Automated 3D Building Generation at Global Scale Based on Satellite Imagery

Arno Hollosi, Thomas Menzel-Berger
Blackshark.ai GmbH, Graz, Austria
ahollosi@blackshark.ai,
tmenzel-berger@blackshark.ai

Hannes Walter, Daniel Lahm
Blackshark.ai GmbH, Graz, Austria
hwalter@blackshark.ai,
dlahm@blackshark.ai

ABSTRACT

In the past the production of a photorealistic, geo-typical, 3D digital surface model of the entire globe has been a near impossibility. Ensuring that the model is geo-referenced, high-performance, and up to date have further complicated the task. Any manual approach is much too slow and automated methods have lacked region-specific aspects. Moreover, the amount of satellite input imagery data and necessary processing power for automated methods has invariably been too massive or client-side rendering of photogrammetric models too slow.

In 2020 a novel approach for a 3D reconstruction solution was developed to overcome said limitations. It consisted of an end-to-end, cloud-computing-based geospatial platform that extracts terrain and infrastructure attributes from a global satellite imagery data set. This data was then transformed into a WGS84 georeferenced, geo-typical, photo-realistic, and semantically defined 3D digital twin of worldwide human-created infrastructure, which was capable of being streamed or stored entirely offline. A highly parallelized and resilient cloud architecture was used to deliver an accurate reconstruction of approximately 1.4 billion buildings and structures.

At the core of the 3D reconstruction lies the Procedural Geometry Generator (PGG). It uses a custom grammar that allows for a diversity of features previous solutions were unable to provide. The solution ensures that an unending proliferation of labyrinthine rules can be avoided. Diversity is realized using regional characteristics, biome definition, unique building constraints and pseudo-randomization. The result is that the procedural definition of buildings defines every single building in a unique, geo-typical- way with high performance. By performing reconstruction on the client side, PGG allows for reconstructions that would otherwise be untenable due to the required amount of streamed data. This method not only increases asset diversity, but also massively decreases required resources.

The resulting platform is a solid foundation for professional image generators, flight simulators and special use cases such as pilot training and sensor simulation. Additional challenges include persistent, synchronized, multi-view, multi-client states or material attributes at global scale.

ABOUT THE AUTHORS

Arno Hollosi is CTO at Blackshark.ai, a company that seeks to unlock planetary insights by building the world's leading authentic 3D digital map. Previously, he was a professor at FH CAMPUS 02 in Graz, worked for a German e-commerce startup, as well as for Siemens and the Austrian government. In his free time, he runs one of the oldest wikis worldwide.

Thomas Menzel-Berger is a research manager at Blackshark.ai and coordinates R&D projects and the IP portfolio of the company. He has a background in science & technology studies and previously worked as a researcher in interdisciplinary projects focusing on socio-technological issues in energy systems and ICT industries.

Hannes Walter is VP Product Marketing at Blackshark.ai. He has a background in UX design, procedural 3D online-representation and additive manufacturing. Previously he has worked for Austria's largest ICT integrator driving their industrial extended reality solution portfolio. He likes to be at the melting point between technology, art, and business.

Daniel Lahm is a technical writer at Blackshark.ai where he documents the changing and developing technologies. His goal is to produce clear concise and accessible information for all stakeholders. He has previously worked in adult education and has a background in philosophy. He is interested in the mechanics of board game design.

Automated 3D Building Generation at Global Scale Based on Satellite Imagery

Arno Hollosi, Thomas Menzel-Berger
Blackshark.ai GmbH
Graz, Austria
ahollosi@blackshark.ai,
tmenzel-berger@blackshark.ai

Hannes Walter, Daniel Lahm
Blackshark.ai GmbH
Graz, Austria
hwalter@blackshark.ai,
dlahm@blackshark.ai

INTRODUCTION

Planetary scale geo-typical scenery for simulation and digital twin use-cases requires massive amounts of 3D reconstructed data. This requirement is a major obstacle for all applications that seek to represent building infrastructure at continental or even planetary scale. The conventional approach is to have a powerful (cloud-based) server architecture with enough storage capacity to manage huge sets of geospatial data. This data is then split into tiles and fed piece-meal to local 3D clients. This conventional approach limits the use to online scenarios. In addition, established file formats may be feature rich and proven, but are inefficient, sometimes wasteful, and incapable of storing planet-sized 3D worlds. This paper describes a new approach to achieve a global-scale 3D reconstruction that provides solutions to the aforementioned challenges.

The approach utilizes a patented innovation (Thaller et al., 2019) called *Procedural Geometry Generator* (PGG) which applies a novel shape grammar. This allows streamlined and efficient transmission of 3D reconstruction metadata to a client's engine for the generation of a holistic global 3D environment. The amount of source data and grammar is small enough to enable entirely offline operation. Source data comes in the form of infrastructure attributes, such as building footprints, and other data sources such as biome maps or semantic metadata for cultural zones and styles.

This paper has three major objectives. It will first demonstrate how a procedural approach using a domain specific language (DSL) can generate objects in any desired detail based on a limited set of data points. Secondly, it will discuss how this approach is amenable for adding and combining additional data sets. Thirdly, performance metrics will be provided to illustrate and compare this approach's ability to generate 3D environments in real time at large scale.

METHODOLOGY

This paper will focus on the reconstruction of human-made infrastructure for global-scale simulations. Such simulations draw data from at least two different sources: a terrain model and infrastructure data. The terrain model consists of an elevation data set (digital terrain model, DTM) and ground textures. These textures may either be synthetic or raster data from satellite or aerial imagery. The infrastructure is typically stored as vectors and geometry accompanied by texture graphics. Vector data might be available in different levels of detail (LOD).

Conventional Approach

The conventional approach is to generate and store the infrastructure vector data on the server side and stream the content in tiles to the client. For example, the CDB standard (Reed, 2021) defines tiled data sets which contain geo-typical and geo-specific 3D objects. These objects are typically stored in the *OpenFlight* format (Presagis, 2018). Among other data, an *OpenFlight* file contains all the vertices and geometries that make up the 3D object. This information is then streamed to the client using the CDB API, as seen in Figure 1.

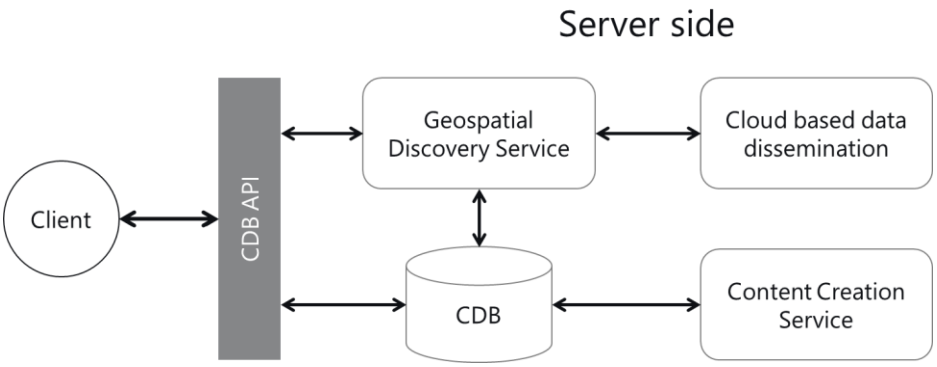


Figure 1: Simplified architecture diagram of a typical CDB deployment

A similar approach is used by the *3D Tiles* specification (Cozzi, Lilley, & Getz, 2018). *3D Tiles* is built on glTF (Khronos, 2021), an open standard for streaming and rendering of 3D models. Like the CDB setup, glTF contains the mesh of 3D objects that are to be placed into the simulation. Other standards, for example, CityGML (Kolbe et al., 2021), OBJ (Wavefront, 1992), and FBX ("FBX", 2013), all store the 3D object mesh's vertices and eventually stream them to the client.

Given that detailed infrastructure objects have many thousand vertices at their highest LOD, the data required to store the meshes is a demanding amount. For example, a test area with 1452 (unique) buildings needs 339MB space as OBJ (uncompressed) and 43MB as FBX (compressed).

It is therefore unfeasible to store the billions of infrastructure objects required for the entire planet in such formats. That is why some formats support geo-typical models that can be reused and copied multiple times into the same scene. The drawback of this approach is object repetition, as they do not have much variability. The resulting scenes might become visually monotonous.

An improved approach, as used by Esri's CityEngine (Arisona, 2021), is to generate the building meshes procedurally on the server side and stream them in one of the mentioned file formats to the client. While this solves the narrow variability of the objects, it comes at the cost of streaming substantial amounts of data to the client. This may limit use cases to scenarios where a high-bandwidth connection is available to the client.

Novel Approach

The PGG technology improves on the conventional approaches by defining a building grammar that is *executed on the client side* to produce the 3D meshes. That is, the procedural generation happens on the client side and, as only the source metadata needs to be transmitted, the streaming efficiency is greatly increased. Even offline modes that hold whole nations, continents, or the entire earth are possible.

PGG is grounded in semantically enriched geospatial data. Figure 2 gives an overview of the basic workflow. On the server side (top in Figure 2), the data is extracted from aerial or satellite imagery with trained machine learning (ML) models and traditional computer vision methods. For flight simulators, we typically use 3-channel (RGB) satellite imagery with 50cm resolution as input. If higher detail is required, aerial imagery with up to 7.5cm resolution is used as input to our server-side detection algorithms.

The algorithmically derived data includes, among other things, building footprints and boundaries, roof typology and color, geometric orientation, and building height (see Figure 3). Similar approaches have been discussed in (Chatterjee, Brassard, & Patel, 2020). Unlike these previous approaches, our server processing uses ML models that work across the whole planet, regardless of biome. Depending on the type and resolution of source material (e.g., off-nadir imagery) façade feature data can also be detected. This data can be combined with information from other sources, such as Open Street Map (OSM), or customer specific data. All this is done server-side and independent of client usage. The resulting data is stored in common formats such as *GeoJSON* (Butler et al., 2016) or alternatively in custom binary formats to save space.

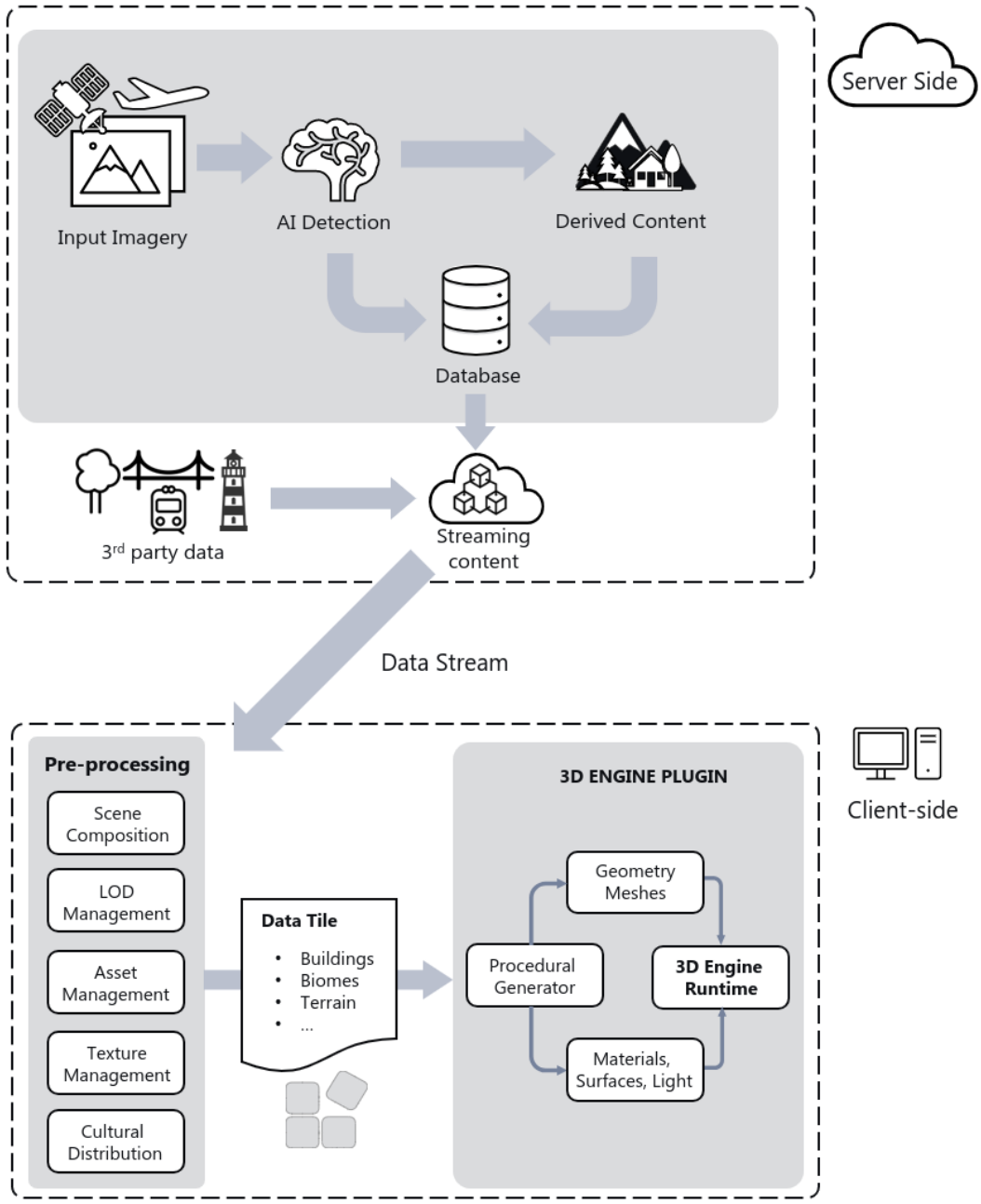


Figure 2: Division of work between server side (detection) and client side (reconstruction)

Figure 3: Extraction of building footprints and roof typologies with ML models.

When the client wants to render the scene, this data is streamed from the streaming content repository in Figure 2. The data stream is then preprocessed on the client side (see Figure 2, bottom). Some data sources might be small enough to be stored in their entirety on the client side. One data source stored in this manner is cultural distribution, which is used to determine which types of single-family homes and other buildings are reconstructed in a given region. Material and script libraries, which are not expected to change during simulation and can therefore be downloaded before the simulation begins, are similarly stored on the server side to further decrease real-time streaming demands. Other data sets, such as building attributes, are typically streamed as tiles.

The preprocessing step manages the data download, the level of detail (LOD) management, textures, and assets. It produces a rich description for each tile ("data tile"). In this step, cultural attributes are contextually or geographically added or enriched depending on the location of the geospatial source data, i.e., data is contextualized for reconstruction. Based on this input and pre-defined rule sets, the procedural generator determines the types of objects to be reconstructed and their rendering details. Information on the terrain (derived from digital surface models (DSM) or digital terrain models (DTM) and land use/land cover (LULC) analytics of the respective tile are also embedded and used as input to the grammar rules.

In the next and last step, the Procedural Geometry Generator (PGG) uses all information contained in the tile to produce the final 3D meshes suitable for consumption inside the 3D rendering engine (e.g., Unity, Unreal, or custom engine).

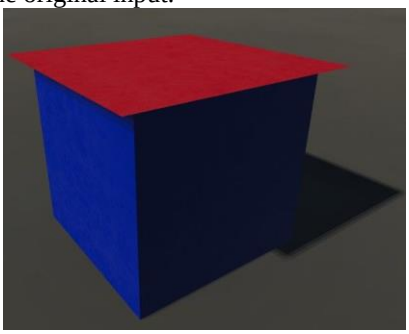
PGG

The procedural grammar is written in a textual domain specific language (DSL). The description is stored in plain-text files called modules. Modules consist of three sections: import declarations; variable, texture, and function declarations; and finally, rules. Rules are applied to shapes, such as a polygon representing a footprint, and describe the transformations the shape is subject to.

Listing 1 shows an example where the *Start* rule is applied to a square polygon. First the polygon is resized (*SetSize*), then extruded (*Extrude*) to give it volume. Next, *CompSplit* divides the resulting cube into three distinct surfaces: *top*, which is the lifted input polygon facing the set positive direction axis (red in the image), *mantle*, which is the façade (faces extending between *top* and *bottom*; blue in the image), and *bottom*, which refers to the mirrored version of the top. In the listing, each surface undergoes particular transformations (e.g., top is resized by a factor of 1.2, which is why it protrudes the blue cube below). Typically, the *bottom* portion is discarded to further save resources (see *Discard* operator in the listing.)

Extrude, *Resize*, *Generate* etc. are built-in operator primitives of the DSL. Currently, PGG has over 100 built-in operators. Together, they allow arbitrary transformations based on the original input.

```
Start:
    SetSize(5, 5)
    Extrude(5)
    CompSplit {
        top: Resize(1.2)
             DrawTexture("", #FF0000)
             Generate;
        mantle: DrawTexture("", #0026FF)
                Generate;
        bottom: Discard;
    };
```



Listing 1: A simple rule for drawing an overhanging rooftop

Listing 2 shows how custom functions can be defined. The grammar operates on typed values: the parameter *value* is a *float*; the function *grey()* returns the type *vec3f*, which is a vector of three floats (here RGB colours). The DSL contains over 50 built-in functions, mostly for mathematical operations (e.g., cosine, exponential, rounding) and vector handling (for 3D points and surfaces).

```
function grey(float value) -> vec3f =
    vec3f(value, value, value);

function darker(vec3f color, float amount) -> vec3f =
    Vec3f.interpolation(color, black, amount);
```

Listing 2: User-defined functions for returning a grey colour and darkening an existing colour.

As modules can import other modules, technical artists and designers can build an ever-growing library of helper rules and functions and in turn mould the DSL into a *building pattern language* (Alexander, 1977) of their own choice. Step by step, complicated shapes, surfaces, and volumes can be constructed from a given set of input data. Figure 4 gives an example of this process.

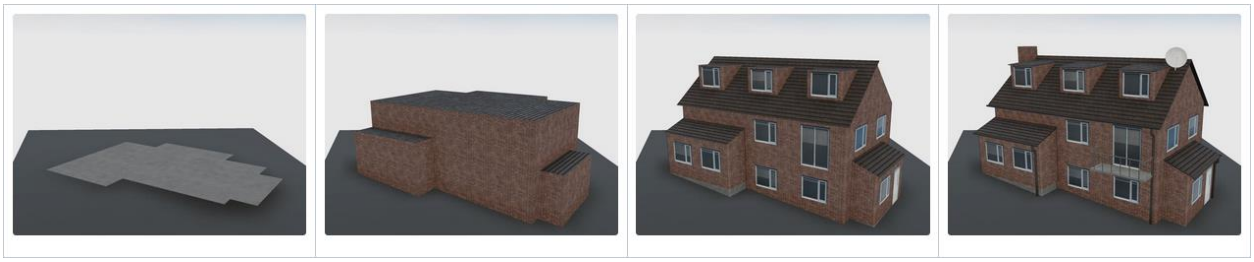


Figure 4: Gradual application of rules to a 2D footprint, based on desired LOD

The PGG DSL can be compiled to C++ code for fast processing. The resulting binary is either part of the PGG client program or downloaded on request. All operators, functions, and rules are therefore executed on the client side. The space intensive mesh of a 3D object itself is never transmitted. Only metadata, such as footprints and other attributes need to be transmitted to the client. This is the main advantage of this novel approach.

Distribution and Variation

If rules and functions operated on building data alone, the very same rule set would be applied across the world. This would lead to a very uniform distribution without cultural variation. For this reason, PGG can incorporate other geographic information as well, e.g., population density, cultural region, altitude, slope of terrain, etc. Accordingly, PGG can select rules and features with respect to the specific geolocation. The current implementation comprises over one hundred building types that can be varied through distribution rules. As an example, Figure 5 shows eight different buildings that are more common in some regions and neighborhoods/zones than in others. A distribution data set can describe the incidence of each type of building and the relative odds of one building over the other in a region.

Figure 5: Eight different buildings that are common in different regions and/or different zones (urban vs. rural)

This feature is not exclusive to buildings but can be used for vegetation as well. While, for example, the vegetation mask may declare a certain area simply as "*forest*," this information can be combined with the typical distribution of trees for that classified region. Thus, a forest might consist mostly of softwood trees in one area and mostly of palm trees in another.

Together with additional geo-typical and regional data sets, any location in the world can be populated with buildings, vegetation, and objects that are typical for that region. Cultural variability is thus achieved by deploying architectural and geo-specific library rules for each geographic location. This results in a variation rich world where users immediately recognize the geo-typical style and perceive the simulation as more realistic.

Distribution uses both classifiers, as discussed above (cultural regions, population density, land use), and building attributes (height, roof type, footprint ratio, area). Using the attributes, the likelihood of a particular building is determined making sure not to break the configured constraints (e.g., no 3-story skyscrapers or 100-floor bungalows).

As the rules are applied to the input shapes and accompanying data (e.g., building height and roof type), the single set of rules can create an arbitrary amount of variation. In addition, the pseudo-randomization can be used to enable or disable certain building features, choose among a set of textures, etc. The result is almost limitless variation, as shown in Figure 6. After determining the likelihood of a rule set, a pseudo-randomization algorithm is used, so that the reconstruction will be the same on every client. This is achieved by linking the initial randomization seed to the global position of the object.

Figure 6: Variations for a rule set for suburban houses.

Semantics and Additional Use Cases

All features of procedurally generated objects, especially buildings, are well defined by the grammar. This means in turn that each pixel in the final rendering is accounted for, including its color, reflection, roughness, etc. Even "invisible" attributes, like material, density, thickness, etc., can be attributed down to the pixel level because of the procedural generation. For example, this enables realistic rendering of scenes at night (with backlit windows) like the one shown in Figure 7.

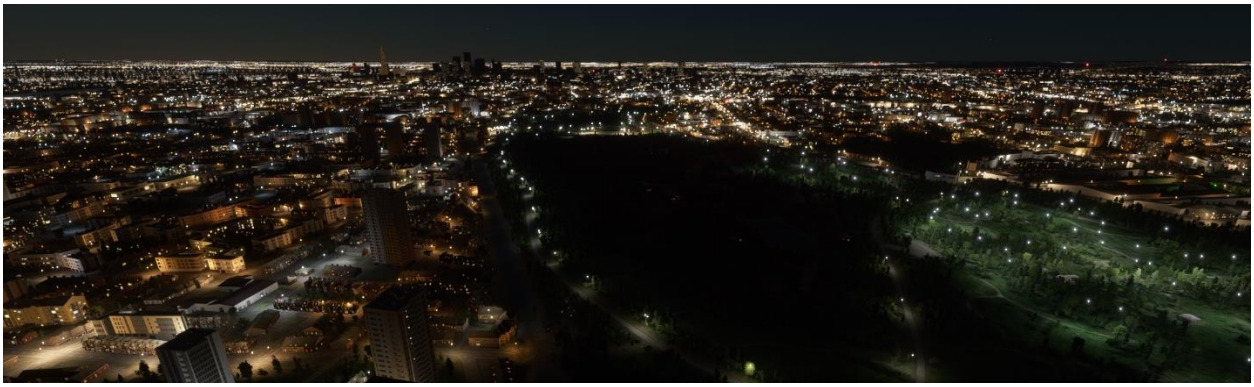


Figure 7: Example of a reconstructed night scene.

The same functionality can be used to render specific views for sensor simulations. Figure 8 shows an example of a runway with airplanes. The left image shows the visual rendering, the right image a rendered segmentation mask which can be used for sensor training.

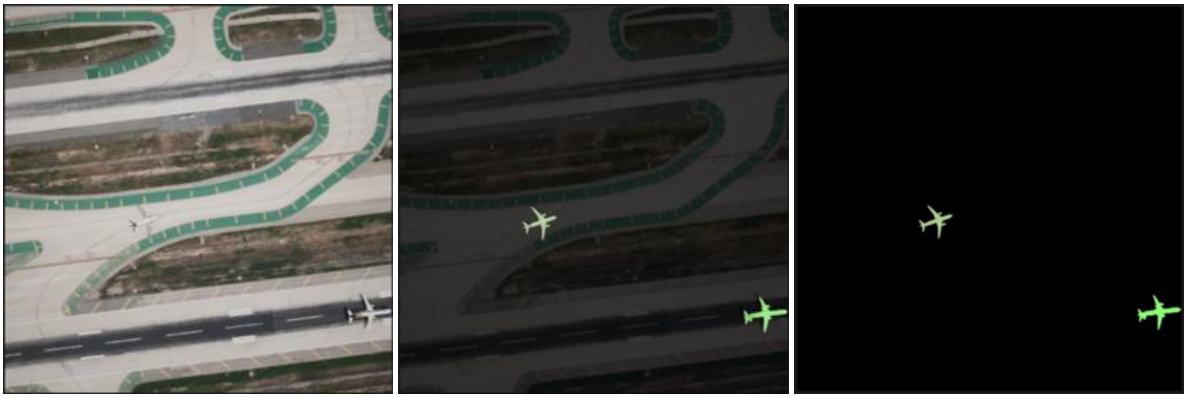


Figure 8: Visual rendering, mixed rendering, and semantic rendering of a PGG scene with airplanes

The mesh generation is typically tailored to the specific needs of the 3D rendering engine. In addition, custom renderers have been implemented that emit the desired LOD mesh in a format consumable by other tools, such as physics simulations, for radio wave propagation or calculation of computer-generated forces (CGF). Figure 9 shows the output of a custom renderer that uses a set of custom attributes for coloring infrastructure and biomes.

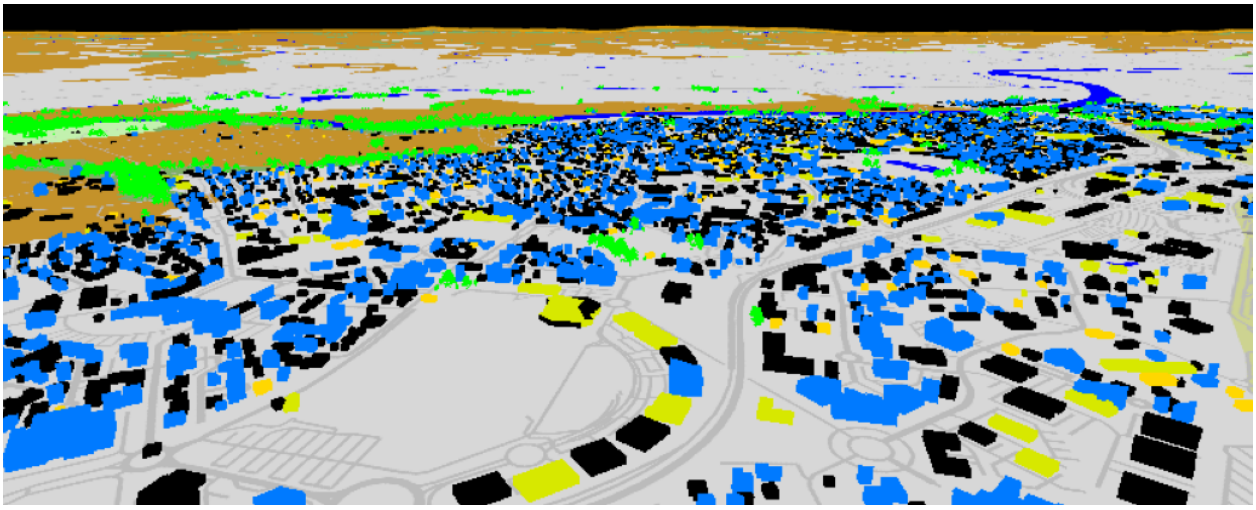


Figure 9: Example of a sensor view of a semantically annotated scene.

RESULTS AND PERFORMANCE

This section provides an overview of the PGG's performance and a comparison with existing solutions that aim for similar outcomes.

Storage Size

Compared to conventional approaches, where the entire mesh of an object needs to be stored in the file, the PGG approach requires several orders of magnitude less storage. For example, this approach was successfully implemented in Microsoft's Flight Simulator 2020. Its global dataset consists of an estimated 1.4 billion buildings requiring only 27,4 GB of storage space in compressed form (on average 100 bytes per building, comprising vector data footprints and roof type with colours). Compared to over 200.000 bytes per building as uncompressed OBJ and about 30.000 bytes per building as compressed FBX, the space savings are evident. In addition, textures take up a further 500 MB of space (not including aerial images and digital elevation model).

This is small enough to be stored on laptops or even powerful mobile devices. Despite the small size, the generated world is rich in variation, as the same building type is not simply placed repeatedly. Instead, as described above, rule sets can efficiently encode geo-typical architectural patterns (only demanding about 100 MB) and produce unlimited permutations based on the input data provided.

Performance (Realtime Generation)

Procedural generation of objects is a well-known, common technique used in applications ranging from games to simulation environments. What makes PGG unique is that reconstruction is done in real-time on the client side. At peak performance, the 3D client side can generate over 40,000 highly detailed objects per second. Reconstructing an entire city with one million buildings taking less than half a minute. As shown in Figure 2, the construction of the 3D mesh is done inside the 3D rendering engine itself. Hence, the mesh is optimized for the engine at hand and does not need to cater to the lowest common denominator of different rendering engines. Intermediate results are cached client-side, so that they are reusable. The result is a simulation that can easily achieve 60 Hz refresh rates and use engine features to the fullest degree to create high-end results (see Figure 10).

Figure 10: High-detail, large-scale reconstruction from analysed 2D ortho images. Tripoli, Libya.

Limitations

PGG's approach is effective because of the self-similarity of human-made structures as well as the similarity of vegetation and biomes in general. A set of a few dozen rules can produce authentic, highly detailed 3D structures. Given a library of one hundred building archetypes with high variability in their rule sets, the whole earth can be populated with geo-typical buildings. A library of this size may, for example, contain several types of family homes for North and South America, Asia, Europe, and Africa, each in urban and rural flavours. This would however mean that all single-family homes have the same style within each region (i.e., with only the aforementioned continental regions there would be no difference between, for example, India and Japan, as both are reconstructed according to the Asian rule set).

Consequentially, the more specific the models should be, the more archetypes are needed, e.g., for differentiating buildings of London's suburbs versus Glasgow's suburbs or urban buildings in Hong Kong versus the same in Osaka. PGG's composable nature, where modules may import other modules, mitigates the expected resulting growth in the number of rules to some extent. For example, rules for windows or rain gutters can be reused across different building types. Hence, given a composable library of building features, artists can efficiently modify or adapt rule sets for a desired region.

Landmark buildings are unique by their very nature. If geo-specific reconstruction of landmarks is required within a simulation, then PGG can provide two options. As a first option, a specific rule set for the landmark can be written for the building. Depending on the required LOD this might be preferable. A second option is a ready-made 3D object mesh used as a replacement for the default reconstruction by PGG. The mesh could be created by an artist or derived from point clouds based on photogrammetry or LIDAR scans. PGG's pre-processing step in Figure 2 enables mixing and matching of such 3D objects with auto-generated reconstructed structures. Figure 11 shows an example, where a custom object, the Space Needle building in Seattle, is placed amidst PGG buildings.

Figure 11: The Space Needle is a custom 3D mesh

Future Work

In its current form PGG and our grammar library are most suitable for air system simulators. For ground-based simulations where the viewpoint is only a few meters above ground level, some enhancements need to be made. While PGG itself has no built-in limitations, the buildings in our grammar library would need an additional LOD to make them visually appealing for ground-based simulations. In addition, features like bump maps should be added to give the surfaces more realistic surfaces with bumps, wrinkles, and protrusions.

The recently released Unreal Engine 5, with its Nanite virtualized geometry (Epic Games, 2022), allows for new directions: instead of building meshes from simple geometries, we are actively investigating using complex 3D scans, such as Quixel's Megascan library (Epic Games, 2020), as basic elements to construct buildings, terrain, and other objects. The grammar needs to be extended with suitable operators and functions to make handling these elements easy.

Dynamic systems, i.e., systems that modify the buildings or the terrain, need better support. Currently, PGG only supports a "1-bit" solution: either the building is there, or it is not. For example, to simulate damaged buildings, the "damage algorithm" needs to first query the original object mesh from PGG, then apply its damage transformation. The resulting mesh is then added as custom 3D mesh, just like the Space Needle in Figure 11. The original PGG building is marked as *removed* to prevent rendering two objects in the same place. This workflow will be improved to better support dynamic systems and enhance their performance.

For CFG and other physics simulations, the 3D meshes produced by PGG can be optimized further. Right now, they simultaneously contain too many and too few details. Too many details in the sense that some surfaces and vertices are only necessary for visualization. Too few details in the sense that some attributes typically used for physics are not yet queryable through the PGG API.

Currently, PGG's interoperability is limited to FBX, OBJ, GeoPackage (Yutzler et al., 2021), and similar 3D formats. Active work is being done on integrating PGG with CDB, USD (Pixar, 2018) and other established standards.

CONCLUSION

By moving the procedural generation to the client, our approach needs orders of magnitude less storage space or transmission bandwidth than conventional approaches. In addition, using a feature-rich procedural grammar allows for the design of highly variable building archetypes in infinite permutations. A combination that enables planet-scale simulations with constrained compute and space resources. Even offline modes on standard mobile devices are possible. PGG exploits the natural self-similarity of the world, be it buildings, vegetation, or terrain. The power of this approach has been successfully showcased in Microsoft's Flight Simulator 2020.

The most value of the described approach is seen when there is only limited data available, such as satellite or aerial imagery that only provides a bird's-eye view. If, on the other hand, a scene is available as high-resolution point cloud, PGG in its current form can be used to add semantics, which in turn enables sophisticated sensor simulation. This area requires further research to explore the full potential of semantic 3D reconstruction.

With the constant expansion of PGG's features and capabilities this technology can be a valuable building block for future digital twin instalments that will enable the much-proclaimed advent of the geospatial metaverse. As artificial intelligence analytics and semantic 3D reconstructions develop in a mutually beneficial co-evolution, PGG aims to be the solution that combines the power of AI detection schemes and future 3D simulation environments in the most data efficient way.

ACKNOWLEDGEMENTS

Richard Maierhofer (senior 3D graphics developer at Blackshark.ai) helped with revision of this paper.

REFERENCES

Alexander, C. (1977). *A pattern language: towns, buildings, construction*. Oxford University Press.

Arisona, S., (2021) Esri, *From CityEngine to Unreal Engine: the journey from first design steps to high-quality real-time visualization (part 3/3)* Esri Community. Retrieved May 11, 2022, from https://community.esri.com/t5/arcgis-cityengine-documents/from-cityengine-to-unreal-engine-the-journey-from/ta-p/1100392

Butler, H., Daly, M., Doyle, A., Gillies, S., Hagen, S., Schaub, T. (Eds.). (2016) *The GeoJSON Format*. Internet Engineering Task Force (IETF), Request for Comments: retrieved May 11,2022 from 7946. https://datatracker.ietf.org/doc/html/rfc7946

Chatterjee, B., Brassard, H., Patel, B. (2020) *Creating Geo-specific Synthetic Environments using Deep learning and Process Automation*. Proceedings of Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC) 2020.

Cozzi, P., Lilley, S., Gabby, G. (Eds.). (2018) *3D Tiles Format Specification*. Version 1.0. Cesium GS, Inc. Retrieved May 11,2022 from https://github.com/CesiumGS/3d-tiles/tree/main/specification

Epic Games Inc. (Pub.). (2020) *Creating digital worlds with Megascans and Unreal Engine.* Epic Games Inc. Retrieved May 25, 2022, from https://www.unrealengine.com/en-US/blog/creating-digital-worlds-with-megascans-and-unreal-engine

Epic Games Inc. (Pub.). (2022) *Nanite Virtualized Geometry.* Epic Games Inc. Retrieved May 24, 2022, from https://docs.unrealengine.com/5.0/en-US/nanite-virtualized-geometry-in-unreal-engine/

FBX binary file format specification. (2013 August 10). Retrieved May 11, 2022, from *Blender Blog*: https://code.blender.org/2013/08/fbx-binary-file-format-specification/

Khronos 3D Formats Working Group (Pub.). (2021) *glTF 2.0 Specification*. Version 2.0.1. Khronos Group. https://www.khronos.org/registry/glTF/specs/2.0/glTF-2.0.html

Kolbe, T. H., Kutzner, T., Smyth, C. S., Nagel, C., Roensdorf, C., Heazel, C. (Eds.). (2021) *OGC City Geography Markup Language (CityGML) Part 1: Conceptual Model Standard*. Open Geospatial Consortium. Retrieved May 09, 2022, from https://docs.ogc.org/is/20-010/20-010.html

Pixar Animation Studios (Pub.). (2018) *Usdz File Format Specification* version 1.2 Retrieved May 24, 2022, from https://graphics.pixar.com/usd/release/spec_usdz.html

Presagis (Pub). (2018) *OpenFlight Scene Description Database Specification*. Version 16.7, Revision A. Retrieved May 11, 2022, from https://www.presagis.com/workspace/uploads/files/openflight16-7.pdf

Reed, C. (Ed.). (2021) *Volume 1: OGC CDB Core Standard: Model and Physical Data Store Structure.* Open Geospatial Consortium. Retrieved May 11,2022 from http://www.opengis.net/doc/IS/CDB-core/1.2

Thaller, W., Richter-Trummer, T., Putz, M., Maierhofer, R. (2019) *U.S. Patent 10,636,209*. Washington, DC: U.S. Patent and Trademark Office.

Wavefront (Pub.). (1992) *Object Files (.obj)*. Appendix B1 of Advanced Visualizer manual. Wavefront Technologies. PDF: Retrieved May 09, 2022, from https://fegemo.github.io/cefet-cg/attachments/obj-spec.pdf. Plain text: http://paulbourke.net/dataformats/obj

Yutzler, J., Daisey, P. (Eds.). (2021) *OGC GeoPackage Encoding Standard*. Version 1.3.1 Open Geospatial Consortium. Retrieved May 24, 2022, from http://www.geopackage.org/spec