

Generating Semantic 3D Models of Underground Infrastructure

Erick Mendez, Gerhard Schall, Sven Havemann,
Dieter Fellner, and Dieter Schmalstieg ■ *Graz University of Technology*

Sebastian Junghanns ■ *Grintec GmbH*

Large geospatial databases are populated with the results of hundreds of person-years of surveying effort. Utility workers access these databases during fieldwork to help them determine asset location. Real-time rendering engines are highly advanced and optimized software toolkits that interactively display 3D information to users. Bringing these two technologies together could give utility workers 3D information about a location's assets while they're in the field.

By combining semantic scene-graph markups with generative modeling, this framework retains semantic information late in the rendering pipeline. It can thus enhance visualization effects and interactive behavior without compromising interactive frame rates.

To connect geospatial databases and rendering engines, we must *transcode* raw 2D geospatial data into 3D models suitable for standard rendering engines. Thus, transcoding isn't simply a one-to-one conversion from one format to another; we obtain 3D models from 2D information through procedural 3D modeling. Transcoding the geospatial database information's semantic attributes into visual primitives entails information loss. We must therefore find the right point in the pipeline to perform transcoding. If we discard semantic information too early, we can't use it to interact with the user later in the pipeline. If we discard it too late, we have to reinterpret the semantics at runtime, which increases overhead and adversely affects performance. We call this the *transcoding trade-off*.

We've created a modeling framework that lets developers optimize this transcoding tradeoff. The framework transcodes geospatial data into 3D in-

teractive visualizations. It does this by combining a conventional scene-graph with semantic markup and on-the-fly generation of procedural models enhanced with an embedded stack-based scripting language. Because we tightly integrate these techniques, we can dynamically choose transcoding and representation methods for each object on the basis of the available high-level semantic information. Our approach also lets us define visualization styles in relation to the semantic markup, independent of actual object structures. To describe our technique, we use an augmented reality (AR) visualization of underground infrastructure, but our approach is generally applicable.

Design considerations

A 3D model for geospatial data visualization aims to provide comprehensible visualizations of the target assets. Because there are numerous geospatial objects and visualization styles, we need a system architecture that lets users add content types and visualization styles as plug-ins of the actual client. Ideally, a compatible 3D browser should be capable of loading and displaying self-contained content. The browser should also display user interfaces so that users can select and manipulate key parameters for their target application.

The following specific requirements apply to many visualization applications with complex data sets, regardless of their intended domain.

- **Appealing shape.** Creating appealing shapes requires advanced modeling methods. For example, branching pipe intersections must be continuous; we can't achieve this by simply

Related work in geospatial modeling

A procedural-modeling system's input parameters can be either artificial or derived from real-world measurements, such as survey or satellite images. Moreover, developers can use input to tightly control their generated models, using facades modeled from textures¹ or tangible modeling,² for example. Alternately, they can loosely control the models, using, for example, synthetic plants,³ buildings,⁴ or castles.⁵ Because our users expect a reliable real-world representation, we must provide strict dependence on real-world measurements and user-controlled parameters.

We generated our models using data exported from geospatial databases in the standard Geography Markup Language (GeographyML, www.opengeospatial.org). CityGML, for example, is a GeographyML specialization for 3D visualization of textured architectural models,⁶ but it requires a special browser. Instead, our work aims to use a standard scene-graph system. Geospatial data is typically exported via a Web Feature Service, which encodes the information in vector format inside a GeographyML instance. Other work exists on forwarding database information to scene-graphs—X-VRML,⁷ for example—but

such approaches generally don't involve on-the-fly procedural modeling.

References

1. P. Müller et al., "Image-Based Procedural Modeling of Facades," *ACM Trans. Graphics (Proc. Siggraph)*, 2007, article 85; <http://doi.acm.org/10.1145/1276377.1276484>.
2. D. Anderson et al., "Tangible Interaction + Graphical Interpretation: A New Approach to 3D Modeling," *ACM Trans. Graphics (Proc. Siggraph)*, 2000, pp. 393–402.
3. P. Prusinkiewicz and A. Lindenmayer, *The Algorithmic Beauty of Plants*, Springer, 1990.
4. P. Müller, "Procedural Modeling of Buildings," *ACM Trans. Graphics (Proc. Siggraph)*, 2006, pp. 614–623.
5. R. Berndt et al., "3D Modeling for Non-Expert Users with the Castle Construction Kit," *Proc. 6th Int'l Symp. Virtual Reality, Archaeology and Cultural Heritage (VAST)*, ACM Press, 2005, pp. 49–57.
6. T. Kolbe et al., "CityGML—Interoperable Access to 3D City Models," *Proc. Int'l Symp. Geoinformation for Disaster Management*, Springer, 2005, pp. 173–181.
7. K. Walczak and W. Cellary, "X-VRML for Advanced Virtual Reality Applications," *Computer*, vol. 36, no. 3, 2003, pp. 89–92.

converting raw geospatial database vectors into cylindrical tubes.

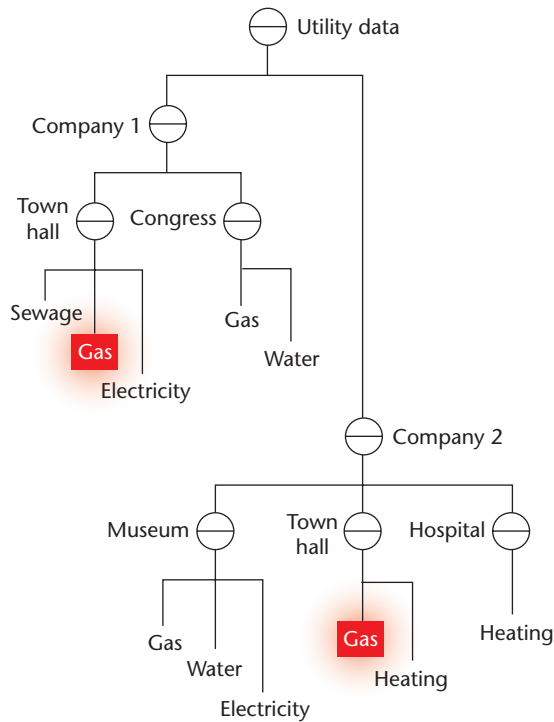
- **Level of detail (LOD).** Strict control of geometric complexity is essential, especially for low-performance mobile computers. The displayed content's geometric complexity should adapt dynamically and continuously (rather than popping) when the view changes.
- **Information filtering.** Displaying a large database's total content will likely produce screen clutter.¹ Users must be able to filter data using spatial and semantic information (such as object type). To achieve this, we use magic-lens techniques.²
- **Flexible styling.** Users might want to choose different styles for individually selected objects or object groups or to suit a particular viewing situation. With AR displays, for example, users might need to change the styling parameters to suit video see-through and registration quality. To ensure users can flexibly handle the styles, they should be stored with the content rather than offered as a feature of the specific 3D browser. Clearly, separating styles and content ensures that styles are reusable across content types.
- **Progressive information revealing.** Offering a semantic level of detail lets users efficiently manage screen real estate using multiple representations of the same object. These representations progressively reveal more visual and functional detail. For example, users can arrange objects in

containment hierarchies. This technique is useful in domains such as underground infrastructure, where cables are arranged inside encasings, which are contained in tubes, and so forth.

As the "Related work in geospatial modeling" sidebar describes, converting geospatial data directly into a static polygonal representation addresses only some of these requirements. And even a dynamic multiresolution tessellation won't help address the needs for interactive manipulation and control. Developers could customize 3D browsers to include these interactive capabilities, but doing so would make the content and browser highly interdependent and would defeat easy extensibility. Given this, we opted for a combination of techniques and implemented them as extensions to Coin3D, a conventional scene-graph library (www.coin3d.org), as part of our Studierstube framework. Our approach handles new object types through built-in interpretations of the scene-graph structure and doesn't require modifications of the scene-graph browser itself. We control asset modeling in two levels of the rendering pipeline:

- **Scene-graph level.** Using scene-graph markup lets us attach the geospatial database's semantic attributes. The markup also provides the hooks for interactive, high-level user control, including filtering, styling, and some semantic LOD control.

Figure 1. A scene-graph with semantic attribute markups. The town hall's reconstruction requires welding, which must not occur near gas. By combining the attributes (town hall + gas), the system automatically locates and highlights dangerous objects during a single scene-graph traversal.



- **Object level.** To provide high-quality tessellated objects and give users geometric and semantic LOD control, we added a new type of scene-graph node. The node has a lightweight embedded interpreter for the stack-based Generative Modeling Language (GenerativeML, see www.generative-modeling.org).³

During scene-graph traversal,⁴ Studierstube passes semantic attributes as parameters from the scene-graph to the GenerativeML nodes.

Transcoding

The first step in our information pipeline is a transcoding pass. That is, we change the data format from Geography Modeling Language encoding to a scene-graph enriched with GenerativeML nodes. GeographyML is based on the notion of *features*, including underground and above-surface infrastructure. A feature consists of several semantic attributes and many geometrical attributes that describe the actual 2D coordinates. Transcoding produces a scene-graph description that bundles each feature's shape objects and semantic attributes. The shape objects refer both to embedded GenerativeML scripts for dynamic parametric objects and to Coin3D classes for static nonprocedural shapes. The transcoding pipeline focuses on the underground infrastructure's common features.

The transcoding approach has an outstanding advantage. Traditionally, to highlight all objects of a certain type in a scene-graph—by changing their

color to red, for example—the system must somehow change the respective material properties of all affected nodes. It can do this by changing one of the node's material fields or by inserting material nodes into the scene-graph. However, changing just one “style node” for all target nodes early in the traversal order is a much better strategy. The scene-graph browser automatically propagates the changed parameters when traverses the graph for rendering; each affected node updates its styling because its attributes have been touched.⁵

We group objects in our scene-graph by not only the graph's hierarchy but also semantic-attribute families. A family's objects don't have to belong to the same part of the scene-graph hierarchy—as Figure 1 shows, they can be scattered throughout the scene-graph and still be jointly affected.

Additionally, we don't explicitly identify an object's family membership; rather, it results from an aggregation of semantic attributes that the scene-graph browser encounters while traversing the path from the scene-graph's root node to the object itself. This mechanism works exactly like other traversal states, such as transformation matrix or material bindings. Semantic-attribute nodes encountered during traversal add or overwrite arbitrary key-value pairs in the current attribute set. The scene-graph browser then propagates this set downwards with traversal, and any node aiming to be environmentally sensitive can query it.

More specifically, a styling node relies on a style map to modify appearance parameters for various shape nodes. Like XHTML's Cascading Style Sheets, our style maps are hierarchically organized. Style map entries are style subgraphs consisting of nodes that influence appearance; Studierstube dynamically inserts these nodes before the target object during scene-graph traversal. As a result, object appearance changes dynamically with the associated semantic attributes. This approach has four advantages:

- **Preserving semantics.** Our framework retains the geospatial database's semantic attributes until the actual rendering traversal, when it uses them to determine the object's appearance. Users can apply existing styles to new object types, assuming their attributes are compatible.
- **Referencing.** Potentially complex semantic attribute combinations can influence an object's appearance. When welding can't occur near a gas pipe, for example, the application may automatically select a style demarking “danger” if the user applies “gas” and “welding” attributes to an object group in spatial proximity.

- **Interactive styling.** The system can display selected semantic attributes in the user interface because the attributes are simply key-value strings. Users can directly modify attribute values, thereby influencing the visualization. For example, users can filter distracting objects by setting the object category's style to "off."
- **Parameter forwarding.** As we describe in the next section, Studierstube can forward semantic attributes as parameters to the GenerativeML engine to create the geometric primitives.

We encode geographical features as small subgraphs inside the scene-graph that contain both its semantic and geometrical attributes (see Figure 2). Semantic attributes add to, set, and modify key-value pairs in the traversal state. This gives any geometrical attribute—which is itself a subgraph containing a styling node and a content node—the ability to check for a mapping that matches its semantics. If such a mapping exists, the attribute modifies its styling node to affect the content node's appearance.

GenerativeML

To create parametric 3D models on the fly, we use GenerativeML. Originally, developers created GenerativeML to serve as a general exchange format for procedural models—for example, as a file format for encoding complex objects' construction history. GenerativeML can generate large amounts of geometric data from compact descriptions. Because its syntax is similar to PostScript, GenerativeML is somewhat like a 3D PostScript. Currently, GenerativeML's main surface representation is the combined boundary representation (cB-Rep)—polygonal meshes whose faces can be nonconvex and of arbitrary degree. cB-Rep is suitable for free-form modeling because edges have a crease attribute to distinguish sharp and smooth edges. In cB-Rep, faces containing smooth edges are rendered as Catmull/Clark subdivision surfaces, not as polygonal faces. GenerativeML's embedded OpenGL renderer is highly optimized and provides on-the-fly tessellation and LOD.

Generative modeling replaces objects with operations. Consequently, we represent the cB-Rep meshes not as lists of geometric primitives but as the product of Euler operators. We issue said operators either directly as GenerativeML operators or through higher-level modeling operations, such as extrusion.

The GenerativeML interpreter and its integrated OpenGL renderer reside in a shared library; users can link the library to any OpenGL application, such as a scene-graph engine. For example, devel-

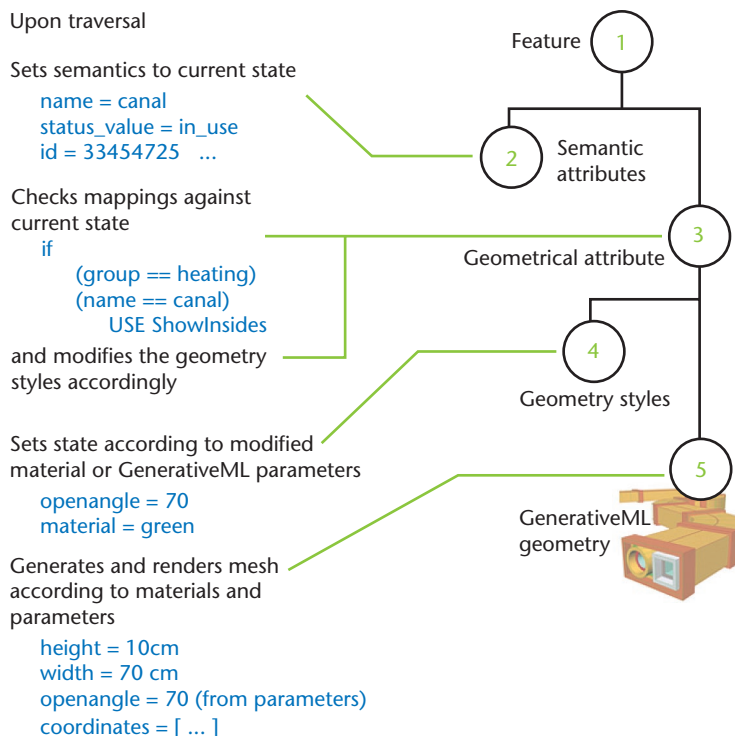


Figure 2. Traversing geographic features. Upon traversal, every geographical feature aggregates its semantic attributes to the traversal state, and checks whether the current total of the semantic values match a style mapping. If so, it fetches predefined styles accordingly. So, the shape node's rendering style reflects its semantic attributes.

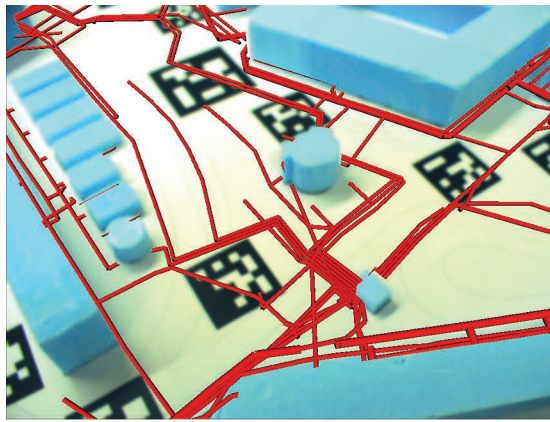
opers have embedded GenerativeML into Coin3D as a new node that communicates with the GenerativeML engine. We store and maintain script code in the GenerativeML scene-graph node's string field, located in the .iv file. Users can execute this code during traversal to produce 3D models of pipes and tubes on the fly. This greatly reduces scene-graph size and lets users vary construction parameters to reflect semantic-level changes, such as highlighting or database queries.

Generative modeling complements a scene-graph with semantic attributes. A scene-graph's strength is in managing large scenes by providing tools such as a scene hierarchy, spatial structuring, and flexible scene traversal. In contrast, GenerativeML operates on the object level. Its purpose is to use the rules and parameters to create the geometry. The interface between the software components is simply the set of semantic attributes that Studierstube passes as model parameters to GenerativeML.

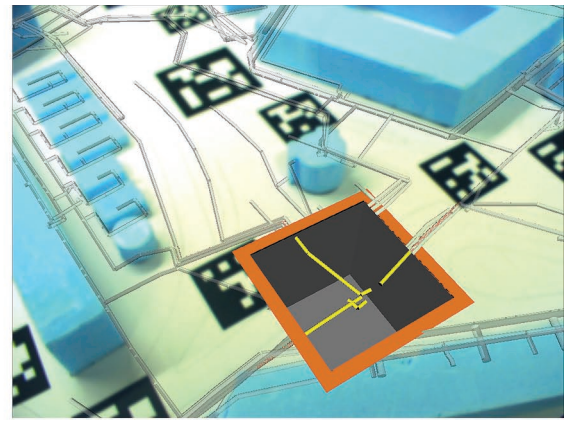
Visualization with filtering and annotations

To filter information, our system directly exploits semantic markup during rendering. Users can use information such as object categories, ownership,

Figure 3. Filtering information using a magic lens. Propagating semantic information lets users combine spatial with semantic filtering. To reduce screen clutter in (a) a cluttered image, we (b) apply an “excavation” magic lens that shows only selected assets. The surrounding area shows a toned-down version of all assets. We produced both images using a tabletop augmented reality (AR) setup with real architectural scale models (light blue).



(a)



(b)

and topological relationships to hide or highlight individual objects. Users often want to apply filtering to spatially bounded areas. To address this, we use magic lenses² that change object presentation styles depending on not only their containment inside the lens but also their semantic markup.⁵ Users can color-code objects or make them transparent inside the lens; alternately, they can change the parameters passed to GenerativeML—such as to open up the lens area’s pipes.

Figure 3 shows a magic lens designed to resemble an excavation hole. To create this, we reduced the object saturation and opacity outside the lens. Inside the lens, we displayed only objects with certain semantic attributes. This approach lets users localize the visual-disambiguation region, while retaining the surrounding assets’ spatial context.

An underground infrastructure application

To illustrate how our system works, we’ll use an example task from the public-utility sector. To avoid damaging existing infrastructure during digging, utility companies must provide private contractors—such as construction companies—with spatial information about their assets. To accommodate this need, utility company field-workers spray paint spots on the ground where digging should occur. Private contractors orient their construction site on the basis of paper maps plotted in advance using the utility company’s geographic information systems (GISs).

The GIS information model

Utility infrastructure—such as underground water or gas distribution systems—is arranged in canals, divided into multiple tunnels at different depths. Depth is either given as an attribute value or estimated on the basis of heuristics (telecommunications, for example, are typically in the first layer at 0.5 m depth). The depth that workers bury each utility system also varies depending on terrain conditions.

The central object in the transcoding pipeline is the *pipe*. All types of pipe-shaped infrastructure—whether electricity, gas, water, sewer, or heating pipes—are abstracted by the same common geometric attribute. However, supported by semantic attributes, we can visually discriminate among them.

One pipe can consist of several segments. To avoid creating excess polygons, transcoding automatically deletes collinear and duplicate points. We can also quantize coordinates for LOD generation, because users typically don’t require millimeter-level precision in their visualizations. We convert nongeometric attributes, such as ID, purpose, or ownership, to semantic nodes inserted on the scene-graph and pass geometric attributes to the GenerativeML nodes. We describe tunnels analogously to pipes, but use rectangular rather than circular cross-sections.

Pipes are in fact hierarchical: pipe objects are typically enclosed by other pipe objects, such as electricity lines or gas pipes arranged inside tunnels. Rather than simply deriving the containment hierarchy geometrically, we describe it through semantic attributes so that users avoid rendering subpipes when the containing pipe is set to “opaque.” GenerativeML’s procedural logic lets users open tunnels to reveal the interiors; thereafter, they can render the contained pipes as well.

In addition to pipes, the underground infrastructure also consists of special facilities such as gate valves, water hydrants, and T-fittings. We blend these facilities with the 3D environment on the basis of special rules, using a library of premodeled 3D objects. To provide geographic context, we include buildings, generating them procedurally from their transcoding-step footprints through simple extrusion, using a semantic height attribute (or a default value if this attribute is unavailable).

Adaptive asset modeling

As Figure 4 shows, we use a small set of parameters to procedurally create all types of pipe-like objects.

These include the polyline determining the pipe's overall shape, radius, thickness, and type; the radius of curved pipe segments, along with their collar radius, thickness, and so on.

Our basic problem is to connect two straight pipe segments with a curved pipe. The pipe's curved portion is a circle segment that starts at the dashed line before the actual joint, located at a given normal distance from the pipe axis. We compute the circle's midpoint as the intersection of the two dashed lines. Then, we sample the circle segment and produce radial line segments, along which we create a vertical profile of choice to create a rotational sweep. The result is a cB-Rep control mesh that we use to produce polygonal facets and Catmull/Clark subdivision surfaces. A pipe's first and last faces have sharp (red) edges that produce a crease, resulting in flat faces with a B-spline boundary. Bad parameter sets—such as those caused by bad geospatial data surveys—can lead to control mesh self-intersections (that is, pipes that are too wide, a curve radius that's too small, or a segment angle that's too acute). GenerativeML has no automatic mechanism to determine the parameters' valid range.

GenerativeML's power is that it efficiently supports processing chains. The stack lets the system flexibly pass one function's data as input parameters to the next function. We exploit this to create different pipe types by simply passing different profiles to a connecting function. Figures 5 and 6 show (next page) canals created from a rectangular profile with a user-defined, interactively controlled opening angle. For open pipes or canals, the profile—and hence the vertex count—is variable, depending on the opening angle.

A minor complication is that subpipes often run next to each other. Although such “offset pipes” seem to run in parallel, when they curve, they share the same curve midpoint but have different radii depending on whether they curve left or right. We can elegantly solve this problem by reusing the pipe-creation procedure, inputting the main poly-line offset as another control parameter. This lets us uniformly deal with hierarchical groups of pipes and canals that follow the same route. Specifically, to create the geometry for a cut-open hierarchical grouping, we need only propagate the opening angle.

As Figure 6 shows, collars around the pipe endings are important cues that help users understand spatial relationships, letting them join different pipe types while preserving a consistent visual appearance. Figure 7 shows how we dynamically generate per-object geometric LOD per frame. We do this by adjusting the tessellation's subdivision depth, depending on the covered pixel-area and

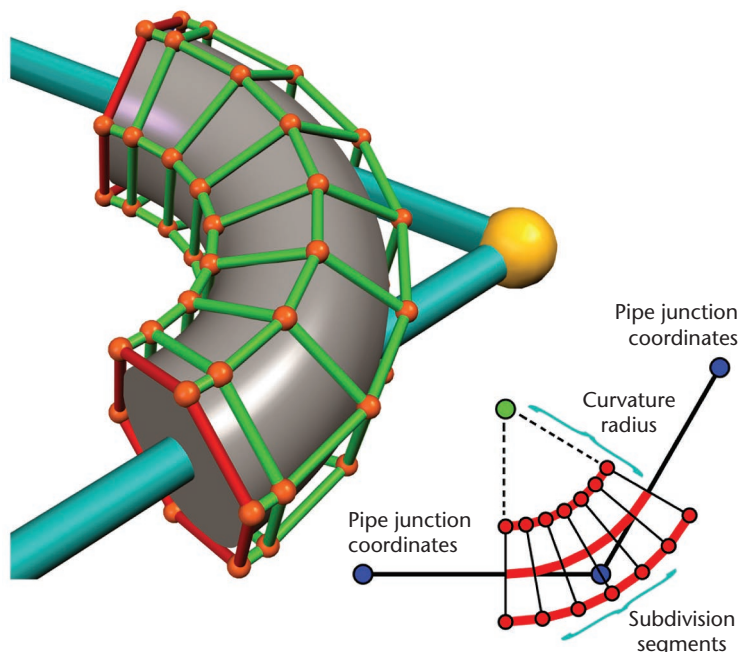


Figure 4. Modeling pipes using parameters. To achieve rounded edges, we compute a circle segment to connect subsequent tubes. We offset and sample circle segments on either side to obtain radial line segments (inset). We then convert the circular profiles (n-gons) to double-sided faces and connect them.

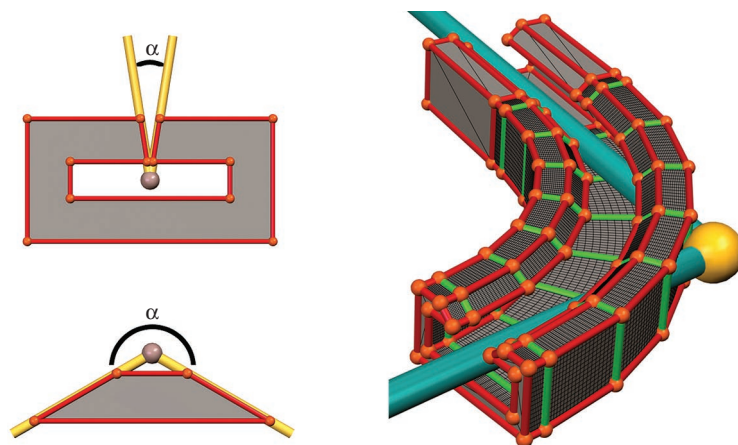


Figure 5. Creating open profiles. We can smoothly vary the opening angle and even animate it. To open up any object, we connect two sampled circle segments with different radii. In this case, we use “canals” with a rectangular profile.

the surface orientation and curvature. The interactive LOD lets us generate more visually pleasing images. For example, it produces pipe junction intersections as sharp connections in the geospatial data. GenerativeML models can soften these junctions by further subdividing the resulting mesh.

Finally, Figure 8 shows an example that combines modeling techniques. We modeled the pipes and the T-junction using GenerativeML; the smooth joint makes the connection clearly visible. The

Figure 6. Illustrating curved joints. We add collars to better distinguish curved joints between straight pipe segments. We can also arrange pipes, canals, and tunnels hierarchically.

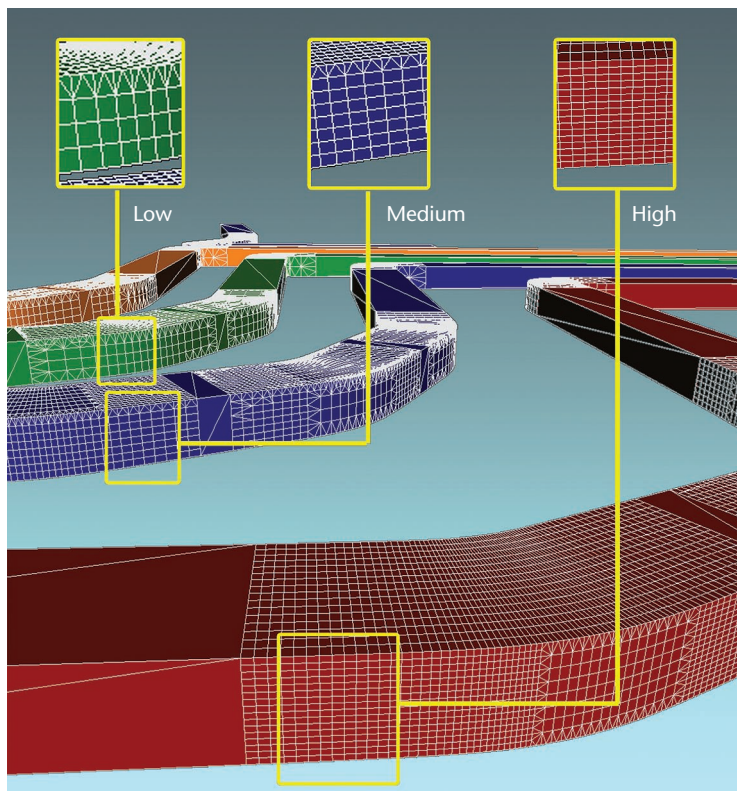
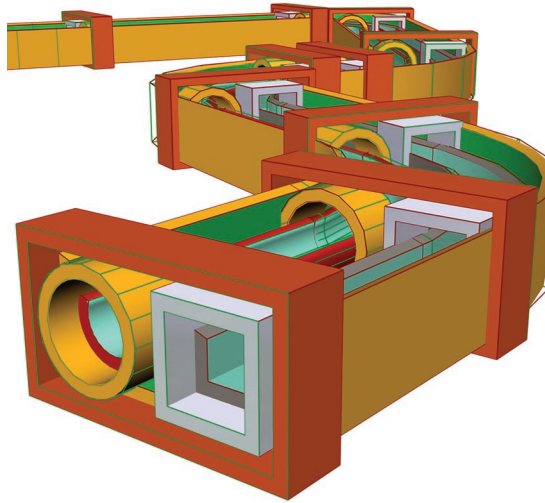


Figure 7. Interactive level of detail. Using interactive LOD lets us increase visual fidelity while minimizing the performance impact. The system automatically handles LOD selection when GenerativML code is called.

white objects are predefined shape library models (used at low resolution to avoid impairing rendering performance). We then applied transcoding to extract all models' geometrical position—as well as their orientation and semantic attributes—from the geospatial database.

AR-style visualization

We now focus on a practical solution for our public-utility scenario. Assuming a database can be accessed by a Web Feature Service, our construc-

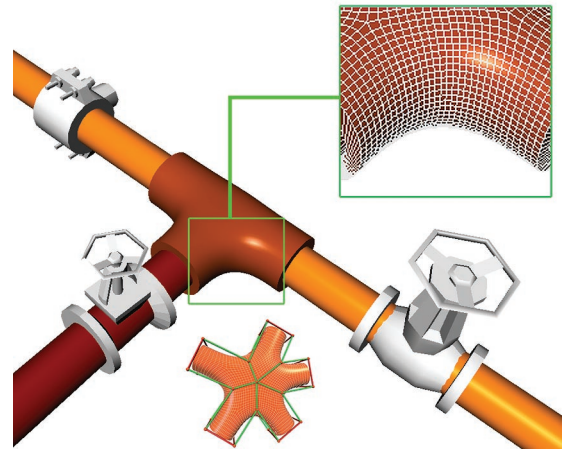


Figure 8. Combining modeling techniques. This image is a composite model of a GenerativML-created T-junction with valve models from a stock library. By using subdivision surfaces, GenerativML can create complex junction shapes such as the inset's five-way junction (lower middle).

tion-site field-workers can perform online queries using a wireless mobile computer that sends the current GPS position. The server returns a set of geographical features encoded in GeographyML. Currently, field-workers use handheld computers that can display 2D map information that's more or less equivalent to conventional paper maps.

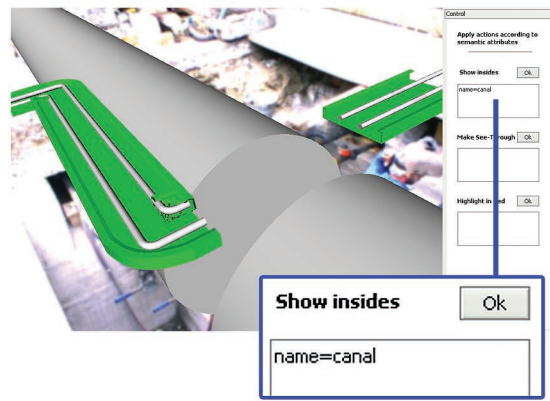
Because we can't acquire the position of pipes, ducts, valves, alignments, and other underground infrastructure from aerial or terrestrial photography, we rely exclusively on data from the utility company's GIS. We get above-surface information such as building footprints, roads, and property lines directly from the geospatial database. Although we focus on delivering the final visualization as part of a video see-through AR display, our visualization techniques are also applicable to desktop virtual reality applications.

Researchers have used our outdoor AR System as a tourist navigation aid.⁶ Here, we focus on our newly developed modeling techniques to visualize underground models. Figures 9 and 10 show geospatial data superimposed in real time on a video feed using the AR interface. Figure 10 also shows a user with a robust handheld AR device used for the outdoor work. We designed the AR interface around a handheld tablet PC. The PC is equipped with a sensor array consisting of GPS, an inertial orientation tracker, and a camera to show a video see-through augmentation of the environment.

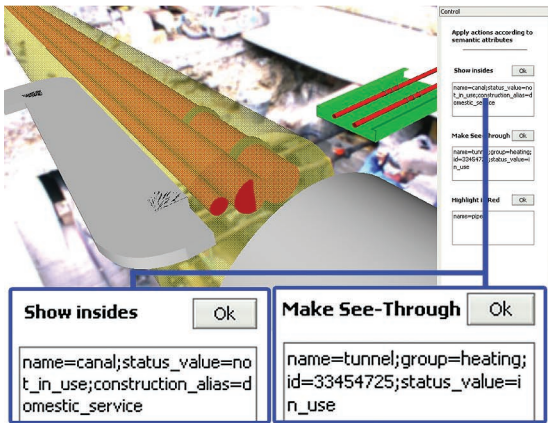
Styling objects using their semantic markup is a powerful tool when users can't control the data's object arrangements. Take, for example, Figure 9's scenario. The visualized information represents heating assets, organized in tunnels and canals that



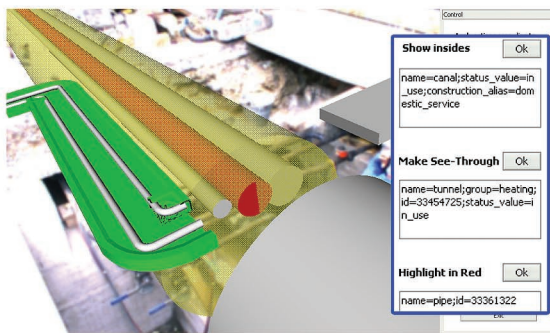
(a)



(b)



(c)



(d)

Figure 9. Visualizing the superimposed geospatial underground model. (a) In this photo of a construction site, the district heating-pipe infrastructure’s supply and return pipes are clearly visible. The technician’s task is to repair some of these pipes. (b) The technician requests that all objects of type canal (in green) show their interiors. (c) The technician tells the system to show the insides of the canal and the tunnel used for heating. (d) The technician asks the system to highlight the object with a specific ID.

contain pipes. A simple task is to request that all objects of type canal show their interiors. This action sets the GenerativeML parameter opening angle to 70 degrees only on those objects with matching semantic attributes—that is, on canals. Not all the objects have opening angles; those that do are set to green (see Figure 9b). This action reveals the pipes inside the green conduct, which is cut open. Alternatively, we could change the objects’ transparency to show their interiors as illustrated (Figure 9c). Typically, field-workers refine their search using additional attributes, such as objects that are in use or those that belong only to a certain category. In Figure 9c, the user would like to open only those tunnels in use for heating.

Our system also lets users assign different styles to objects. The styles, created by an application designer, can convey subjective meanings, such as “dangerous” or “important.” In Figure 9d, the user instructs the system to highlight an object with a specific ID that’s targeted for repair.

AR-style visualizations have two key benefits.

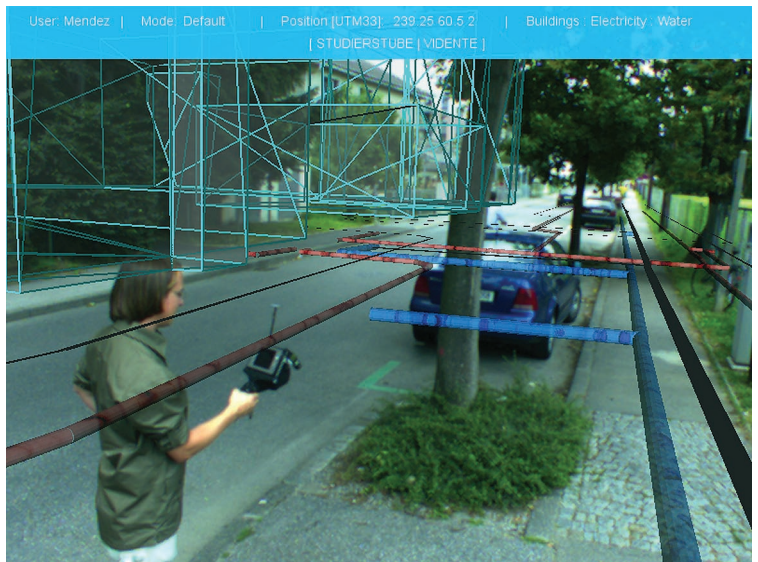


Figure 10. The AR display on a handheld device. The view shows a second AR user with a handheld AR display. The left side of the frame shows wireframed building models that help retain spatial context.

First, users can locate and visualize underground infrastructure before excavation (here, for the sake of illustration, it was done after). Essentially, our approach gives users “X-ray vision” of the buried assets, which can help them locate damage or plan construction, for example.

Second, our approach offers users visual guidance, both during excavation activities and in examining technical infrastructure. Users can retrieve on-demand information about pipe properties—such as purpose, specification, length, and material, as well as the construction date, maintenance companies, and so on. Technicians no longer need to contact the company’s back office or consult printed documentation when they need information related to the pipes or the construction site. Our data-transcoding technique lets us store all pipe properties in the corresponding source geospatial database, and all are available for information visualization. Consequently, our system offers intuitive on-site assistance.

Discussion

To study the transcoding trade-off effects, we performed a series of tests involving different pipe networks and mesh sizes for three separate transcoding stages:

- *S0: Static.* The transcoding ignored semantic attributes; a single mesh held all generated 3D objects.
- *S1: Adaptive on transcoding.* The transcoding evaluated semantic attributes, generating material values for each geospatial feature before deploying it to the 3D browser.
- *S2: Adaptive on traversal.* The transcoding preserved and evaluated semantic attributes during scene-graph traversal, where template mapping and material bindings occur.

On average, the S1 condition performed at 64 percent compared to S0 ($\sigma = 1.052931$), while S2 performed at 57.5 percent compared to S0 ($\sigma = 1.55204$). The tests indicate that the overall S1 and S2 performance will remain consistent regardless of the mesh’s size and will depend mainly on the object separation in subgraphs. As expected, S1 and S2 performed at similar rates. (The 6.5 percent performance difference is because, while the number of traversals is the same, S2 has more overhead owing to template mapping during traversal.)

Our system has a few limitations. Its information flow is strictly one-way, from the scene-graph to the GenerativeML nodes that generate the geometry procedurally. The scene-graph itself is static: at runtime, no nodes are added or deleted; only the connections

between these nodes (or subgraphs) can be changed. We could obtain more flexibility if it was possible to create parts of the scene-graph procedurally.


For large networks, the scene-graph should be able to load progressively. Then the system could refine nodes by inserting a subgraph or by collapsing subgraphs into a single node on the basis of proximity, visibility, and semantic queries.

A technical limitation is that Coin immediately renders each object that it encounters during traversal. Because a pipe has several different materials, many material state changes would occur. This prevents rendering optimizations, such as collecting from different objects all faces made of the same material.

Finally, the semantic markup lets users style objects during the traversal itself without prior knowledge of their graph position. This can cause caching problems, because every subgraph must be reformatted to reflect its semantic style mappings upon traversal.

Although our system is still in development, preliminary feedback from utility-sector users is encouraging. We believe that the approach—retaining semantic information and using it both for textual annotation and for influencing 3D shapes and visualization styles—has wide applicability in modeling human-made structures from existing legacy data.

As our tests show, deciding how much information to reformat during transcoding implies a trade-off between performance and flexibility. Preserving semantic information down to the traversal stage dramatically increases flexibility, letting users apply visual and modeling changes to the scene’s objects. But it also decreases performance: the number of traversals increases because every node adapts to reflect its semantic mapping by traversing a styling node before the geometrical content. A better strategy is to consider particular user tasks—as in our infrastructure maintenance example—thereby reducing the number of semantic attributes for the transcoding pipeline to preserve.

Our future research will involve hardening the system for field use and establishing full editing capabilities so that AR system field users can feed-back changes they observe into the geospatial database. 

Acknowledgments

Our work was sponsored by Österreichische Forschungsförderungsgesellschaft contract Bridge 811000

and Austrian Science Fund FWF contracts Y193 and W1209-N15. We also thank Grintec GmbH for providing geospatial data.

References

1. T. Höllerer et al., "User Interface Management Techniques for Collaborative Mobile Augmented Reality," *Computers and Graphics*, vol. 25, no. 5, 2001, pp. 799–810.
2. E. Bier et al., "Toolglass and Magic Lenses: The See-Through Interface," *ACM Trans. Graphics (Proc. Siggraph)*, 1993, pp. 73–80.
3. S. Havemann, *Generative Mesh Modeling*, PhD thesis, Department of Computer Science, Braunschweig Tech. Univ., 2005.
4. P. Strauss and R. Carey. "An Object-Oriented 3D Graphics Toolkit," *Proc. Int'l Conf. Computer Graphics and Interactive Techniques (Siggraph)*, ACM Press, 1992, pp. 341–349.
5. E. Mendez et al., "Interactive Context-Driven Visualization Tools for Augmented Reality," *Int'l Symp. Mixed and Augmented Reality (ISMAR)*, IEEE CS Press, 2006, pp. 209–218.
6. D. Schmalstieg et al., "Managing Complex Augmented Reality Models," *IEEE Computer Graphics and Applications*, vol. 27, no. 4, 2007, pp. 48–57.



Erick Mendez is a research engineer and PhD student at the Graz University of Technology. His research interests include context information and visualization techniques for augmented reality systems. Mendez has an MSc in computer science from The George Washington University. Contact him at mendez@icg.tugraz.at.



Gerhard Schall is a research engineer and PhD student at the Graz University of Technology. His research interests include mobile augmented reality and ubiquitous and pervasive computing. Schall received a Dipl.-Ing. in telematics from the Graz University of Technology. Contact him at schall@icg.tugraz.at.



Sven Havemann is an assistant professor at the Graz University of Technology. His research interest is in developing novel shape representations for describing 3D objects using generative modeling. Havemann received a PhD in computer

science from Braunschweig Technical University. Contact him at s.havemann@cgvtugraz.at.



Sebastian Junghanns is a software engineer at Grintec GmbH and a PhD student/external researcher at the Graz University of Technology. His research interests are in mobile and Web-based geospatial systems in the utility asset management domain. Junghanns received an MS in geoinformatics from Salzburg University. Contact him at sebastian.junghanns@grintec.com.



Dieter Fellner is a professor of computer science at the Darmstadt Technical University and the Graz University of Technology. He also leads Fraunhofer IGD in Darmstadt. His research interests include formal languages, telematics services, user interface design, software engineering, computer graphics, and digital libraries. Fellner received a Doctorate and Habilitation in technical mathematics from the Graz University of Technology. Contact him at d.fellner@cgvtugraz.at.



Dieter Schmalstieg is a professor of virtual reality and computer graphics at the Graz University of Technology, where he directs the Studierstube research project. His research interests include augmented reality, virtual reality, distributed graphics, 3D user interfaces, and ubiquitous computing. Schmalstieg received a Doctorate of Technology in computer science from the Vienna University of Technology. He's an editorial advisory board member on *Computers & Graphics*, a member of IEEE ISMAR's steering committee, chair of the Eurographics Working Group on Virtual Environments, and advisor of the K-Plus Competence Center for Virtual Reality and Visualization in Vienna. Contact him at schmalstieg@icg.tugraz.at.

For further information on this or any other computing topic, please visit our Digital Library at <http://www.computer.org/csdl>.