipated increased recognition value will be especially relevant to e.g. *spatially extensive* three-dimensional convoy route planning in the advance of rescue and evacuation missions from the driver’s perspective or the interactive training of security staff prior to official visits. It is further shown that the use of existing geographic data can speed up scene creation compared to traditional workflows and what the simulation performance is like in VBS4 for considerable 2D and 3D inputs.

3. Data Set Preparation

The open geographic data set of Berlin in Germany covering the whole city of 891 km² was selected for the case study (Senatskanzlei Berlin, 2025). Among other remote sensing products, the collection includes the digital terrain model (DTM), RGBI true-ortho mosaic (TOM), and semantic 3D building models with textures, all of which are relevant to simulation. In addition, digital vector maps from the OpenStreetMap project were obtained from (Geofabrik GmbH, 2025) for the target mainly to derive a land use/land cover (LULC) thematic layer, with gaps filled in using ESA WorldCover data (Zanaga et al., 2022). The full extent of the city area was kept for the DTM and LULC products because of lower resolution requirements during simulation. For the TOM and building models, a subset of the Mitte borough of about 40 km² got processed according to VBS4 conventions. This part of the city is home to numerous government institutions and foreign embassies.

3.1 Raster Image Layers

The digital terrain model forms the ground surface of the virtual 3D scenery on which any other elements are placed. Its 297 graylevel GeoTIFF raster tiles (Open Geospatial Consortium, 2019) created from LiDAR data in 2021 are part of the official topographic-cartographic information system of Germany named ATKIS. Each tile covers an area of 2 x 2 km at a ground sampling distance (GSD) of 1.0 m and is geographically located in zone 33U of the Universal Transverse Mercator (UTM) coordinate system. The ”bare ground” elevation is encoded according to the 2016 German height reference system (DHHN2016) using single-precision floating-point numbers.

To obtain a DTM compatible with VBS4, the tiles are merged into a single raster image. The Berlin area is masked with a polygonal shape from the OSM project. This flags the surrounding pixels as invalid using an elevation that does not occur within the city limits. Reprojection of the masked DTM into the 3D WGS84 geographic reference frame is performed with the 2008 Earth Gravitation Model (EGM2008) as an approximation of the DHHN (Gruber, 2019). During reprojection, simultaneous resampling to a GSD of 2.5 m, i.e., a resolution close to the VBS4 render limit, is applied. The resulting bitmap of 23209 x 11440 pixels further is smoothed with a bilateral filter to selectively eliminate small elevation irregularities that otherwise might cause unexpected accidents during interactive vehicle operation. To stay well below the 2/4 GiB size limits and maintain compatibility with legacy image processing tools, the image is split into a set of two tiles of no more than 16384 pixels per dimension and stored as GeoTIFF bitmaps. Except for the filter step that involves self-developed software, DTM preparation is carried out as a batch job running open source tools from GDAL (GDAL/OGR contributors, 2025).

The true-ortho mosaic stitched together from aerial images projected onto a digital surface model (DSM) colors the terrain and procedurally grown vegetation during simulation. The original data set from 2024 consists of 283 JPEG 2000 tiles (Santa-Cruz et al., 2002) covering the red, green, blue and near infrared spectral bands (RGBI). Each tile displays a surface area of 2 x 2 km at a GSD of 0.2 m and hence has 10000 x 10000 pixels with four components of the byte type. Processing the TOM into a VBS4 background image resembles the procedure used for the DTM. However, only the first three sub-bands are preserved, the filter stage is omitted, and the tiles not containing the Mitte borough are discarded. The result is four 16384 x 16384 RGB images, resampled to a GSD of 0.5 m.

The land use/land cover layer defines the appearance and physical properties of the surface materials and vegetation cover in VBS4. When the player walks or drives through the virtual environment, the LULC class at the current location affects the speed of movement and triggers distinctive sound effects. Surface information is derived from the binary OpenStreetMap database extract for the entire Berlin area from 2025 and vectorized ESA WorldCover data from 2021. The semantically annotated geometric primitives are automatically rasterized into four 8-bit graylevel bitmaps at a GSD of 1.0 m preserving the true width of linear elements (Frommholz, 2025). As the extents for the bitmaps are given as pairs of WGS84 polar coordinates due to OSM conventions, their dimensions are either 25746 x 12873

or 25793 x 12897 pixels, depending on the latitude of the upper left image corner. Intensities are assigned from 39 predefined VBS4 land use/land cover categories. Figure 1 depicts the city center in the produced DTM, TOM and LULC raster layers.

3.2 Building Models

Building models for the Mitte borough come from a compressed archive containing a monolithic CityGML file that conforms to version 2.0 of the standard. The file describes 19736 semantically annotated structures in boundary representation at level of detail 2 (LOD2), i.e., including differentiated roof shapes, but without openings. The individual 3D meshes created in 2014 are located in the same UTM coordinate system as the DTM and use largely similar DHHN92 heights. Colors are provided by 366427 diffuse 24-bit RGB texture images in JPEG and PNG formats (Santa-Cruz et al., 2002) linked to the building polygons via non-canonical 2D texture coordinates. The spatial resolution of the façade bitmaps appears to be sub-decimeter assuming typical dimensions for window openings and considering the number of pixels used to portray them.

Before the building data can be added to a VBS4 scene, the archive needs to be decompressed. The model then is read once and rewritten to detect parser issues. Similarly, the textures are checked for read errors. Radiometric equalization and geometric normalization are performed to balance colors and map the texture coordinate tuples into the standard range. The huge quantity of images must further be consolidated, and the vertical positions of the buildings have to be adjusted to the terrain to ensure a seamless transition to the surface. These tasks are automatically executed using a shell script that invokes a custom modular 3D mesh processing tool.

3.2.1 Texture Normalization During rendering, the two-dimensional texture coordinates assigned to the vertices of each polygon of the Berlin building models determine the part of the referenced texture bitmap that is to be ”glued” on the respective face in order to color its body. Their canonical range is $(u, v) \in ([0; 1], [0; 1])$ relative to the image dimensions. Coordinates outside this interval are interpreted according to the wrap modes specified in the CityGML file. These modes are always set to ”wrap” for the Mitte building subset, which means that the textures are to be repeated.

Out-of-range texture coordinates will produce render artifacts if the game engine does not obey the original wrap mode. Also, when multiple texture bitmaps are to be combined into an atlas bitmap, adjacent texture patches might be overwritten, causing faces to partially appear in wrong colors. To circumvent these issues, the texture bitmaps referenced by the building model polygons of the Mitte data set are pre-wrapped. Their texture coordinates get adjusted to the dimensions of the newly created images effectively normalizing them to the $[0; 1]$ interval.

Texture normalization is carried out in three steps. The analysis step retrieves the dimensions of all images referenced by the CityGML file and computes the bounding box around all 2D coordinate tuples that belong to a specific texture. The acquired information is kept in a hash table for fast retrieval using the texture storage locations as keys. The render step creates new texture bitmaps whose width and height are calculated as the product of the dimensions of the original texture and the bounding box. For each pixel of a new bitmap, pixels from the original image are read, and when the original frame is left, the source pixel position is reset repeating the original texture. The render loop processes multiple inputs in parallel due to the large number of source textures. As fractional read positions occur for the source pixels, subpixel interpolation with bicubic splines is performed. The interpolator determines a weighted color value from the 5 x 5 neighborhood suppressing artifacts at the image boundaries through pixel mirroring. Brightness differences between the rendered texture images are equalized with a gamma adjustment so that the mean intensity in the image corresponds to 50% of the dynamic range. The rendered texture images are uniformly saved in PNG format with lossless compression. The third step traverses the building geometries again to shift the original texture coordinates by the bounding box minimum and divide them by the bounding box size. This adapts the coordinate tuples to the output texture dimensions and guarantees canonical values. Figure 2 illustrates the pre-wrapping process.

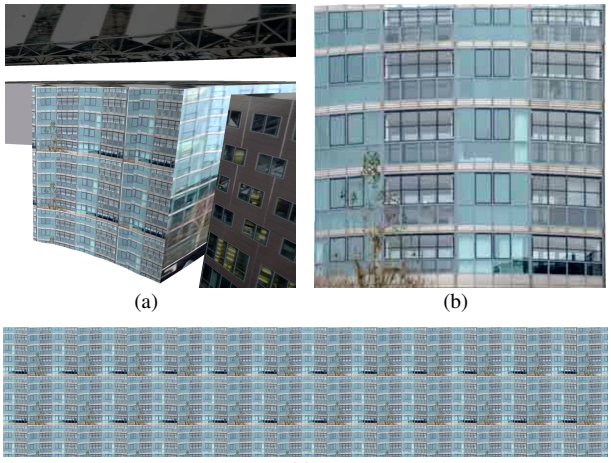
3.2.2 Texture Consolidation Texture normalization leaves the quantity of bitmaps unchanged. To reduce loading times for the colored building models in the 3D engine, the individual images must be combined into a smaller number of texture atlases. Atlas generation mirrors (Frommholz et al., 2017), whose sequential patch placement algorithm for polygonal inputs has been optimized for rectangular bitmaps, enhanced with deduplication, and added support for texture spacing.

Texture consolidation starts with dividing the monolithic CityGML input into sets of 150 buildings to be able to preview partial results of the lengthy process. For each set, the referenced images are read to be sequentially pasted into the current atlas bitmap. Calculation of the next free spot and the check for overlaps with previously inserted texture patches are accomplished using a block availability map (BAM). The BAM holds a list of available atlas scanline segments sorted by their horizontal offsets. Thus, in the original implementation, for polygonal textures to be added, the BAM is searched after the first free scanline segment. For all positions within the segment, attempts are made to occupy its pixels and the pixels of subsequent potentially unconnected scanline segments horizontally and vertically depending on the shape and dimensions of the source texture. This generally requires numerous intersection calculations involving irregularly formed bodies. However, for the rectangular bitmaps referenced by the Berlin 3D models, the occupancy checks are greatly accelerated, since only the width of the texture has to be compared with the length of the current scanline segment at the first segment position. Figure 3 illustrates the occupation management with the block availability map.

The placement algorithm arranges the source textures compactly within the atlas boundaries. However, for state-of-the-art game engines and the graphics accelerators they rely on, gaps between neighboring patches may be desirable for high-quality 3D rendering due to the anti-aliasing algorithms implemented in hardware. These algorithms apply low-pass filters for subpixel interpolation when the atlas bitmaps are being read by the fragment shaders using fractional texture coordinates, and for mipmapping (Akenine-Möller et al., 2008). As the filters average colors in the vicinity of the center pixel, adjacent textures or unoccupied atlas regions will be inevitably included in the calculation near the boundaries of the currently accessed patch. This may create distracting fringes at the edges of the building polygons during visualization as shown in figure 4. Edge artifacts from texture sampling will be effectively suppressed if a virtual border is added to all patches to be inserted into the atlases prior to placement and if the border area is later filled

(a) (b) (c)

Figure 1. VBS4 raster image layers for the city center of Berlin (a) digital terrain model (DTM, intensities normalized for display, height difference $\Delta h \leq 7m$), (b) true-ortho mosaic (TOM), (c) land use/land cover (LULC) map.



(a) (b)

(c)

Figure 2. Texture normalization (a) 3D building model façade with texture coordinates $(u, v) \in ([-6.49; 5.62], [-1.65; 1])$, (b) original texture bitmap, (c) pre-wrapped texture bitmap.

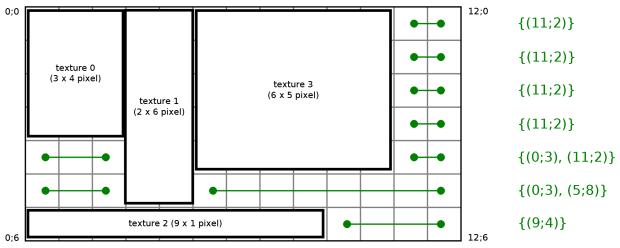
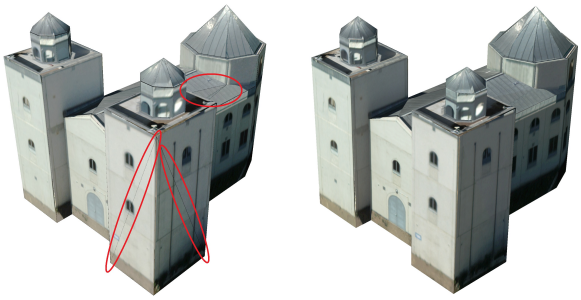


Figure 3. Texture atlas occupation management with the BAM storing free scanline segments as ordered (offset;length) tuples.

from surrounding pixels. Hence, for the Mitte buildings, the minimum patch distance was empirically set to two pixels, resulting in a spacing of four pixels between nearby textures.

Once a sufficiently large free spot has been found for a source texture, its position inside the atlas and its size are stored. The placement information is further added to a lookup table and instantly recalled when the same image is referenced multiple times by the 3D building models. In fact, for the Berlin data set, surfaces often consist of topologically closed sets of non-intersecting polygons, or, multipolygons, that are colored from different parts of a single texture. Thus, the implemented deduplication mechanism eliminates redundancies and reduces the dimensions and number of the atlases created. If no space is available in the current texture atlas, its bitmap and the associated BAM are expanded horizontally and vertically so that their dimensions correspond to the next power of two. This supports

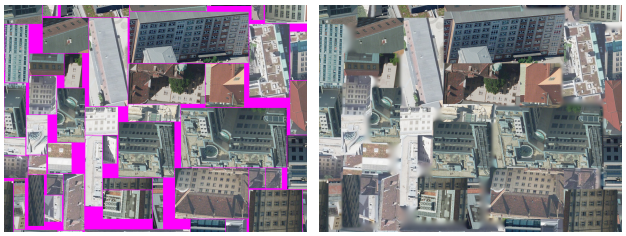


(a) (b)

Figure 4. Texture sampling artifacts during visualization (a) densely packed atlas causing color fringing, (b) smooth transitions with loosely packed texture atlas and interpolation.

the creation of the mipmap pyramid. The maximum atlas size is 8192 x 8192 pixels, which most graphics cards can still handle. When the width or height limits are reached, a new atlas bitmap is set up. Its size matches the first texture to be inserted, rounded up to the nearest power of two. If either dimension of the initial texture exceeds 8192 pixels, it stays a standalone atlas.

After the placement has been calculated, the atlas bitmaps are rendered by copying the source texture pixels to their new positions taking into account the optionally added virtual image border. Unoccupied pixels are interpolated from the inserted texture patches using semi-global inverse distance weighting (IDW) (Frommholz, 2022) to accommodate the anti-aliasing filters applied during visualization. To ensure that only void pixels are altered, pure magenta with the RGB triplet $(255; 0; 255)$ is chosen as the background color for each texture atlas. This color is virtually never found on natural or man-made objects. The resulting bitmaps are saved in PNG format to reduce file sizes without losing color information, as the data will be further processed for use with VBS4. Figure 5 depicts a part of a sample texture atlas before and after interpolation.



(a) (b)

Figure 5. Final atlas bitmap (a) textures copied to their atlas positions, (b) gap interpolation with inverse distance weighting.

As the final step in texture consolidation, the texture coordinates assigned to the vertices of the building model polygons, which still refer to the original color sources, are recalculated. Their new values are taken from the stored positions inside the corresponding texture atlas and normalized to the $[0; 1]$ interval based on its dimensions. The adjusted data sets of 150 buildings are split into individual structures by the mesh processing tool and saved as CityGML files.

Altogether, the number of textures for the 3D models of Berlin-Mitte has been reduced from 366427 to 282, i.e., by a factor of nearly 1300. Meanwhile, the total pixel count has grown by about 18% from 13.34 to 15.79 gigapixels after switching to atlas bitmaps due to pre-wrapping, patch placement overhead, and the spacing added for the anti-aliasing filters.

3.2.3 Vertical Adjustment Even if the DTM and building models are well registered, discrepancies between the terrain and the bottom surfaces of the structures can occur, causing the latter to hover visibly above the ground in VBS4. Possible remedies to eliminate gaps between the terrain and building models include

- raising the terrain to the ground surfaces of the buildings,
- sinking the buildings into the terrain using vertical shifts,
- stretching the buildings into the terrain and adjusting the textures and texture coordinates accordingly, i.e.,
  - create new larger texture bitmaps for the polygons, blit the original content, and interpolate the seams,
  - shrink the content of the texture bitmaps for the polygons in-place and interpolate the seams,

as illustrated in figure 6. For the Mitte dataset, assuming that the structures are accurately positioned vertically, the terrain is raised to meet the ground surfaces of the building models. The first approach has been chosen for its runtime advantage, reduced DTM occlusions, and preservation of absolute building heights. In the implementation as part of DTM preparation, the mesh processor extracts ground surface polygons based on the semantic labels in the CityGML representation and stores their vertices as PolygonZ records in an ESRI shape file. The shape file gets burnt into the DTM with the GDAL rasterization tool in 3D mode, which also handles coordinate system conversion.

Regarding the third approach, stretching the polygons of the building models into the ground requires adjusting the textures to leave their aspect ratio unchanged. This can be done by copying the original color information into larger images with space for the interpolated pixels of the extended polygons. Alternatively, the dimensions of the original textures could be kept downsizing their content. Once the source textures have been packed into texture atlases, enlarging the patches in-place usually requires a time-consuming recalculation of the atlas bitmaps since adjacent pixels that color a different polygon may get overwritten. Downsizing avoids finding a new patch placement, but reduces the spatial texture resolution in proportion to the geometry stretch. Size changes to textures in both directions can either be made symmetrically or asymmetrically. Symmetrical resizing, for example, by central dilation, may result in larger images than necessary. Asymmetrical resizing involves the detection of the expansion or contraction direction to determine the ground-facing edges of façade polygons, for instance, based on the building surface semantics. In presence of non-canonical texture coordinates, any texture adjustments have to be incorporated into pre-wrapping.

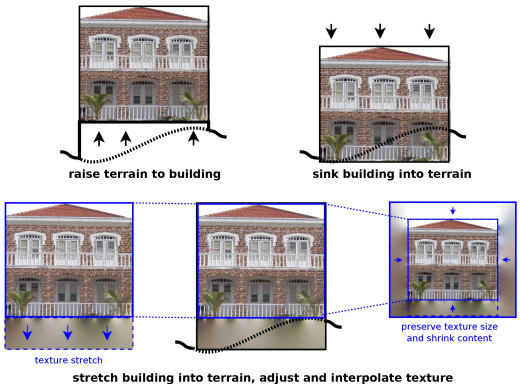


Figure 6. Ways to fix building models hovering over the terrain.

4. Integration into the Simulator

To integrate the prepared terrain database of Berlin into VBS4, a new scene is created in the software, referred to as ”battlespace,” which alludes to the main customers of the simulation environment. The battlespace origin is selected on a world map close to its true location. Afterwards, the image products are imported directly via the VBS4 Geo interface, and the default buildings are manually removed with the eraser tool at maximum diameter. The 3D building models in CityGML format are not natively supported by the simulator and have to be compiled into a custom add-on. The add-on will be set up to target the Offline Mission Editor (OME) instead of Geo mode. OME utilizes customizable text-based configuration files that enable automatic geographically consistent object placement, and it allows to interactively move and edit the individual structures later.

The procedure for the automatic conversion of textured building models from CityGML into a custom simulator add-on is shown in figure 7. It has been largely implemented as a command line application written in the C++ programming language. The vbstool calls functions from the proprietary software development kit (SDK) coming with VBS4 and therefore, like the simulator itself, runs under 64-bit Windows operating systems only.

As a prerequisite to the building conversion workflow, the add-on configuration is initialized by opening the battlespace with the integrated image products in VBS4, switching to OME, and resaving the scene. Vbstool then is invoked with the installation location of the add-on, the path to the CityGML buildings, their coordinate system, and the name of choice for the add-on as parameters. The program parses the input models, cleans them from degenerate elements like unconnected vertices or zero-area polygons, and runs ear decomposition to triangulate the faces. The triangles are reprojected into the UTM coordinate system used internally by VBS4. The UTM zone depends on the selected battlespace origin. To avoid rounding errors, the coordinates of the origin are extracted from either the Geo or OME configuration files, i.e., battlespace.json or mission.biedi, depending on the preferred design mode.

Once reprojection is complete, several levels of detail will be generated for the buildings. The display LOD is used for visualization and comprises the textured model triangles. The triangles are translated into native VBS4 data types (BISim structures, named after the VBS software maker) if OME is addressed or undergo grid convergence correction first if the buildings are to be used in Geo mode. Grid convergence correction

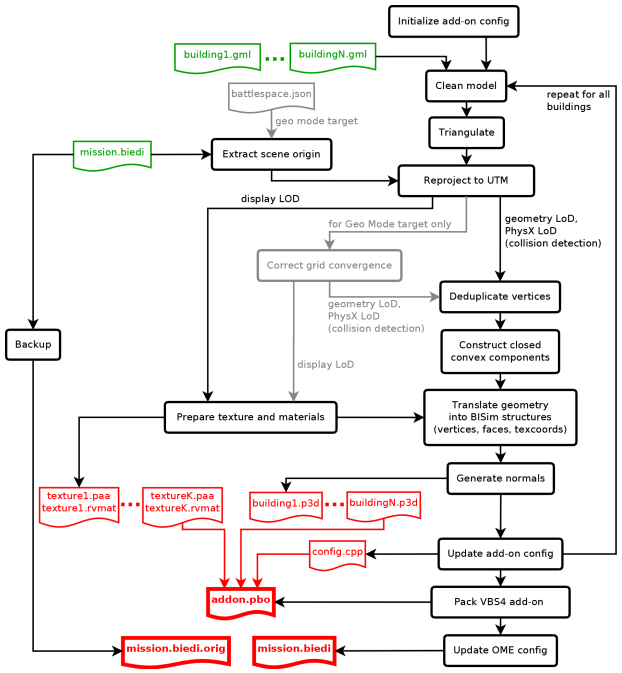
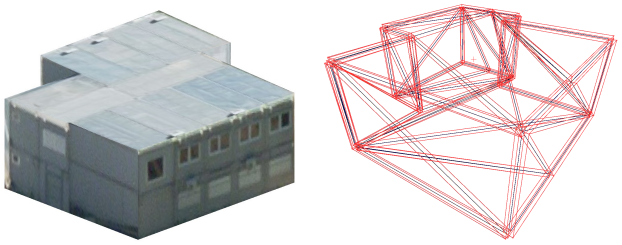


Figure 7. VBS4 building model conversion workflow.

eliminates the rotational deviation between true north on the Geo globe and grid north of the UTM map projection the buildings are in and which, presumably for performance reasons, VBS4 does not compensate during model placement. Besides the display LOD geometries, the texture bitmaps assigned to the triangulated building meshes will be tone-mapped to 8 bits per color channel if they expose a higher dynamic range. The resulting images are ultimately converted to PAA format using SDK functions. During conversion, lossy S3 Texture Compression (S3TC) is applied (Paltashev and Perminov, 2014), which reduces video memory utilization and for which decompression is implemented in graphics cards. Also, each texture is assigned a material file storing the ambient, diffuse, specular, and emissive lighting parameters as text. These settings are currently static and were chosen empirically.

For collision detection and deformation, the sophisticated display LOD for each building model is added geometry, PhysX and fire LODs. The 3D shapes of these LODs must be closed and convex to speed up intersection calculations. Currently, the collision LODs are identically derived from the reprojected triangulated building models after grid convergence correction in case of Geo targets. Unlike the display LOD, whose mesh vertices have been deliberately duplicated to enable per-face textures, coinciding 3D points are now merged. The resulting model is tested whether its shape is already closed and convex, and if so, it will be directly translated into BISim structures. If the condition is not met, to still be able to collide with the building during simulation, right trigonal prisms that fulfill the required mathematical properties are generated by offsetting each mesh triangle along its two normals as shown in figure 8. This workaround is universally applicable, but inflates the face count of the collision LODs. If the technical limits of VBS4 are exceeded in this regard, overly complex open or concave building models will end up without contact geometry.

To also define a certain level of damage resilience for structures of different size, mass values are assigned to each collision LOD. These values are estimated from the assumed average density of a building of $\rho = 250\ kg/m^3$, multiplied by half the volume of its bounding box, distributed across all vertices. After all LODs have been created for each model, vertex normal vectors will be recalculated. This ensures that the 3D objects appear properly illuminated in VBS4. The entire LOD set is saved as proprietary P3D files referencing the materials.



(a) (b)

Building models, textures - Berlin Partner für Wirtschaft und Technologie GmbH, custom permissive license

Figure 8. VBS4 LODs for concave building (a) display LOD, (b) geometry LOD with collision prisms (red) for each face (black).

Afterwards, the add-on configuration described by the config.cpp file remains to be updated. The file registers the building models with VBS4 and defines their functionality using a syntax similar to C++. At present, only the destruction behavior of the structures is set up so that a default pile of rubble will be produced on full disintegration in the simulator. However, the interaction response of wall openings like windows or doors can also be specified in config.cpp, provided that their geometries are contained in the CityGML files.

Once the add-on has been configured, all its data is packed into a single output file using SDK functions. The resulting proprietary archive is copied to the add-on installation path passed to vbstool. For the placement of buildings inside the battlespace, the software alters the OME text configuration and applies another coordinate transformation into a local Cartesian system. Auto placement is currently not working for add-ons that address Geo mode as these use undocumented but open geographic databases. Since OME allows interactive editing of the positioned individual buildings, turning them into first-class elements like virtual vehicles or people, it is the preferred vbstool target.

5. Performance Evaluation

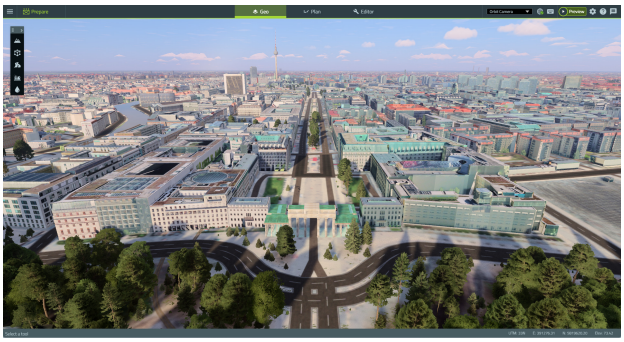
The open Berlin data set was prepared on a Linux server with 32 available physical cores of an AMD EPYC 7663 CPU, 256 GiB of memory, and rotating disk storage in a RAID-6 setup. DTM processing took 490 seconds, with 173 seconds spent on adjusting the terrain to the buildings. Conversion of the TOM and LULC layer required 567 and 859 seconds, respectively. Obtaining individual CityGML models with optimized textures for the Mitte borough finished after 37937 seconds, which translates to 1.92 seconds per structure. Table 1 lists the time needed for each step of the building preparation workflow. After a total of about 11 hours, the terrain database was complete.

VBS4 version 22.1.1.2 ran under the Windows 11 operating system on a workstation PC with an 18-core Intel Xeon W-2195 processor, 256 GiB of main memory, external SSD storage, and an nVidia Quadro RTX 5000 graphics card with 16 GiB of video RAM. The bitmap products were integrated into the simulation within 90 minutes, and the same amount of time was required for the interactive erasure of the default structures.

processing step	runtime (s)
data set decompression	52
model rewriting	136
source texture check and fixing	41
texture normalization	2695
model split into sets of 150 buildings	238
texture consolidation and interpolation	34599
model set split into individual buildings	176
building preparation time in total	**37937**

Table 1. Runtime details of building preparation workflow.

Conversion of the prepared CityGML building models with automatic placement using vbstool took 7.16 hours and yielded a custom add-on of 8.9 GiB in size. Thus, the overall setup time for a basic geospecific scene of Berlin-Mitte was about one day. However, a few extra days should be planned for road work like checking the auto-generated highways, adding missing infrastructure such as bridges or tunnels, detailed landscaping, and other repairs. Figure 9a shows a panoramic view of the prepared Berlin scene as it appears in the simulator after data import without fine-tuning, except for the Brandenburg Gate. This signature landmark of Berlin, which is missing in the original data set, was created manually using 3D graphics software and inserted into the battlespace. A comparison of a real street scene and its geospecific virtual counterpart can be seen in figures 9b and c. Despite different fields of view, acquisition dates and resolutions, similarities in the shapes and arrangement of the structures, the façade design and, to a lesser extent, landscape elements, are visually evident. The true-ortho mosaic and building models likewise display geographic congruence.



(a)

(b) (c)

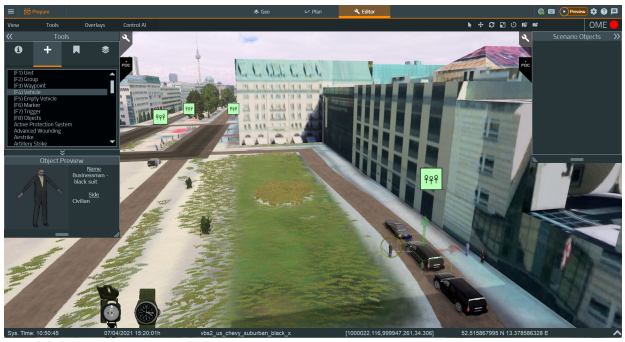
VBS4 scene derived from: DTM/TOM - Geoportal Berlin, DL-DE Zero-2.0; LULC data - OpenStreetMap, openstreetmap.org/copyright, © ESA WorldCover project 2021, with modified Copernicus Sentinel data (2021) processed by ESA WorldCover consortium, CC BY 4.0; Buildings - Berlin Partner für Wirtschaft und Technologie GmbH, custom permissive license; other elements - Bohemia Interactive Simulations/OneArc
Photo: Leonhard Lenz, 2020, CC0 1.0

Figure 9. Berlin data in VBS4 (a) panoramic city view, (b) real scene from ground perspective, (c) equivalent geospecific scene.

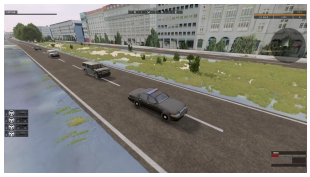
In comparison, according to (Frommholz et al., 2022), the traditional entirely manual modeling process for a site with approximately 500 buildings takes about two weeks for the predecessor of VBS4. Such workflows naturally enable an almost perfect simulation experience. However, if the amount of work required for the small scene is linearly scaled to an area with 19736 buildings like the Mitte borough of Berlin, its design would take about 1.5 years. Thus, the creation of at least a basic simulation environment from available remote sensing data must be considered a viable option for mission scenarios where time is a decisive factor.

When the virtual scene has been configured, it takes about 30 minutes on the workstation from starting VBS4 to the Offline Mission Editor being ready for operation. Experienced users can then usually set up the designated mission elements and goals within a few hours. This, for example, includes adding vehicles, friendly and less friendly non-player characters (NPCs), waypoints, obstacles, danger zones, and other features. Figure 10 shows the formation of a vehicle group in OME tasked with transporting a particularly vulnerable person from an embassy building near Brandenburg Gate to the airport.

Once the cars were set up and simulation started, interactive convoy training was possible at 15 to 40 frames per second on a 1080p display and VBS4 at medium details. Thus, the serious gaming experience in the extensive environment was adequate to smooth. Hardware monitoring revealed that the 3D engine was apparently not limited by the graphics card nor the workstation’s central processing unit. Since main memory and video RAM usage peaked at 60 GiB and 12 GiB respectively, which is below the installed capacities, the cause of the performance issues must be further investigated. There were also occasional instabilities during object manipulation in OME and simulation, forcing a restart of VBS4. This should be resolved through a software update.



(a)



(b) (c)

VBS4 scene derived from: DTM/TOM - Geoportal Berlin, DL-DE Zero-2.0; LULC data - OpenStreetMap, openstreetmap.org/copyright, © ESA WorldCover project 2021, with modified Copernicus Sentinel data (2021) processed by ESA WorldCover consortium, CC BY 4.0; Buildings - Berlin Partner für Wirtschaft und Technologie GmbH, custom permissive license; other elements - Bohemia Interactive Simulations/OneArc

Figure 10. Convoy training in VBS4 (a) scenario setup in OME, (b) official vehicles on Unter den Linden boulevard (road length 1.5 km), (c) smoke from firecracker thrown into path of lead car.

6. Conclusion

This paper disclosed details on how to rapidly transform comprehensive open 2D and 3D remote sensing data into a large-scale terrain database for the Virtual Battlespace 4 simulator

deployed in a civil security context. The preparation and import workflows to be possibly adapted to other game engines turned a terrain model, orthophoto backdrop, land cover layer and textured building meshes of the city of Berlin into a geospecific virtual representation of increased recognition value. Interactive simulation of a vehicle convoy, as used for high-ranking individuals with special security requirements, was successfully demonstrated although the software got pushed to its limits.

Future work will focus on visualization performance to accommodate even larger scenes in VBS4. Despite the already low polygon count of the CityGML LOD2 buildings, even simpler LOD1 box models could be automatically derived and switched to for distant structures to reduce the load on the simulation hardware. Textures then need to be averaged and warped onto the new meshes to ensure an acceptable amount of immersion. Further, functional enhancements remain to be implemented for the terrain database. Working windows and doors, as already supported in CityGML, would allow the player to enter buildings provided that their interior is at least coarsely modeled. To obtain information about the room layout, terrestrial sensing or official plans could be used to geometrically construct and semantically annotate floors, walls, and stairwells. If such data is unavailable, a generic interior could be generated at the expense of realism.

References

Akenine-Möller, T., Haines, E., Hoffman, N., 2008. *Real-Time Rendering 3rd Edition*. A. K. Peters, Ltd., Natick, MA, USA.

Bohemia Interactive Simulations, 2022. VBS4 22.1.0 Manuals. Bohemia Interactive Simulations, Prague, Czech Republic.

ESRI, 1998. ESRI Shapefile Technical Description. White paper, Environmental Systems Research Institute, Inc., Redlands, CA, USA.

Frommholz, D., 2022. Image Interpolation on the CPU and GPU using Line Run Sequences. *ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci.*, V-2-2022, 53–60.

Frommholz, D., 2025. Locally rendered high-resolution Land Use/Land Cover Bitmaps from OpenStreetMap Data for geospecific 3D Simulation. *ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci.*, X-G-2025, 269–276.

Frommholz, D., Kuijper, F., Bulatov, D., Cheung, D., 2022. Geospecific terrain databases for military simulation environments. *Electro-Optical Remote Sensing XVI*, 12272, International Society for Optics and Photonics, SPIE, 1227207.

Frommholz, D., Linkiewicz, M., Meißner, H., Dahlke, D., 2017. Reconstructing Buildings with Discontinuities and Roof Overhangs from Oblique Aerial Imagery. *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci.*, XLII-1/W1, 465–471.

GDAL/OGR contributors, 2025. GDAL/OGR Geospatial Data Abstraction Library. Open Source Geospatial Foundation. gdal.org (9 June 2025).

Geofabrik GmbH, 2025. OpenStreetMap Data Extracts. Download server. download.geofabrik.de (18 July 2025).

Gruber, T., 2019. Die Bestimmung physikalischer Höhen mit Satelliten. 20. Internationale Geodätische Woche, Obergurgl, Austria.

Hansen, H. S., Schrøder, L., 2019. The societal benefits of open government data with particular emphasis on geospatial information. *Electronic Government and the Information Systems Perspective*, Springer International Publishing, Cham, Switzerland, 31–44.

Höhl, W., 2020. Official Survey Data and Virtual Worlds - Designing an Integrative and Economical Open Source Production Pipeline for xR-Applications in Small and Medium-Sized Enterprises. *Big Data and Cognitive Computing*, 4(4).

Khayyat, M., 2015. Open Data Licensing: More than meets the eye. *Information Polity*, 20(4), 231-252.

Lee, A., Chang, Y.-S., Jang, I., 2020. Planetary-Scale Geospatial Open Platform Based on the Unity3D Environment. *Sensors*, 20(20).

Lei, B., Stouffs, R., Biljecki, F., 2023. Assessing and benchmarking 3D city models. *International Journal of Geographical Information Science*, 37(4), 788–809.

MLIT, 2025. PLATEAU Handbook of 3D City Models - 3D City Model Data Conversion Manual Revision 3.0. Ministry of Land, Infrastructure, Transport and Tourism, Tokyo, Japan.

Mooney, P., Minghini, M., 2017. *A Review of OpenStreetMap Data*. Ubiquity Press, 37–60.

Open Geospatial Consortium, 2012. OGC City Geography Markup Language (CityGML) Encoding Standard version 2.0.0. ogc.org/standards/citygml. (10 October 2025).

Open Geospatial Consortium, 2019. OGC GeoTIFF Standard version 1.1. ogc.org/standards/geotiff. (10 October 2025).

OpenStreetMap, 2025. OpenStreetMap webpage. OpenStreetMap Foundation. openstreetmap.org (10 July 2025).

Paltashev, T., Perminov, I., 2014. Texture Compression Techniques. *Scientific Visualization*, 6, 96-136.

Santa-Cruz, D., Grosbois, R., Ebrahimi, T., 2002. JPEG 2000 performance evaluation and assessment. *Signal Processing: Image Communication*, 17(1), 113-130.

Senatskanzlei Berlin, 2025. Berlin Open Data. BerlinOnline GmbH. daten.berlin.de (12 November 2025).

Vohera, C., Chheda, H., Chouhan, D., Desai, A., Jain, V., 2021. Game engine architecture and comparative study of different game engines. *2021 12th Int. Conf. on Computing Communication and Networking Technologies*, Kharagpur, India, 1–6.

Willenborg, B., Sindram, M., Kolbe, T. H., 2018. *Applications of 3D City Models for a Better Understanding of the Built Environment*. Springer International Publishing, Cham, Switzerland, 167–191.

Würstle, P., Padsala, R., Santhanavanich, T., Coors, V., 2022. Viability Testing of Game Engine Usage for Visualization of 3D Geospatial Data with OGC Standards. *ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci.*, X-4/W2-2022, 281-288.

Zanaga, D., Van De Kerchove, R., Daems, D., Keersmaecker, D. et al., 2022. ESA WorldCover 10m 2021 v200. doi.org/10.5281/zenodo.7254221 (17 September 2024).