CEIG - Spanish Computer Graphics Conference (2015)
Jorge Lopez-Moreno and Mateu Sbert (Editors)

Industrial Facility Modeling Using Procedural Methods

M. S. Bishop[1] and N. Max[1]

[1]College of Engineering
Department of Computer Science
University of California, Davis, USA

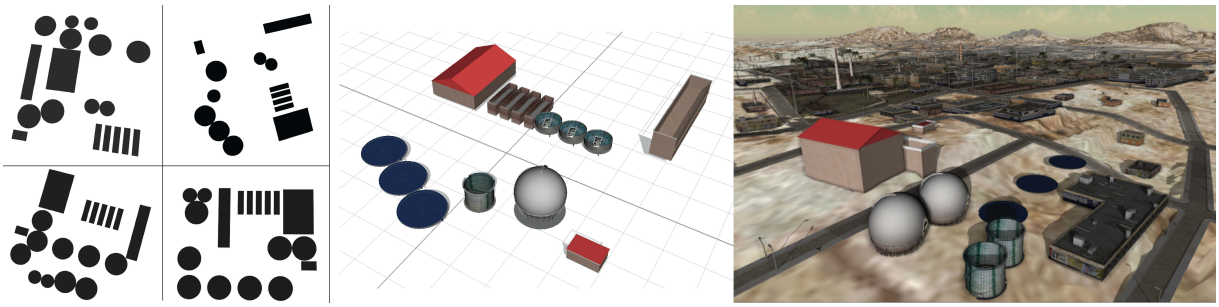


Figure 1: Industrial facility footprints (left) and procedurally modeled facility in CityEngine's desert scene [ESR] (right).

Abstract

We present an end-to-end system for procedurally modeling an industrial facility. The system is a collection of utilities that work together to assemble, lay out, and model a typical industrial facility (e.g. a wastewater treatment plant). A plug-in to the CityEngine® procedural modeling application was built in Java ™ using an open-source framework. The plug-in provides the interface to access the facility assembly and layout engines, the facility rule file and Python script generators, and the OBJ footprint exporter. The system provides functionality for placing the facility model into an existing 3D scene using an established facility location algorithm that maximizes the minimum distance from existing structures in the scene.

Categories and Subject Descriptors (according to ACM CCS): I.3.2 [Computer Graphics]: Graphics Systems—Stand-alone systems I.3.8 [Computer Graphics]: Applications—Procedural modeling

1. Introduction

Generating 3D structures using procedural methods is relatively straightforward for stand-alone buildings [MSK10, MWH*06] and structures with repetitive geometries [EMP*03, MM11] because unconnected geometric footprint shapes (rectangles, circles, polygons, etc.) can be crafted by hand or imported from existing databases like OpenStreetMaps [HW08, Ope11].

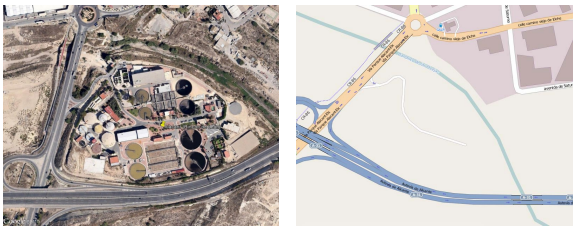
In contrast, footprints (Figure 1 (left)) for industrial facility sites like wastewater treatment plants (WWTPs) may entail complex inter-relationship dependencies between building structures that are generally simplified or non-existent in current databases (Figure 2). To address these challenges we developed a system to create facility models that use an industrial facility footprint which is faithful to the industrial process being modeled. We use the WWTP process flow diagram which begins at pre-treatment and ends at disinfection (chlorination, ultraviolet radiation, etc.) to assign adjacent facility components as nodes in a connected graph.

One of the most challenging aspects in developing a 3D modeling system for a specific type of industry is in acquir-

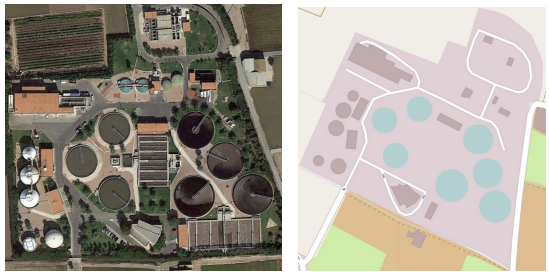
© The Eurographics Association 2015.

DOI: 10.2312/ceig.20151205

delivered by EUROGRAPHICS DIGITAL LIBRARY www.eg.org diglib.eg.org



(a) Aerial image of a WWTP. (b) Missing footprint of Fig. 2a.



(c) Aerial image of a WWTP. (d) Footprint for Fig. 2c.

Figure 2: Two aerial images [Goo15a] of WWTPs (Figures 2a and 2c) and their respective OpenStreetMap [Ope11] footprints (Figures 2b and 2d).

ing the domain knowledge about the use of the connected and unconnected components in the facility design. Therefore, we rely on the data and methods described in Bishop et al. [BFM14] for selecting facility components of WWTPs using a graphical model and arranging facility components using a physics based simulation and mathematical optimization.

In this paper, we present topics beyond our previous work [BFM14] that include:

- using geometric details about the modeled facility by learning the size of certain components in the facility from a hand labeled data set,
- using a force-directed layout algorithm to generate the initial configuration to the physics based layout engine, and
- creating a hybrid layout scheme that combines the output of the linear program optimization with the physics based layout.

Additionally, we describe the details for implementing an Eclipse plug-in that we use to create a facility modeling tool which runs inside CityEngine.

2. Related Work

A comprehensive survey of procedural modeling by Smelik et al. [STBB14] provides useful insights and categories of procedural methods used in virtual worlds. Another survey by Musialski et al. [MWA*13] provides a broad overview of techniques related to urban 3D reconstruction including computer vision techniques, facade modeling, and point cloud methods of reconstruction. While both surveys provide an in-depth review of the current algorithms and systems for procedural modeling and 3D reconstruction, there are relatively few procedural modeling papers that address automatic footprint generation in a way similar to Merrell et al. [MSK10] or the synthesis and creation of new content similar to the ideas presented by Watson et al. [WMV*08].

3. Approach

In this section, we describe additions to the data set used for constructing the probabilistic model, improvements to the layout methods in [BFM14], and the CityEngine rule file and Python script generator.

3.1. Facility and Geometric Data

A keyhole markup language (KML) [Goo15b] file with georeferenced locations for GoogleEarth Pro was programmatically constructed in Python with markers and links to reference information for the location of 445 WWTPs [Ent13] in the communities of Valencia, Castellón, and Alicante. A KML file is based on the XML format and can store placemarks, annotations, positions, geometry, and other attributes. KML files are convenient for accessing locations in Google's geographic information system applications, for traveling quickly to aerial images of geographic areas of interest, and for storing geometric surface features.

Using the KML locations we manually construct a representative data set of geometric details from aerial images (Figure 3). The data are used to impute the geometric details of all the facilities basic geometric shapes. We use the method of Expectation Maximization given by Schafer [Sch97] and as implemented in [HFH*09]. The radius of circular and spherical objects, the length and width of rectangular objects, and the site areas were measured.

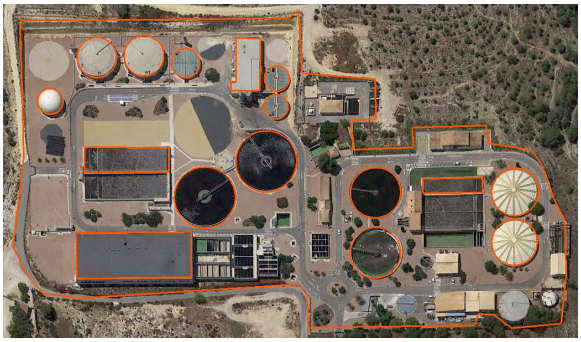


Figure 3: Facility geometric data.

3.2. Footprint Generator

A footprint, or 2D geometric shape, for buildings and structures is used in procedural modeling as the initial shape for

© The Eurographics Association 2015.

operations like extrusions, rotations, and other transformations. We synthesize industrial facility footprints by optimizing the arrangement of a set of inferred facility components from a Bayesian network [BFM14].

We adapted and improved upon the two layout methods described in [BFM14]. The first improvement (Section 3.2.1), or *hybrid layout*, involves running the physics based simulation inside the cells output by the block plan linear program optimization. The second improvement (Section 3.2.2), adds the use of a force-directed layout algorithm [FR91] to generate the set of initial coordinates for each component in the facility prior to running the physics based layout. This approach differs from the prior work which randomly initializes the initial coordinates.

3.2.1. Hybrid Layout

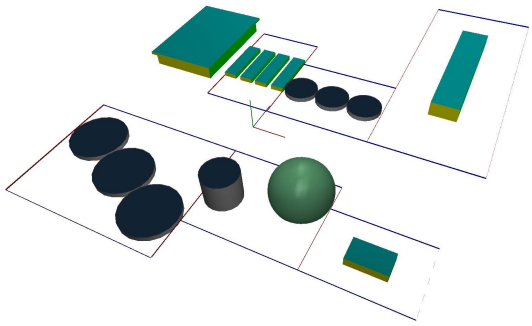
We create a hybrid layout scheme by combining the random and cellular optimization approaches from [BFM14]. The footprint is created by placing similar components in an optimized block, or cell, and allowing the physics simulation to run. A steady-state occurs when the movement of each component is less than a threshold close to zero. The cells are created by the linear program optimization [MV88]. The components in the physics simulation are restricted to their respective cells by rigid planes at the cells' borders. See Figure 4.

3.2.2. Force Directed Layout

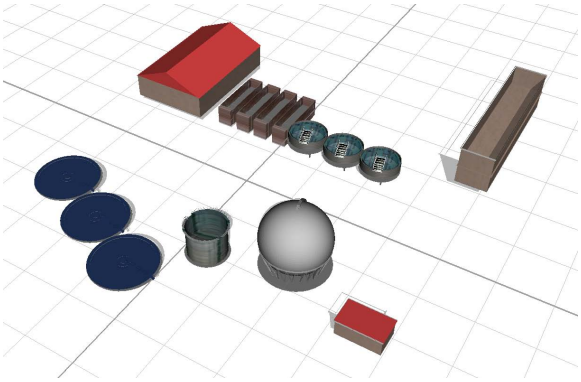
Rather than using random coordinates for the initial placement of facility components we introduce the use of the Fruchterman-Reingold force-directed graph layout algorithm [FR91] to the facility layout problem. The Fruchterman-Reingold layout algorithm repels all non-neighboring nodes using a modified version of Hooke's law used for computing spring compression forces. The attracting force is $F_{attr}(x) = \frac{x^2}{k}$ and the repelling force is $F_{repel}(x) = \frac{-k^2}{x}$ where x is the magnitude of the difference vector between two object's positions, $k = \sqrt{\frac{a}{n}}$, a is the area of the site, and n is the number of facility components in the graph [Tam07]. The output of the force-directed algorithm is used as the input to the physics based simulation [Bul13]. The physics based simulation uses spring constraints, point constraints, and initial velocities to spread out the connected facility components in a site area which is represented as a convex polygon.

3.3. Rule and Script Generators

We use CityEngine's CGA shape grammar [MWH*06] that was derived from [WWSR03] for the procedural modeling of various buildings in the WWTP. Rules are generated and assigned programmatically using a Python script generated and exported to the CityEngine project from the footprint generator. The script is run via a Jython instance and automatically called when a user selects the *Build Facility* plug-in menu item. Refer to Section 5.1 for details about the facility modeling plug-in.



(a) Layout constrained by rigidbody planes.



(b) Example procedural model built with a footprint.

Figure 4: Hybrid footprint example.

4. 3D Scene Construction

We created a synthetic scene to test the placement of the industrial facility models. The scene was constructed using terrain, high-resolution aerial imagery as textures, 3D models, and the facility model built from the methods outlined in this paper.

4.1. Facility Location

The placement of both desirable and undesirable facilities in urban areas is well-studied [Her08, RT90] and both have geometric solutions. In our work, we are concerned with the *obnoxious*, or undesirable, facility location problem. The obnoxious facility location problem is concerned with placing a facility (nuclear power plant, WWTP, etc.) a *maximum minimum* distance away from a set of locations.

© The Eurographics Association 2015.

4.1.1. Locating Facilities in Urban Areas

Once a layout and 3D model have been constructed, we are interested in placing the facility model at a reasonable site in the scene. We place a facility according to the *obnoxious* facility location problem that can be solved using a geometric construction (Figure 5). Our system takes as input a custom convex polygon in OBJ format. The footprint generator uses the polygon boundaries for site-boundary constraints. The polygon can be located either at a user's discretion, or the centroid of the polygon can be optimally placed using the obnoxious facility location method described in this section.

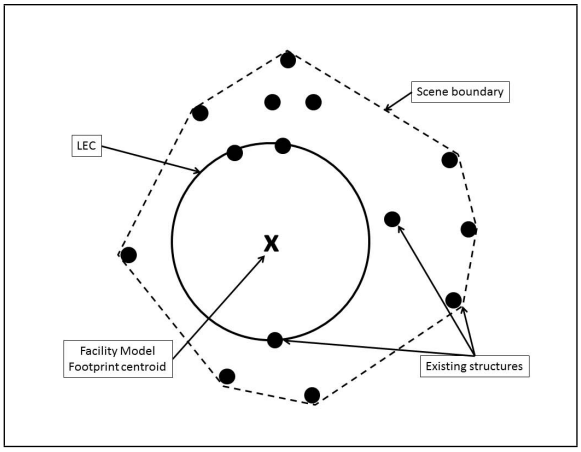


Figure 5: Obnoxious facility location is the center x of the largest empty circle (LEC) inside the convex hull of a set of locations [Tou83].

We use a geometric algorithm by Toussaint [Tou83] for computing the largest empty circle (LEC) for a set of given locations. The center of the LEC contained in the convex hull of all the locations of existing structures in the scene is the location for the obnoxious facility. We use the CGAL computational geometry library for computing the LEC. A user can choose to use this location or place the facility at any desired location in the 3D scene.

4.2. Facility Aesthetics

Once a layout has been generated we randomly place industrial 3D model assets from Google's 3D warehouse [Tri13] throughout the scene to add to the realism.

4.2.1. 3D Models

Fifty-six high fidelity 3D models of buildings (Figure 6a) were downloaded from [Tri13] and included in a CityEngine scene of the University of California, Davis (UC Davis) campus (Figure 6b). The 3D models were inserted into the 3D scene along with procedurally generated buildings and the generated 3D facility model. Prior to their use in CityEngine, the high fidelity models needed to be converted from Google Sketchup (skp) to OBJ format, and transformed to the origin in world space, uniformly scaled, and cleaned up in Maya.

(a)

(b)

Figure 6: Subset of the 3D Warehouse [Tri13] building models of UC Davis (top) and close-up of a water tower used in constructing the synthetic scene (bottom).

In addition to the models, we created a CityEngine asset library for the WWTPs and cityscape scenes. The asset library includes textures, digital elevation models (DEMs), building rules, and 3D models. A rendered close-up of the facility model, which uses a combination of existing models and procedurally generated facility components, is shown in Figure 7.

4.2.2. Terrain and Surface Textures

For the terrain, we used 7.5 minute 1-arc second (30 meter) resolution DEMs for Yolo and Solano counties in Northern California, USA. Each county contains several individual DEM files (or tiles) that were downloaded from [Unia]. The tiles were loaded into an opensource GIS software system [QGI09], stitched together and converted from the Universal Transverse Mercator (UTM) coordinate system in the North American Datum (NAD 27) to WGS 84.

In addition to terrain, we used a set of high resolution (.15 meter) orthorectified aerial imagery (ortho-imagery) acquired by the USGS in March 2009 [Unib] as the ground surface texture. The full-resolution for each texture is

© The Eurographics Association 2015.

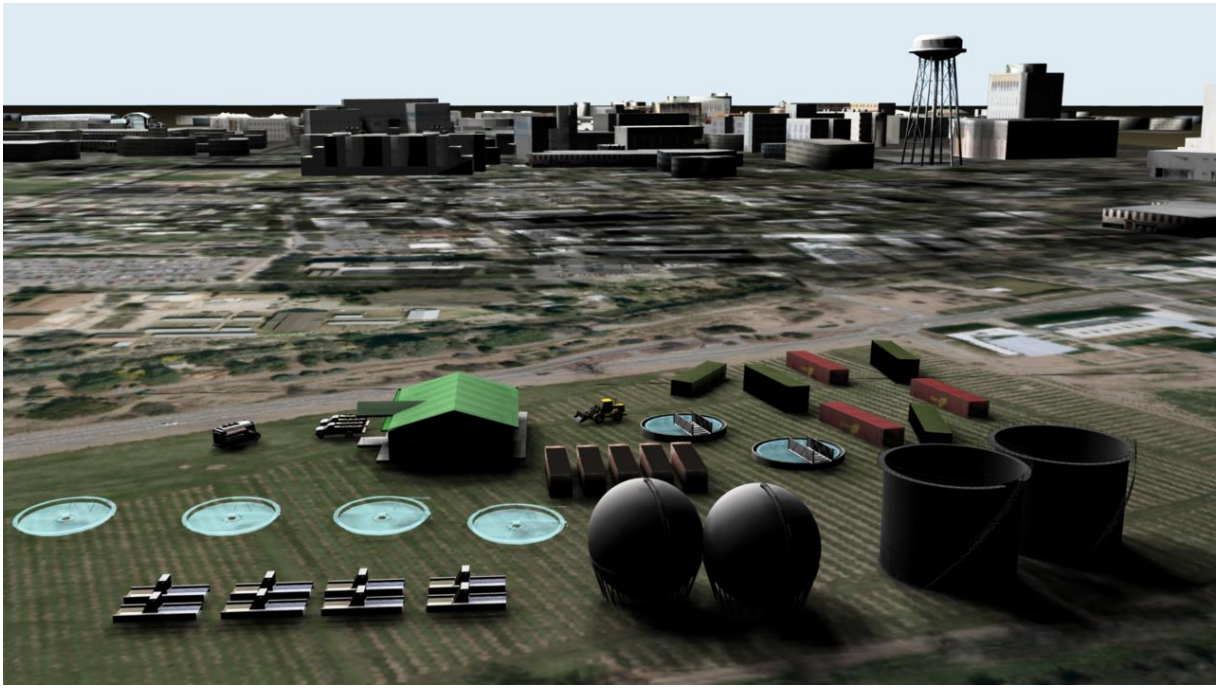


Figure 7: WWTP facility rendered with 3D models (vehicles, tanks, shipping containers, and clarifiers) from [Tri13].

5280x5280 and is approximately 108MB in size. The images were reduced to a resolution of 2000x2000 prior to using them in CityEngine. The ortho-imagery uses the North American datum 83 (NAD 83) California Zone 2 coordinate system readjusted by the national spatial reference system 2007 survey (NSRS 2007). Each image is converted to WGS 84 zone 10N prior to being used as a texture in CityEngine.

5. Implementation

We use the Eclipse Rich Client Platform (RCP) [Ecl] to generate a dynamically loaded Java [Ora15] plug-in that is self-contained in a Java archive (jar) file to perform specific 3D modeling tasks inside CityEngine.

We use the active instance of the 'Java version of the Python' (Jython) [Cor15] interpreter to execute CityEngine Python code in the CityEngine interface. The plug-in can execute two external applications, a customized Weka BayesNet Editor that can generate a list of facility components and the OpenGL layout viewport and script generator. Refer to the system schematic shown in Figure 8. To install the facility modeling plug-in, the jar file is placed inside the *CityEngine/plugins* directory.

5.1. CityEngine Java Plug-in

The Eclipse RCP is built using the OSGI component architecture specification [OSG] which provides the framework

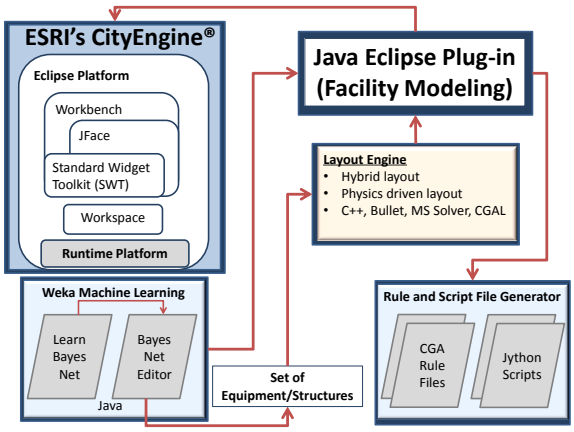


Figure 8: Schematic of the facility modeling plugin and the Eclipse plugin architecture [IBM06].

for dynamically loading bundles of files, or plug-ins, in an Eclipse RCP application. The essential files in an RCP plug-in are a manifest.mf file, the Java source files, and a plugin.xml file that maintains the plug-ins contributions via extension points into the Eclipse RCP application (commands, menu items, etc.) [ML05].

The main features of the CityEngine plug-in are the following:

© The Eurographics Association 2015.

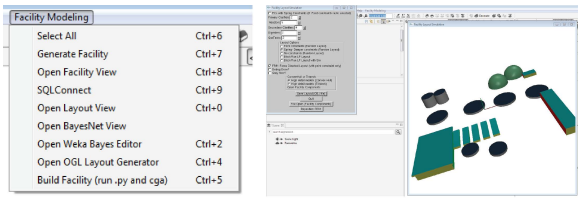


Figure 9: Facility modeling Java plug-in menu (left) and OpenGL layout generator and viewport (middle and right).

- An 'Open Weka Bayes Editor' menu (Figure 9 (left)) which allows a user to condition the Bayesian network, for example, to set the population size or a facility's flow rate.
- An 'OpenGL Layout Generator' menu item that opens the layout generator and OpenGL viewport (Figure 9 (right)).
- Lastly, a 'Build Facility (run .py and cga)' option that opens a dialog box so users can point to the location of the Python script created from the layout generator. The python script executes a series of commands and rules (written in CityEngine's CGA shape grammar). The script drives the construction of the WWTP facility model using the exported OBJ footprint from the layout generator.

Using predefined rules, each footprint component is assigned a start rule that is evaluated in CityEngine when the 'Build Facility' menu item is selected. The predefined rules include steps for constructing certain components in the WWTP (e.g. the administration building, biologic reactors, etc.) and for importing existing 3D models into the scene.

6. Results

Once an industrial facility site has been generated we use Autodesk's Maya to test render synthetic 2D images. An animated turntable of the site was created which mimics real-world data collection techniques and, when rendered, generates synthetic 2D aerial imagery (Figure 10) that can be used for other computational work.

7. Discussion

This paper discusses a method for improving the workflow of procedurally modeling industrial facilities by integrating a layout engine written in C++ into the CityEngine interface. The benefit of this is that 3D modeling workflows which use CityEngine can be customized using the Eclipse RCP framework by custom plug-ins. We present our plug-in and discuss the functionality associated with the menu items, including the layout engine, BayesNet editor, and rule file and script generator entries. In addition, we presented modifications to the layout algorithms presented in [BFM14] and discussed additions to data, and placement of industrial facilities in a scene.

8. Future Work

We are working on adding synthetic roads to the facility footprint using the road styles discussed in [CEW*08, PM01]. We propose using a geometric construction to find the medial axis between different sets of polygonal components on the site. For example, all clarifiers and pretreatment components are grouped into sets of the computed convex hulls of each component. The medial axis between the polygon groups can then be used as roads. We are also investigating improvements to the linear program optimization initial conditions. Hand-tuning the set of cells as input is not optimal and thus automatic cell arrangements based on a topological segmentation [GP12] of the terrain are being researched.

9. Acknowledgements

This research was funded in part by a grant from the U.S. National Nuclear Security Administration's Nonproliferation University Collaboration and from the 2014 Erasmus Mundus Programme of the European Commission's Transatlantic Partnership for Excellence in Engineering program. Special thanks to the members of the Institute for Water and Environmental Engineering at the Universitat Politécnica de Valéncia for their expertise in WWTP engineering.

References

[BFM14] BISHOP M. S., FERRER J., MAX N.: Data driven assembly of procedurally modeled facilities. In *Eurographics 2014 - Short Papers Proceedings, Strasbourg, France, 2014* (2014), Galin E., Wand M., (Eds.), Eurographics Association, pp. 37–40. 2, 3, 6

[Bul13] : *Bullet Physics Engine*, Oct 2013. Version 2.81. URL: http://bulletphysics.org/. 3

[CEW*08] CHEN G., ESCH G., WONKA P., MUELLER P., ZHANG E.: Interactive procedural street modeling. *ACM Trans. Graph. 27*, 103 (2008), 1–10. 6

[Cor15] CORPORATION FOR NATIONAL RESEARCH INITIATIVES: *Jython: Python for the Java Platform*. Python Software Foundation, Reston, Virginia, 2015. URL: http://www.jython.org. 5

[Ecl] ECLIPSE FOUNDATION: *EPL - Eclipse Rich Client Platform*. URL: http://wiki.eclipse.org/index.php/Rich_Client_Platform. 5

[EMP*03] EBERT D. S., MUSGRAVE F. K., PEACHEY D., PERLIN K., WORLEY S.: *Texturing and Modeling, Third Edition: A Procedural Approach (The Morgan Kaufmann Series in Computer Graphics)*, 3 ed. Morgan Kaufmann Publishers, 2003. 1

[Ent13] ENTITAT DE SANEJAMENT D'AIGÜES,COMMUNITY OF VALENCIA, SPAIN: Wastewater treatment plant data. http://www.epsar.gva.es/sanejament/index.aspx, May 2013. 2

[ESR] ESRI: *CityEngine® Procedural Modeling Tool*. 1

[FR91] FRUCHTERMAN T. M. J., REINGOLD E. M.: Graph drawing by force-directed placement. *Softw., Pract. Exper. 21*, 11 (1991), 1129–1164. 3

© The Eurographics Association 2015.

Figure 10: Example frame from an aerial view of a synthetic scene with an industrial facility (structures with red roofs).

[Goo15a] GOOGLE, INC.: *GoogleEarth Pro.*, 2015. 2

[Goo15b] GOOGLE, INC.: *Keyhole Markup Language*, 2015. 2

[GP12] GERBER S., POTTER K.: Data analysis with the morse-smale complex: The msr package for R. *Journal of Statistical Software 50*, 2 (2012), 1–22. 6

[Her08] HERAGU S.: *Facilities Design*, 3rd ed. CRC Press, 2008. 3

[HFH*09] HALL M., FRANK E., HOLMES G., PFAHRINGER B., REUTEMANN P., WITTEN I. H.: The weka data mining software: An update. *SIGKDD Explorations 11*, 1 (2009). 2

[HW08] HAKLAY M., WEBER P.: OpenStreetMap: User-generated street maps. *Pervasive Computing, IEEE 7*, 4 (Oct 2008), 12–18. 1

[IBM06] IBM CORPORATION: *Eclipse Platform Technical Overview*. Tech. rep., IBM, 2006. 5

[ML05] MCAFFER J., LEMIEUX J.-M.: *Eclipse Rich Client Platform: Designing, Coding, and Packaging Java Applications*. Addison-Wesley Professional, 2005. 5

[MM11] MERRELL P., MANOCHA D.: Model synthesis: A general procedural modeling algorithm. *Visualization and Computer Graphics, IEEE Transactions on 17*, 6 (June 2011), 715–728. 1

[MSK10] MERRELL P., SCHKUFZA E., KOLTUN V.: Computer-generated residential building layouts. In *ACM SIGGRAPH Asia 2010 papers* (New York, NY, USA, 2010), SIGGRAPH ASIA '10, ACM, pp. 181:1–181:12. 1, 2

[MV88] MONTREUIL B., VENKATADRI U.: *From gross to net layouts: an efficient design model*. Tech. rep., Québec: Université Laval, Faculté des sciences de l´administration, September 1988. HF 5006 U58 88-56. 3

[MWA*13] MUSIALSKI P., WONKA P., ALIAGA D. G., WIMMER M., VAN GOOL L., PURGATHOFER W.: A survey of urban reconstruction. *Computer Graphics Forum 32*, 6 (2013), 146–177. 2

[MWH*06] MÜLLER P., WONKA P., HAEGLER S., ULMER A., VAN GOOL L.: *Procedural modeling of buildings*, vol. 25. ACM, 2006. 1, 3

[Ope11] OPENSTREETMAP: *OpenStreetMaps, licensed under the Creative Commons Attribution-ShareAlike 2.0*, 2011. URL: http://www.openstreetmap.org. 1, 2

[Ora15] ORACLE CORPORATION: *Java Programming Language*, 2015. URL: https://www.oracle.com/java. 5

[OSG] OSGI ALLIANCE: *The OSGI Architecture*. URL: http://www.osgi.org/Technology/WhatIsOSGi. 5

[PM01] PARISH Y. I. H., MÜLLER P.: Procedural modeling of cities. In *Proc. of the 28th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 2001), SIGGRAPH 2001, ACM, pp. 301–308. 6

[QGI09] QGIS DEVELOPMENT TEAM: *QGIS Geographic Information System*. Open Source Geospatial Foundation, 2009. URL: http://qgis.osgeo.org. 4

[RT90] ROBERT J.-M., TOUSSAINT G.: Computational geometry and facility location. In *Proc. International Conference on Operations Research and Management Science* (1990), pp. 11–15. 3

[Sch97] SCHAFER J. L.: *Analysis of Incomplete Multivariate Data*. Chapman & Hall, New York, 1997. 2

[STBB14] SMELIK R. M., TUTENEL T., BIDARRA R., BENES B.: A survey on procedural modelling for virtual worlds. *Computer Graphics Forum 33*, 6 (2014), 31–50. 2

[Tam07] TAMASSIA R.: *Handbook of Graph Drawing and Visualization (Discrete Mathematics and Its Applications)*. Chapman & Hall/CRC, 2007. 3

[Tou83] TOUSSAINT G.: Computing largest empty circles with location constraints. *International Journal of Computer & Information Sciences 12*, 5 (1983), 347–358. 4

[Tri13] TRIMBLE: *Trimble 3D Warehouse: online 3D model repository.*, 2013. 4, 5

[Unia] UNITED STATES GEOLOGICAL SURVEY: *7.5 Minute Digital Elevation Models*. URL: http://www.webgis.com/terr_pages/CA/dem75/solano.html. 4

[Unib] UNITED STATES GEOLOGICAL SURVEY: *High Resolution Orthoimagery*. URL: http://gisdata.usgs.gov/listofortho.php. 4

[WMV*08] WATSON B., MÜLLER P., VERYOVKA O., FULLER A., WONKA P., SEXTON C.: Procedural urban modeling in practice. *IEEE Computer Graphics and Applications 28*, 3 (2008), 18–26. 2

[WWSR03] WONKA P., WIMMER M., SILLION F., RIBARSKY W.: Instant architecture. In *ACM SIGGRAPH 2003* (New York, NY, USA, 2003), ACM, pp. 669–677. 3

© The Eurographics Association 2015.