Open3D: Crowd-Sourced Distributed Curation of City Models

Zhihan Lu Paul Guerrero Niloy J. Mitra Anthony Steed
University College London

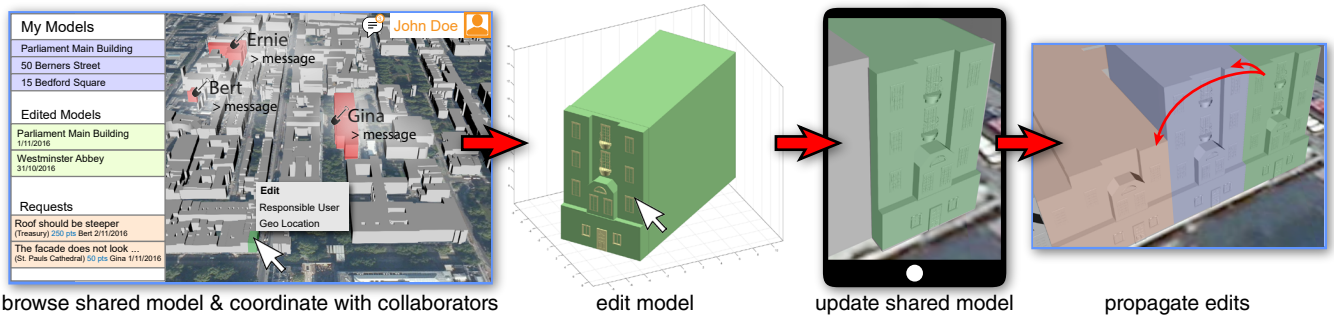

Figure 1: *We propose a system called* Open3D*, that enables collaborative curation of large-scale city models. Users of the system may view the model in the browser (left), which also acts as a social hub. Editing (middle-left) is performed using a novel parametric editing approach that is efficient and intuitive. Changes are applied back to the shared model, taking care to avoid conflicting edits (middle-right) from different users. Operations performed on one model can be retargeted (middle-right) to multiple similar models with a small number of manual adjustments. Note that the color overlays and symbols in the browser (left) are mockups of planned features.*

Abstract

Detailed, large-scale 3D models of cities are important assets for many applications. While creating such models is difficult and time consuming, keeping them updated is even more challenging. In comparison, in many domains, crowd-sourcing of data is now an established process for expanding the scope or detail of data sets. In this paper, we describe the initial prototype implementation of Open3D, a crowd-sourcing platform for distributed curation of large-scale city models. We present an open architecture with interfaces that clearly separate model storage and indexing from viewing or editing. To support collaborative editing of extremely large models, we propose to use a modeling and model description paradigm that can integrate polygon-based modeling with parametric operations. We demonstrate the main concepts and prototype through an online city model that can be synchronously edited by multiple users, with live changes being propagated among clients. The main implementation consists of a set of *web services*, which support key functions such as model storage, locks for editing and spatial queries; a light-weight *viewer* based on the Cesium library, which runs on desktops and mobile devices; and a prototype *editor*, which clients can install to edit the models.

Keywords: collaborative editing, large 3D models, shape modeling, procedural edits, architecture

Concepts: •**Human-centered computing** → **Collaborative and social computing systems and tools;** •**Computing methodologies** → *Shape modeling;*

Web3D '16, July 22-24, 2016, Anaheim , CA, USA
ISBN: 978-1-4503-4428-9/16/07
DOI: http://dx.doi.org/10.1145/2945292.2945302

1 Introduction

Digital 3D models are used in almost all areas of design and engineering from drug discovery through to architectural design optimization. With the advent of new imaging technologies (e.g., advanced remote sensing systems and consumer 3D cameras), we can now capture models that are both deep and broad in detail. In the context of urban modeling for example, it is now possible to create accurate and detailed 3D models of cities. Such models are regularly used for various mapping applications, to provide context-aware background for VR applications, or to assist users via situation-aware feedback and suggestions.

Creating such large-scale models, however, is both challenging and labor intensive. Models are still predominantly created by skilled professionals cleaning up raw scans with polygon-level operations in the paradigm of CAD/CADM software that has been around for a few decades. In terms of representation, a 3D model is a file that is loaded into a running software instance. The model is loaded, edited and then saved out back to a disk. The file that contains the model might be version-controlled or stored on a distributed file system. Such a workflow assumes 3D model creation and maintenance is handled by a single central agency, ignores high-level structural information in the original scenes, and is difficult to open up for collaborative handling.

This contrasts with the editing of large-scale 2D documents where with the evolution of web technologies, document editing is now supported by a variety of distributed tools. Platforms such as Wikipedia and Google Docs show that construction and maintenance of very large-scale resources can be done with distributed collaborators in real-time. Inspired by these paradigms, we propose Open3D as a platform for distributed, collaborative curation of very large-scale 3D models. Although there are recent efforts around web3d technologies that have targeted distribution of large models for viewing, support for simultaneous collaborative editing of these models remains limited. Our goal is to decouple storage of 3D models from viewing and editing using open interfaces that enable a web-service-like decomposition of editing (see Figure 1).

In this paper, we propose an initial architecture and prototype implementation for Open3D. We demonstrate collaborative simulta-

neous editing of a city model with each user being able to visualize segments of the city, indicate building(s) to edit, lock and update the building geometry using a novel parameterized modeling system, and then push changes to all viewers. We identified two key requirements for such a collaborative curation platform for large-scale 3D models: the platform should (i) ensure *consistency* among the 3D assets as they can be simultaneously edited by multiple users, and (ii) provide *easy-to-use* highlevel manipulation handles to simplify editing experience, especially since we do not expect users to have specialist training in 3D editing tools.

To address the first requirement, we propose a server-based locking mechanism to dynamically grant editing rights to users. Users can select building(s) to edit, and the server issues the relevant editing right(s). A more advanced version would be to allow 'merging' of possibly conflicting multiple edits for same region of interest. To address the second requirement, we propose a hybrid representation that allows highlevel parameterized edits to underlying procedural 3D models. As an added advantage, we can retarget user edits to other similar models thus further simplifying editing of large-scale models, especially in a collaborative setting. Finally, to bootstrap the initial model creation stage, we present an automatic approach to fuse information from multiple GIS sources (e.g., OpenStreetMap, ordnance survey data) to create an initial 3D model of a city using available metainformation. Open3D models can directly be browsed and investigated using a Cesium plugin and are readily viewable on mobile devices.

2 Related Work

3D Online. The goal of collaboratively editing a large-scale 3D model online draws on several areas of previous research. The first of these is the broad area of networking for graphics applications [Steed and Oliveira 2009]. Networked graphics applications such as games often use custom client software, but this can be a problem for maintenance for large user populations. The broad category of web 3d technologies uses models of distribution of code and assets that are based on well-understood models from the world-wide web [Evans et al. 2014]. Principally, that the users installs a browser client that abstracts the local operating system and provides network support, and that assets are loaded from servers. Application code is also considered an asset and thus it is loaded from external servers. Any behaviour is built using scripts that utilize the browser functionality. While plugins for specific formats have been very popular, the predominant way of exploiting web 3d is to use 3D functionality built-in to the browser through WebGL [Kronos Group]. The main function of WebGL is to represent the OpenGL functionality in the underlying operating system. It is thus rather low-level and many JavaScript libraries have emerged that provide higher level functionality. For example the XML3D [Sons et al. 2010] and X3DOM [Behr et al. 2009] libraries provide 3D scenegraph functionality on top of the Document Object Model (DOM). Both of these support storage of assets in separate files that can be served by a standard webserver.

Editing and revision management. In software development, version control systems (VCSs) such as Subversion, Git, and PerForce are ubiquitous. VCSs allow distributed editing of a shared codebase and also facilitate management of multiple versions. A key policy in software management is whether or not to allow locks on objects to prevent conflicts [Mens 2002]. At one extreme, a file or object might be locked, preventing others editing it; at the other, anyone can edit files or objects, but may need to merge changes later.

Most modern VCS technologies are not ideal when used with 3D models because they take a merge later approach. Unfortunately edits to 3D assets often change whole files because they affect a lot of vertices or reorder assets, and thus the textual edits that are supported well by current VCS tend to fail on 3D models which are either binary files or have to be treated as binary files because the structure cannot be determined. Recent work has started to tackle this. Doboš and Steed [2012] presented a system for evaluating mesh differences and resolve clashes. The XML3DRepo system then demonstrates how to use a REST-style API to store versions of files in that exchanged using the XML3D format [Doboš et al. 2013]. A later demonstration used X3DOM [Scully et al. 2015] as the transfer format. Denning and Pelacini [2013] presented MeshGit and [Doboš et al. 2014] developed systems for differencing and merging polygonal models. Some large-scale CAD systems provide similar functionality [Dassault Systemes; Bentley Systems]. The online collaborative tool Onshape uses a concept called microversions where changes in different clients can be merged at the individual action level.

In Open3D, we will take a locking approach rather than a merge later approach. The locking approach is more common in real-time distributed virtual environment systems, for example, see the CIAO concurrency scheme [Sung et al. 1999]. One issue with taking this approach to editing large scenes has been how to identify the lockable entity. On a shared file system a file might be marked as read only. This problem can be avoided in online systems, by having some naming scheme for individual meshes or mesh collections. This fine-grained naming allows for locks on objects small enough to avoid conflicts in many situations, see Section 4.

City models. In the specific area of geographics information systems, various database tools have been developed to store spatial information [Rigaux et al. 2001]. Oracle Spatial provides spatial indexing and search for geographic information [Oracle Corporation 2012]. However, this tool and similar tools tend to focus on large geographic 2D data rather than the processing of individual polygonal model edits. Open standard have emerged for the representation of geographic data such as OpenGIS [GeoTools] and CityGML [Open Geospatial Consortium], and these have a strong relationship to web 3d tools. However, they do not directly address the issues of versioning and collaborative editing. A desirable feature for future work on Open3D might be able to support such toolsets via import or export of regions of the models.

Many attempts have been made to generate virtual cities from existing GIS data. For example, a virtual model of London was generated from 2D plans and supported in a collaborative virtual world [Steed et al. 1999]. Recently Byelozyorov et al. [2013] demonstrated an open architecture to generate models from OpenStreetmap data. Gaillard et al. [2015] demonstrate generation of a web visualization of city models starting with CityGML data. Google Building Maker [Google Inc.] is a discontinued tool for city curation in Google Earth that has been replaced by an automatic reconstruction of building models from aerial imagery. Although no clear reasons were given for the decision, main concerns seem to have been the required amount of manpower and insufficient model quality. Results of the automatic reconstruction look impressive when textured, but the underlying mesh, while sufficient for simple map visualization, is quite noisy and inaccurate. With our parametric model, we aim to reduce the time required to model similar buildings while maintaining a high quality level.

The main problem with a browser-based visualization of city models is that there is a wide variety of capabilities of browsers with respect to rendering capacity, network capacity, storage, etc. Furthermore, city models tend to be extremely large. For example, later in this paper, we use the 3D Tiles capability of the Cesium system [Analytical Graphics, Inc.]. This represents datasets in a hierarchical manner, allowing client code in the browser to manage loading and unloading of assets at different resolutions. To sup-

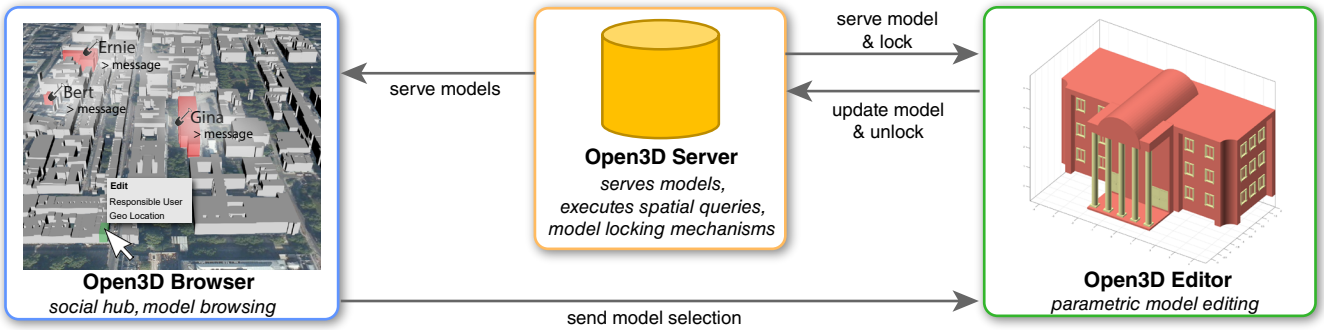

Figure 2: *Components of the Open3D System. The full scene can be viewed in the Open3D Browser (left). Analogous to a Wiki, this component also allows users to interact and coordinate editing efforts. After picking a model, editing is performed in the separate Open3D Editor (right). Both of these components communicate with the Open3D Server (center) to retrieve or update models. Editing conflicts are avoided through a locking mechanism.*

port the widest range of browser, some combination of data preprocessing and server-based rendering can be used. See, for example Instant3DHub [Behr et al. 2015]. Krämer and Gutbell [2015] evaluate some of the options for managing large urban models in current web 3d tools; Kim et al. [2015] evaluate some of the options for encoding geospatial data within X3D; while, Stein et al. [2014] investigate use of spatial acceleration data structure to enable larger models to be supported on the web.

Procedural modeling. In the context of large-scale 3D models, procedural model generation has a long and successful history (see survey [Smelik et al. 2014]). The recurring challenge, however, is that complex and detailed models typically require advanced procedural modeling operations [Schwarz and Müller 2015], which are difficult to author for most novice users. Hence, Open3D proposes a hybrid representation, wherein users can create detailed building models through high-level specifications and by reusing comparable edit histories from modeling sessions of similar buildings in a collaborative curation setup.

3 Overview

In collaborative editing systems, users work on separate parts of a larger structure. The core tasks performed by collaborators involve finding a part to work on and editing it, not unlike single-user systems. However, collaborative crowd-sourced systems have additional requirements: assets need to be kept consistent across all users, the editing process has to be simple and appealing to reach a large audience, data needs to be protected from vandalism, and a social platform needs to be provided to support a community.

The Open3D System consists of three main components connected over a web API: *server*, *browser*, and *editor* (see Figure 2). In a typical workflow, a collaborator browses the model while interacting with the community, selects an individual part of the model, edits the part, and returns to the browser when done.

The entire large-scale model is stored on the *server*, which is responsible for serving parts of the model as needed and merging updated model parts from the editor before pushing changes to all collaborators. It avoids conflicting edits by locking model parts.

The *browser* facilitates navigation of the large-scale model and selection of individual parts, gives an overview of contributor activity and acts as a social hub where contributors can communicate and coordinate modeling tasks. It provides a read-only view of the model that can be augmented with additional information. This can include an indication of models that are currently being edited, or models that have been modified recently.

The *editor* supports a hybrid between parametric and traditional polygon modeling. The idea is to combine intuitive traditional modeling with the efficiency of parametric models. During a traditional modeling session, the parametric model is automatically constructed or updated in the background and can be used to efficiently modify or generate variations of a model. This parametric model also acts as an edit history reaching back to the initial creation of the model. In case of model corruption, it can be used to restore an earlier model version. Details are described in Section 5.

4 The Open3D System

In current collaborative editing systems, browsing and editing can either be tightly coupled, as in Wikipedia, where editor and browser are part of the same web page; or more loosely connected, as in OpenStreetMap [OpenStreetMap Foundation], where editing is performed with external applications. Since editing 3D models usually requires a larger and more complex toolset than editing text, we opt for a decoupled architecture, where browser and editor are two separate components. This separation allows for a greater flexibility in the choice of editor: custom editors, as well plugins for established editors like Blender or Maya might be used.

Model formats and data flow. Open3D aims at supporting interoperability with custom browsers, editors, and servers. Therefore, data flow and communication between components has to be standardized. Below we describe the standards used for transmitting data between the components. See Figure 3 for an illustration.

Data flows mainly between server↔browser or server↔editor, with very little direct communication between browser and editor. The Open3D server delivers two types of model data: the geometry of individual model parts and scene metadata that describes the position of model parts in a geodetic reference system.

Scene metadata may be delivered as a flat list of part positions using the CZML format [Analytical Graphics, Inc.], or as a spatial hierarchy using Cesium 3DTiles [Analytical Graphics, Inc.]. As opposed to generic spatial databases, 3DTiles is specifically designed for displaying large city models, with heterogeneous level of detail and good support for interactivity, streaming and rendering. CZML was developed by the same group and is currently supported by the the same libraries that also support 3DTiles. Given such scene information, a client may perform a spatial query and only request model parts that are in view. When working with very large scenes, the server may also perform a spatial query and only return the parts of the scene metadata relevant for the current viewpoint of the browser, for example, metadata for all model parts within a given radius of the viewpoint. When using Cesium 3DTiles, different levels of detail may be returned as well.

Model part geometry is delivered in three different formats: glTF [Khronos Group], Collada [Barnes and Finch 2008] and our

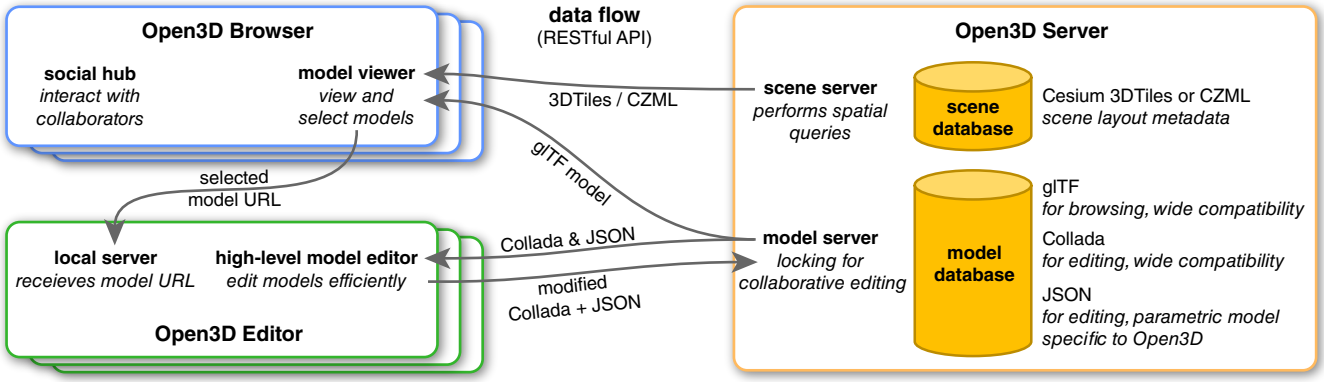

Figure 3: *Component details and data flow between components. Multiple heterogeneous browsers or editors may be connected to the same server. Data flow between components is implemented with a RESTful API that allows for easy interoperability with custom browsers, editors or servers.*

custom parametric format using JSON. Each format is suitable for a specific use case. glTF is an open model format that differs from other formats like X3D [Web3D Consortium 2008] or Collada in that it was specifically designed for efficient streaming and loading into an OpenGL renderer. This format is suitable for use in the browser on both PC and mobile platforms, since large amounts of model data can be transmitted and parsed efficiently. However, it is not well suited for editing, since information not essential for rendering is discarded to optimize file size. Collada is a format for 3D assets that is compatible with a large number of 3D editors. We use this format for transmission of models to and from the editor due its wide compatibility and completeness of features. Both glTF and Collada are managed by the Khronos Group and there exists good support for converting between the two formats. Our custom parametric format contains the information necessary to support parametric editing operations. In case of vandalism or corruption of the Collada file, it can be used to revert the model to a previous state. Each model has a unique identifier that is referenced in the scene metadata and that can be use to access the model.

Browser↔editor communication is limited to transmitting the model part selection from browser to editor, as described next.

Editing cycle. Here we describe the communication between server, browser, and editor, as well as the expected behavior of the server in more detail. All communication takes place over a RESTful API. Figure 5 shows the requests issued during a typical browsing/editing cycle.

Communication between browser and server is handled by repeating two request types: the first request retrieves scene metadata relevant for the current viewpoint from the server. The server issues a spatial query with the given viewpoint into the scene database to find and return the requested metadata. The second request retrieves the glTF model parts referenced in the scene metadata from the server. Model parts are always referenced by unique identifiers. Before issuing the request, the browser may remove identifiers of parts that are not in view or that are not likely to come into view.

At any time, the user may select a model part in the browser and start editing. The selected model part identifier is sent to the editor, which retrieves the associated Collada and parametric model files. On the server side, the given model identifier is marked as locked as soon as the Collada or JSON files are requested, preventing any other user from editing the model part. After editing, the altered model files are pushed back to the server. The server then updates the glTF file from the modified Collada file and unlocks the model. The user may then continue browsing for a new model part.

Locking. Keeping the shared model consistent and avoiding editing conflicts are important tasks of a collaborative editing system. We employ locking to ensure that only one user can edit a model part at any given time. The browser receives the current locked/unlocked status of a model when retrieving its glTF file. Models currently being edited are indicated by a red color in the viewer and requesting the Collada or parametric model files for one of these models returns an error. A timeout may be applied to avoid locking models permanently. See Section 8 for ideas on how to achieve more fine-grained locking using our parametric model.

Current implementation. As a research prototype, we implemented most components of the architecture described above. An overview of our current implementation is shown in Figure 4. Please also refer to the accompanying video. Both scene and model servers are implemented using Express [Node.js Foundation] running in the performant Node.js Javascript environment [Node.js Foundation], giving us platform independence and a greater flexibility to quickly test changes, which is the focus of our current implementation. Spatial queries and Cesium 3DTiles on the scene server are not fully supported in our prototype. Instead, we currently return the CZML metadata for the entire scene.

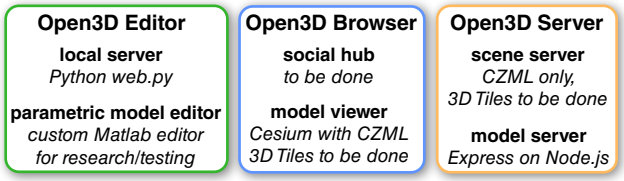

Figure 4: *Current implementation of each component.*

The model viewer is implemented in the Cesium javascript library [Analytical Graphics, Inc.], an open-source Javascript framework for streaming and visualizing three-dimensional geographic data. The viewer runs in a browser on any platform that supports WebGL. The social hub is not yet available and will be added in a future version of our system, see Section 8 for details.

The editor implements our new parametric editing model and we have opted to use a custom framework implemented in Matlab [MATLAB 2016] that facilitates quick cycles of implementing and testing new features. The local server that receives the selected model part URL and opens up the editor is implemented in Python using web.py and Matlab's Engine API for Python.

This set of components gives us an initial framework we can use to perform research and testing. We plan to expand this framework in the future to include the missing components, as well as to test the viability of possible future additions to the Open3D architecture, such as more fine-grained locking mechanisms. Our system will be opensourced as we hope others will collaborate and contribute to alternate browser/client plugins.

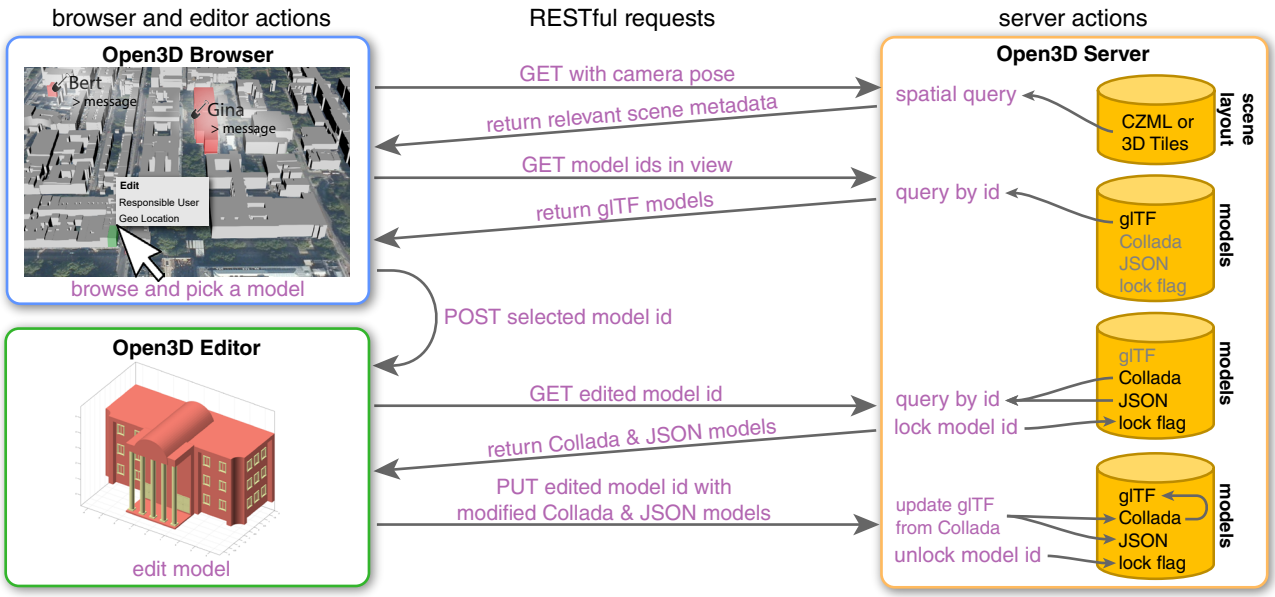

Figure 5: *Actions performed by our system during a typical browsing/editing cycle. To avoid conflicts, models are locked on the server while being edited.*

5 Parametric Model Representation

In urban modeling, there are two main approaches to modeling buildings: the traditional polygon- or mesh-based modeling approach used in software such as AutoCAD [AutoDesk, Inc.], and parametric modeling (e.g., procedural models) such as employed in CityEngine [Esri]. Traditional modeling is intuitive and creating a new building from scratch is relatively straight-forward. However, modifying an existing building or creating new variations of an existing building usually involves a lot of repetitive work, like rearranging all the details that were modeled on the building, such as windows, doors, etc. In procedural modeling, on the other hand, creating a new building from scratch is more involved. Expert knowledge is required to design such a parametric model and, similar to writing code, planning ahead is necessary to account for possible scenarios the parametric model might be used in. Modifying an existing building or creating new variations, on the other hand, can usually be efficiently done by adjusting only a few parameters.

Our collaborative approach favors simple modeling techniques that can reach a large community of contributors, while working with large cities requires an approach that allows for easy modification of buildings. To satisfy both requirements, we introduce a new modeling approach we call *implicit parametric modeling*. Initially, buildings are created in a traditional modeling session. During the modeling session, we build an initial parametric model in the background, based on the operations performed by the user. This initial model is a first rough guess at the parameters the user may want to modify and can be refined by the user as needed through a simple editing interface.

The parametric model for a simple building is shown in Figure 6. Individual operations performed during the editing session form the nodes of a directed acyclic graph, called the *operation graph* (see Appendix A for a list of currently implemented operations). Edges describe dependencies between operations, which define an order in which the operations must be performed: child nodes must always be applied after their parents. There are two types of edges. *Direct dependencies* exist from a node that creates or modifies a part of the scene to a node that uses this modified part as input. These are depicted as solid arrows in Figure 6. For example, the 'Copy' operation uses the output of the 'Extrude' operation as input (part B v2) and is therefore dependent on the 'Extrude' operation. *Implicit dependencies* exist from nodes that use a given part as input to nodes that modify this part. These dependencies prevent operations from changing scene parts before other operations get a chance to use them. An example is the 'Array Copy' operation, which must be performed before the 'Delete' operation that modifies part F v1. The parameters of our parametric model are then defined as the parameters of each operation in the graph.

Given the operation graph, the building can be edited efficiently by changing operation parameters. Examples of three edits are shown in Figure 6A, 6B, and 6C, where the rectangle width, circle radius, and extrusion height parameters are changed to modify the building width, column radius and building height, respectively. These edits would be more difficult in a traditional modeling approach, where individual vertices would need to be selected and modified, and windows and doors would need to be moved.

Retargeting operations. For some operations, there is ambiguity in how to retarget a given operation to a changed input. For example, retargeting the door placement shown in Figure 6A to a building with modified width is ambiguous: should the door remain at the center of the facade or rather maintain its distance from the left, or right edge of the facade? Depending on the situation, the user may want to use any of these retarget types. We implement several retarget types for each operation and set one as default. After changing a parameter of the model, the user may change the retarget type if needed by manually adjusting inappropriately retargeted scene parts. In Figure 6A, for example, by dragging the door to its intended position. Our system can then automatically select the retarget type that best matches the new position. Figure 6D shows the same result with a modified retarget type that maintains the distance to the right, instead of the left facade edge.

6 Importing GIS Data

Open3D makes it relatively easy to directly import GIS data and generate initial rough models based on this data. An overview of the steps performed by our method is shown in Figure 7. We extract building footprints and building heights from the GIS data and use them as parameters in a predefined parametric model consisting of two operations: creation of the footprint polygon using the footprint vertices as parameter and extrusion using the building height

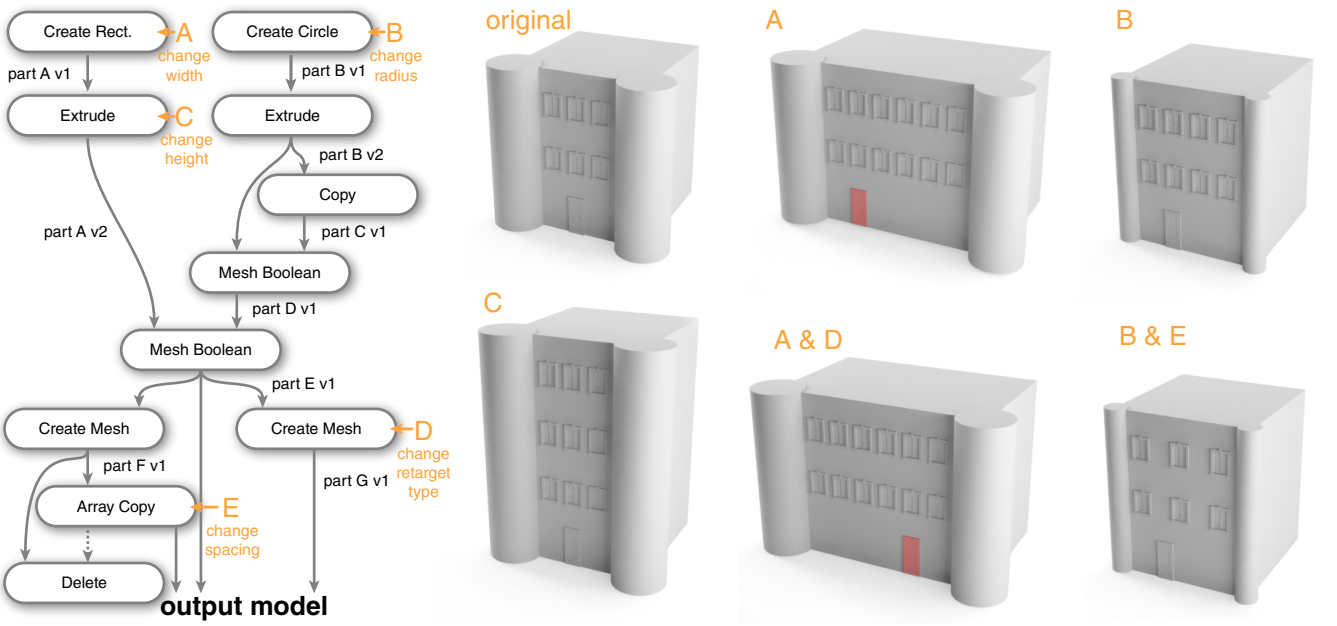

Figure 6: *The operation graph of a building model. Rounded rectangles denote operations and solid arrows input and output model parts. Dotted arrows illustrate implicit dependencies between operations. On the right we show the results of several modifications of the operations graph. Modifications that would require a lot of operations in traditional modeling can be done efficiently using our parametric model. Each variations was created with one or two modifications of the operations tree.*

as parameter. The parametric model is used to generate the initial models for all all imported buildings. To avoid numeric issues with vertex coordinates, we save vertices in the local coordinate system of each building and store the location of each building in a generated scene CZML file. The models are then added in glTF, Collada and parametric format to the Open3D server and the generated scene file is merged with the scene file on the server.

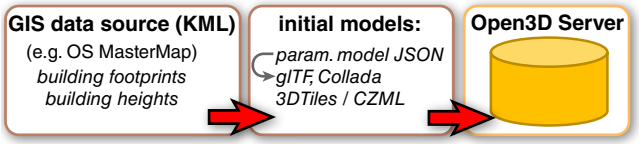

Figure 7: *Steps performed for importing models from a GIS data source. Building footprints and heights are obtained from the data source and used to automatically synthesize initial buildings using our parametric model.*

To accommodate additional data that might be available in a GIS source, for example the height of building roofs, we can extend the parameteric model used to generate the initial buildings with additional operations that use the added data as input. Both the definition of the parametric model and the binding of data to operation parameters is currently defined in a simple matlab script, however it would be straight-forward to create an import interface instead.

In our implementation, we currently support the CZML format, as well as the KML format, which is used by Google Earth, Google Maps and most of the major GIS data providers.

7 Examples and Discussion

In this section, we show several examples of buildings created with our implicit parametric modeling approach Three different building types modeled in our editor are shown in Figure 8. On the right of each building, we show variations created by modifying up to three parameters of the parametric model constructed implicitly in the background of the editing session. The exact number of operations performed for initial model creation and subsequent modification are shown below each example. Modifications include the width and height of the building mass model, the radius or thickness of details such as roof, pillars and small towers, and the layout of protrusions, windows and doors on the facades. In traditional mesh modeling, modifying properties such as the building height would require a large number of low-level adjustments. In addition to modifying individual window and door positions, parts of the facade would have to be re-modeled from scratch, since high-level information such as the cylindrical shape of the small towers may no longer be available in the merged mesh. Note that generating variations of buildings with different height or width is equivalent to propagating the parametric model to new buildings of different height or width, as shown in Figure 1.

This modeling approach is designed to scale with large city models and multiple collaborators. In typical cities, buildings with similar architectural style often appear multiple times throughout the city. Using our model, one collaborator can model an initial building in a traditional modeling session, and other collaborators can re-use the resulting parametric model to create buildings with similar style in just a few operations, refining the parametric model as needed by updating the re-target types of operations.

8 Conclusion

We presented Open3D, a system architecture for crowd-sourced distributed curation of large-scale city models. Open3D enables a community of collaborators to simultaneously edit a city model. Server-side locking mechanisms prevent editing conflicts, a web-based browser enables efficient inspection of the model on desktop and mobile devices, and a new modeling approach allows for intuitive and efficient curation (i.e., both creation and editing) of large-scale models.

Continuing the development of Open3D, we plan to include more fine-grained locking mechanisms on the level of operations in the parametric operation graph, as opposed to the level of building models. Multiple users could work on different aspects of the same

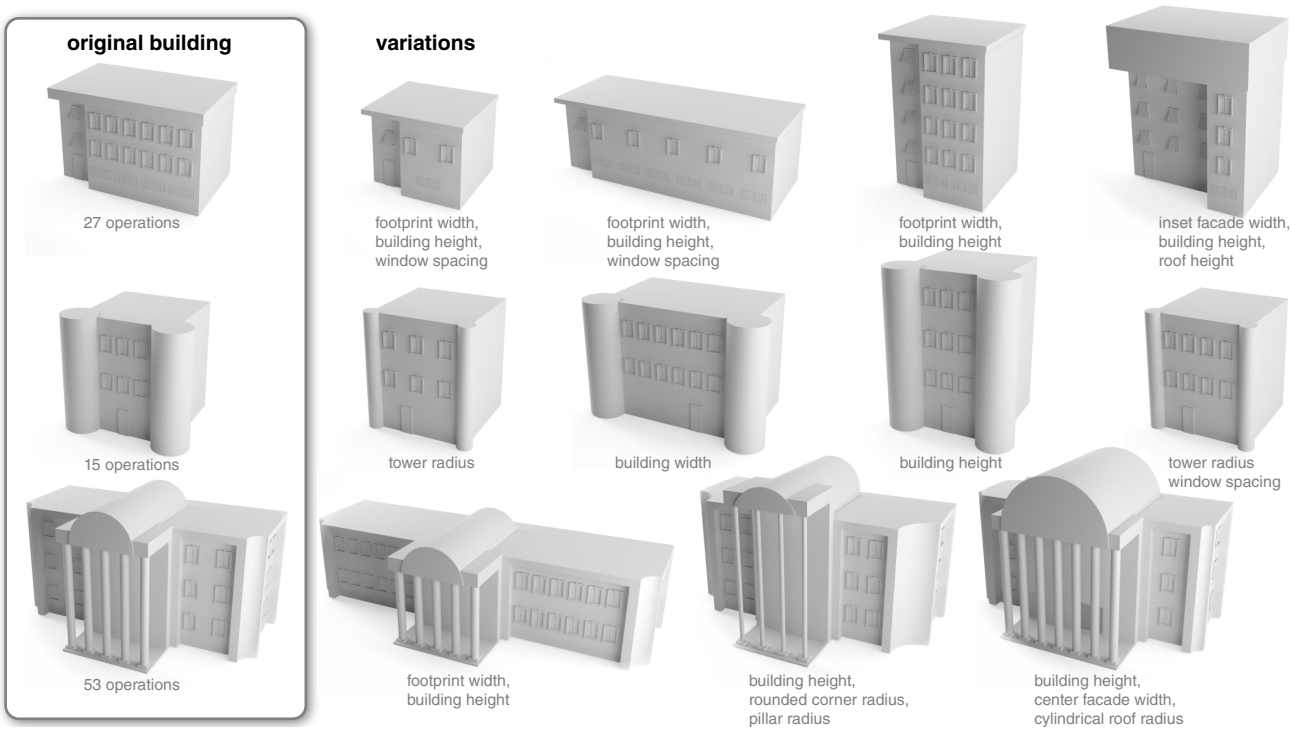

Figure 8: *Variations of building models created with our parametric modeling approach. The number of operations in the three parametric model is shown below the original buildings on the left. Each variation was created by modifying the parameters of three operations or less, which are given below each building. In traditional mesh modeling, editing the original models to create these variations would be significantly more difficult.*

model; one user might work on the shape of the mass model, while a second user modifies the shape of individual windows. Similar to a version control system, changes could be merged back by merging the modified operation trees. The parametric model also provides a compact representation of a model that could be used to efficiently store an incremental history of each scene model. This history could be used as protection against data corruption and vandalism. In the near term, we plan to finish the implementation of all components described in the Open3D architecture, including the social hub and spatial server queries. We would also like to add more operations to our parametric model, for a more streamlined modeling experience. With the social hub, we plan to give users a space to discuss models and required changes (e.g. with mark-up of model parts), and possibly post and accept gamified modeling tasks. Finally, we plan to extract a reference for the 3D shape and texture of each building from available street-level photographs (e.g., Google Street View), as additional assistance for building modeling.

Acknowledgements

We thank the anonymous reviewers for their feedback and Moos Hueting for help with the video. This work was supported by the Open3D Project (EPSRC Grant EP/M013685/1).

References

ANALYTICAL GRAPHICS, INC. Cesium. https://cesium.agi.com/. Accessed: 2016-04-14.

AUTODESK, INC. AutoCAD. http://www.autodesk.com/products/autocad. Accessed: 2016-04-15.

BARNES, M., AND FINCH, E. L. 2008. COLLADA - Digital Asset Schema Release 1.5.0. Specification, Khronos Group, April.

BEHR, J., ESCHLER, P., JUNG, Y., AND ZÖLLNER, M. 2009. X3dom: A dom-based html5/x3d integration model. In *ACM Web3D*, 127–135.

BEHR, J., MOUTON, C., PARFOURU, S., CHAMPEAU, J., JEULIN, C., THÖNER, M., STEIN, C., SCHMITT, M., LIMPER, M., DE SOUSA, M., FRANKE, T. A., AND VOSS, G. 2015. webVis/Instant3DHub: Visual computing as a service infrastructure to deliver adaptive, secure and scalable user centric data visualisation. In *ACM Web3D*, 39–47.

BENTLEY SYSTEMS. Assetwise. https://www.bentley.com/en/products/brands/assetwise. Accessed: 2016-04-14.

BYELOZYOROV, S., JOCHEM, R., PEGORARO, V., AND SLUSALLEK, P. 2013. From real cities to virtual worlds using an open modular architecture. *TVCJ 29*, 2, 141–153.

DASSAULT SYSTEMES. Enovia. http://www.3ds.com/products-services/enovia/solutions. Accessed: 2016-04-14.

DENNING, J. D., AND PELLACINI, F. 2013. Meshgit: Diffing and merging meshes for polygonal modeling. *ACM TOG 32*, 4 (July), 35:1–35:10.

DOBOŠ, J., AND STEED, A. 2012. 3d revision control framework. In *ACM Web3D*, 121–129.

DOBOŠ, J., SONS, K., RUBINSTEIN, D., SLUSALLEK, P., AND STEED, A. 2013. Xml3drepo: A rest api for version controlled 3d assets on the web. In *ACM Web3D*, 47–55.

DOBOŠ, J., MITRA, N. J., AND STEED, A. 2014. 3D timeline: Reverse engineering of a part-based provenance from consecutive 3d models. *CGF Eurographics*.

ESRI. CityEngine. http://www.esri.com/software/cityengine. Accessed: 2016-04-15.

EVANS, A., ROMEO, M., BAHREHMAND, A., AGENJO, J., AND BLAT, J. 2014. 3d graphics on the web: A survey. *Computers & Graphics 41*, 43–61.

GAILLARD, J., VIENNE, A., BAUME, R., PEDRINIS, F., PEYTAVIE, A., AND GESQUIÈRE, G. 2015. Urban data visualisation in a web browser. In *ACM Web3D*, 81–88.

GEOTOOLS. OpenGIS. http://docs.geotools.org/stable/userguide/library/opengis. Accessed: 2016-04-15.

GOOGLE INC. Google Building Maker. http://www.gearthblog.com/blog/archives/2013/03/google_will_be_discontinuing_the_bu.html. Accessed: 2016-05-27.

KHRONOS GROUP. GL Transmission Format (glTF). https://github.com/KhronosGroup/glTF. Accessed: 2016-04-15.

KIM, J.-S., POLYS, N., AND SFORZA, P. 2015. Preparing and evaluating geospatial data models using x3d encodings for web 3d geovisualization services. In *ACM Web3D*, 55–63.

KLEIN, F., SONS, K., JOHN, S., RUBINSTEIN, D., SLUSALLEK, P., AND BYELOZYOROV, S. 2012. Xflow: Declarative data processing for the web. In *ACM Web3D*, 37–45.

KRÄMER, M., AND GUTBELL, R. 2015. A case study on 3d geospatial applications in the web using state-of-the-art webgl frameworks. In *ACM Web3D*, 189–197.

KRONOS GROUP. WebGL. https://www.khronos.org/registry/webgl/specs/1.0/. Accessed: 2016-04-14.

MATLAB. 2016. *version 9.0 (R2016a)*. The MathWorks Inc.

MENS, T. 2002. A state-of-the-art survey on software merging. *IEEE TSE 28*, 5 (May), 449–462.

MICHAELIS, N., JUNG, Y., AND BEHR, J. 2012. Virtual heritage to go. In *ACM Web3D*, ACM, 113–116.

NODE.JS FOUNDATION. Node.js. https://nodejs.org/en/foundation/. Accessed: 2016-04-15.

ONSHAPE INC. Under the hood: How collaboration works in Onshape. https://www.onshape.com/cad-blog/under-the-hood-collaboration. Accessed: 2016-04-14.

OPEN GEOSPATIAL CONSORTIUM. Ogc city geography markup language (citygml) encoding standard, 2.0.0. http://www.opengeospatial.org/standards/citygml. Accessed: 2016-04-14.

OPENSTREETMAP FOUNDATION. OpenStreetMap editors. http://wiki.openstreetmap.org/wiki/Editors. Accessed: 2016-04-22.

ORACLE CORPORATION, 2012. Oracle spatial & oracle locator, location features for oracle database 11g, January. http://www.oracle.com/technetwork/database/options/spatial/overview/introduction/index.html.

PLESCH, A., AND MCCANN, M. 2015. The X3D geospatial component: X3DOM implementation of GeoOrigin, GeoLocation, GeoViewpoint, and GeoPositionInterpolator nodes. In *ACM Web3D*, 31–37.

RIGAUX, P., SCHOLL, M., AND VOISARD, A. 2001. *Spatial Databases: With Application to GIS*. Morgan Kaufmann.

SCHWARZ, M., AND MÜLLER, P. 2015. Advanced procedural modeling of architecture. *ACM TOG 34*, 4 (July), 107:1–107:12.

SCULLY, T., DOBOŠ, J., STURM, T., AND JUNG, Y. 2015. 3drepo.io: Building the next generation web3d repository with angularjs and x3dom. In *ACM Web3D*, 235–243.

SHARAKHOV, N., POLYS, N., AND SFORZA, P. 2013. Geospy: A web3d platform for geospatial visualization. In *ACM MapInteract*, 30–35.

SMELIK, R. M., TUTENEL, T., BIDARRA, R., AND BENES, B. 2014. A survey on procedural modelling for virtual worlds. *CGF 33*, 6, 31–50.

SONS, K., KLEIN, F., RUBINSTEIN, D., BYELOZYOROV, S., AND SLUSALLEK, P. 2010. Xml3d: Interactive 3d graphics for the web. In *ACM Web3D*, 175–184.

STEED, A., AND OLIVEIRA, M. 2009. *Networked Graphics: Building Networked Games and Virtual Environments*. Elsevier. ISBN-10: 0123744237.

STEED, A., FRÉCON, E., AVATARE, A., PEMBERTON, D., AND SMITH, G. 1999. The london travel demonstrator. In *ACM VRST*, 50–57.

STEIN, C., LIMPER, M., AND KUIJPER, A. 2014. Spatial data structures for accelerated 3d visibility computation to enable large model visualization on the web. In *ACM Web3D*, 53–61.

SUNG, U.-J., YANG, J.-H., AND WOHN, K.-Y. 1999. Concurrency control in CIAO. In *IEEE VR*, 22–28.

V. KOTHURI, R., GODFRIND, A., AND BEINAT, E. 2012. *Pro Oracle Spatial for Oracle Database 11g*. Apress Academic.

WEB3D CONSORTIUM, 2008. Extensible 3d (X3D). ISO/IEC 19775:200x, http://www.web3d.org/x3d/specifications/x3dspecification.html.

A Implemented Operations

The following operations are currently implemented in our prototype. New operations can be added to this list as needed.

- *CreateCircle*: parameterized by a center and radius, creates a circle polygon.
- *CreatePolygon*: parameterized by polygon vertices, creates an arbitrary polygon.
- *CreateRectangle*: parameterized by a center, width and height, creates an arbitrary rectangle.
- *CreateMesh*: parameterized by vertices and faces, creates an arbitrary mesh.
- *ChangePose*: parameterized by a change in pose (including change in position, orientation and scale), changes the pose by a delta relative to the current pose.
- *Copy*: parameterized by a target pose, copies the input to the given target pose.
- *ArrayCopy*: parameterized by element spacing and a target rectangle, creates and array of copies of the input in the given target rectangle.
- *Extrude*: parameterized by an extrusion height, extrudes the input by the given amount.
- *PolygonBoolean*: parameterized by the type of boolean operation, performs a boolean operation on two input polygons.
- *MeshBoolean*: parameterized by the type of boolean operation, performs a boolean operation on two input meshes.
- *Delete*: no parmeters, deletes the input scene parts.