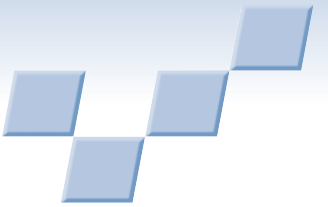

Procedural Urban Modeling in Practice

Benjamin Watson ■ *North Carolina State University*

Pascal Müller ■ *Procedural Inc.*

Peter Wonka ■ *Arizona State University*

Chris Sexton ■ *Johns Hopkins University*

Oleg Veryovka and Andy Fuller ■ *Electronic Arts*

Film and game studios can no longer meet audience demand for visual content by increasing production budgets. Instead they are turning to procedural modeling, particularly for modeling cities. The authors review procedural modeling, examine the CityEngine tool, and study the use of procedural urban modeling in Electronic Arts' Need for Speed games.

Procedural modeling is finally going mainstream. Until recently it was a boutique solution, used only when no other options existed. Now procedural modeling is often the cost-effective solution, used because the alternatives are too expensive. In film, games, and other applications, consumers expect richer, higher-quality digital content for their dollar. Because budgets won't allow content producers to increase cost significantly, they have only one choice: they must improve their tools. Procedural modeling is the primary ingredient in these tools.

One of the main drivers of these trends is urban content. Cities are huge, richly detailed artifacts often required in digital productions. Modeling them with existing, nonprocedural tools can take hundreds of man-years. Several researchers are creating procedural techniques specifically for automating city modeling.

A brief history

Computer graphics practitioners have long used procedural modeling to generate nonurban content. L-system grammars generate plants,[1] while agent-based particle systems model fuzzy objects such as fire and smoke[2]—most famously the "Genesis effect" in *Star Trek*. Perlin's noise[3] simulates clouds and natural textures, while Reynolds' boids[4] apply agent-based methods to animate flocks, schools, and herds, including stampeding wildebeests in *The Lion King*. Genetic techniques similar to those described by Sims[5] will soon see use in Spore, a game from Electronic Arts (EA).

Synthesizing natural landscapes has a similarly long research history, but since cities sit on landscapes this approach applies more directly to urban problems than the techniques discussed previously. Fournier and colleagues use stochastic, fractal techniques inspired by Mandelbrot.[6] Musgrave and colleagues extend these techniques to model erosion effects.[7] More recently, Zhou and colleagues applied texture synthesis—used to make large, unique textures from small source patches—to create natural terrain that respects artist constraints.[8] Planetside's Terragen, a commercial software package widely used in film and games, implements many of these techniques.

Synthesizing urban terrain

Procedural techniques dedicated to urban synthesis have only begun to appear more recently. Parish and Müller use L-systems to model extensive street layouts and buildings.[9] Given input maps of geography, population density, and layout constraints, their system generates streets, subdivides land, and creates skyscrapers. Figure 1 shows an example of road generation.

 Published by the IEEE Computer Society

Figure 1. Examples of procedural road synthesis.[9]

Lechner and colleagues use agent-based technology to build transportation networks, subdivide land, allocate use, and manage population density.[10] Their agents adapt to their environment, letting artists steer development interactively to meet specific application constraints. A recent extension to their work by Sexton and Watson vectorizes the gridded simulation output, producing realistic, smoothed, urban terrains that you can import into various geographic information system (GIS) tools for further analysis.[11]

Esch and colleagues support interactive modification of street layouts through 2D tensor fields (tensors are a generalization of vectors).[12] Artists can change layouts indirectly by manipulating the tensor field or directly by reshaping the roads themselves.

In gaming and other interactive applications, the ability to synthesize urban content in real time would be especially useful. Greuter and colleagues describe a first pass at this problem, laying out cities using a simple grid, and creating simple skyscrapers from input footprints on the fly.[13] To meet real-time constraints, they pay particular attention to caching and limiting computation outside the view frustum.

Synthesizing buildings and other structures

In 1971, the architect Stiny introduced shape grammars to bring a new formalism and rigor to designing and analyzing architecture.[14] Many designers adopted shape grammars, but they remained largely conceptual tools, synthesizing only conceptual mass models of buildings and other structures. Wonka as well as Müller and colleagues took the next step, introducing split grammars for synthesizing detailed building facades,[15] building a shape grammar called CGA Shape for creating entire building exteriors,[16] and creating the CityEngine integrated modeling environment. More recently, they have used computer vision to automatically generate grammars describing facades.[17] Aliaga and colleagues describe a similar system that facilitates authoring grammars for texturing building exteriors.[18]

Merrell generates buildings (and many other shapes) using a texture-synthesis-inspired technique.[19] The artist partitions a small example model into parts using a 3D grid. This partition then generates a set of constraints: two parts might only be adjacent in the output model if they were adjacent in the example model. Merrell uses optimized search in this constrained space to generate output models

Many structures such as bridges and train stations are best characterized by the layout of their supporting beams and infrastructure. Pottmann and colleagues synthesize the quadrilateral, pentagonal, and hexagonal meshes that characterize many beam layouts.[20] They can find layouts for almost any curved surface. Smith and colleagues automate the design of the truss structures that support buildings, bridges, and many other structures.[21] Their method accommodates the real-life engineering constraints that relate geometry to mass and stress.

Floor plans describe building interiors and often dictate the appearance of building exteriors, especially in homes. Harada and colleagues apply optimization algorithms to the design of 2D architectural floor plans,[22] while Martin devises a grammar that constructs graphs in which nodes represent rooms and links connections between rooms.[23] He then translates these graphs into interior and exterior building geometry. Hahn and colleagues generate building floor plans and corresponding 3D interiors in real time by randomly dividing rectangular floors into rectangular rooms and hallways.[24] The division respects several basic architectural constraints such as connecting adjacent hallways and ensuring that small private rooms are immediately accessible from public spaces. Like many researchers, Hahn and colleagues store and reuse random seeds during real-time generation, ensuring that a given floor has the same floor plan at each viewing.

Synthesizing other urban content

Urban content is more than building placement and shape. Legakis and colleagues synthesize "cellular" textures of brick, tile, and masonry for buildings and other structures.[25] Textures respect component (such as brick) shape, commonly used tiling patterns, and structure geometry. For example, brick textures on each side of a corner reflect the fact that the same 3D bricks occupy both textures.

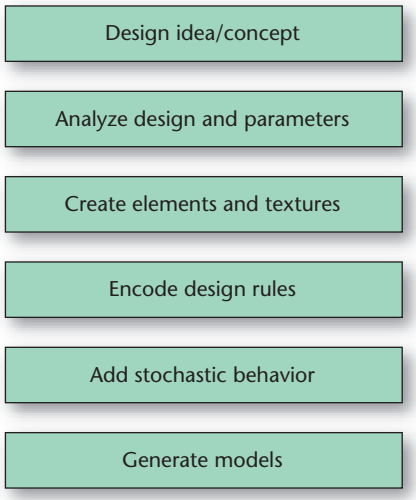

Figure 2. The workflow for a typical architectural procedural-modeling project.

Weathering and wear are important elements of urban realism. Dorsey and colleagues model the weathering of stone, including oxidation and erosion.[26] Chen and colleagues simulate weather's effects on additional materials and model other types of weathering, including moss, rust, and dirt accumulation.[27]

Without people, cities are ghost towns. Thomas and Donikian populate cities with cars and pedestrians, each with matching behaviors and animations.[28] Maïm and colleagues apply the CityEngine to generate an annotated city model and interpret these semantics to automatically populate the scene and trigger special behaviors in the crowd, depending on the characters' location.[29]

CityEngine applications

One of the most mature procedural modeling tools available is the CityEngine. Effective use of the CityEngine, and indeed almost any urban-modeling tool, requires familiarity with architecture. You should begin by acquiring a good understanding of basic building elements such as windows, doors, columns, pilasters, quoins, gates, roofs, cornices, arches, walls, and ornaments. We recommend examining one to three architecture books with labeled illustrations of these elements. One of the best is by Köpf and Binding, but unfortunately it's available only in German.[30] A similar book is a *Visual Dictionary of Architecture* (John Wiley & Sons, 1996).

The next step is to create grammar rules for combining these basic elements. Unfortunately, the existing architectural literature describes these rules ambiguously; formal and procedural methods aren't widely used in architecture and have little tradition in the field. Typically, modelers must therefore derive rules and structure without relying strongly on the existing literature.

How difficult is using the CityEngine? In the three example projects we describe, we collaborated with archeologists, urban planners, and artists to produce their models and found that shape grammars are as easy to learn as scripting languages. Users unfamiliar with scripting generated grammars without difficulty using the CityEngine's visual interface or its image-based methods. Soon the market will render its own judgment: Procedural Inc. plans to release the CityEngine in the second quarter of 2008.

Workflow

Workflow in the CityEngine typically begins with a specific idea stemming from a photograph, a drawing, an architectural figure, or a new design concept (see Figure 2).[9,31] The next step is to analyze the design and find the most important parameters.

For example, consider Le Corbusier's Cruciform Skyscraper. Le Corbusier designed several detailed variations of the skyscraper between 1920 and 1930, with the most famous incarnations appearing in the master plans for his Contemporary City (1922) or the Plan Voisin (1925). The enormous (for that time) 60-story skyscrapers were built on steel frames and encased in huge curtain walls of glass. They housed both offices and the flats of the wealthiest urban inhabitants and were set in large, rectangular, park-like green spaces.

Figure 3 shows a sketch of the skyscraper design and a visual analysis showing its simplified structure. During analysis you must name individual elements and identify important design parameters. Here we choose the names core, spine, wing, and tooth. We also start with seven parameters for the main mass of the building: overall height, ground floor height, platform height, wing width, small-wing width, teeth width, and the distance between two teeth. During this analysis, we also model detailed textures and individual building elements, such as window geometry. (After several design projects, you can reuse many previously created building elements.)

We then encode shape grammar rules for assembling the skyscraper's crude mass model out of basic solids (mainly boxes) and for constructing its facades. After confirming that the resulting proportions match a specific sketch, we carefully start randomizing parameters to create stochastic rules that generate a whole city. It's important to begin with one working instance and then add randomness gradually, because too much randomness creates chaotic, uninteresting designs.

We recommend that new CityEngine users gain

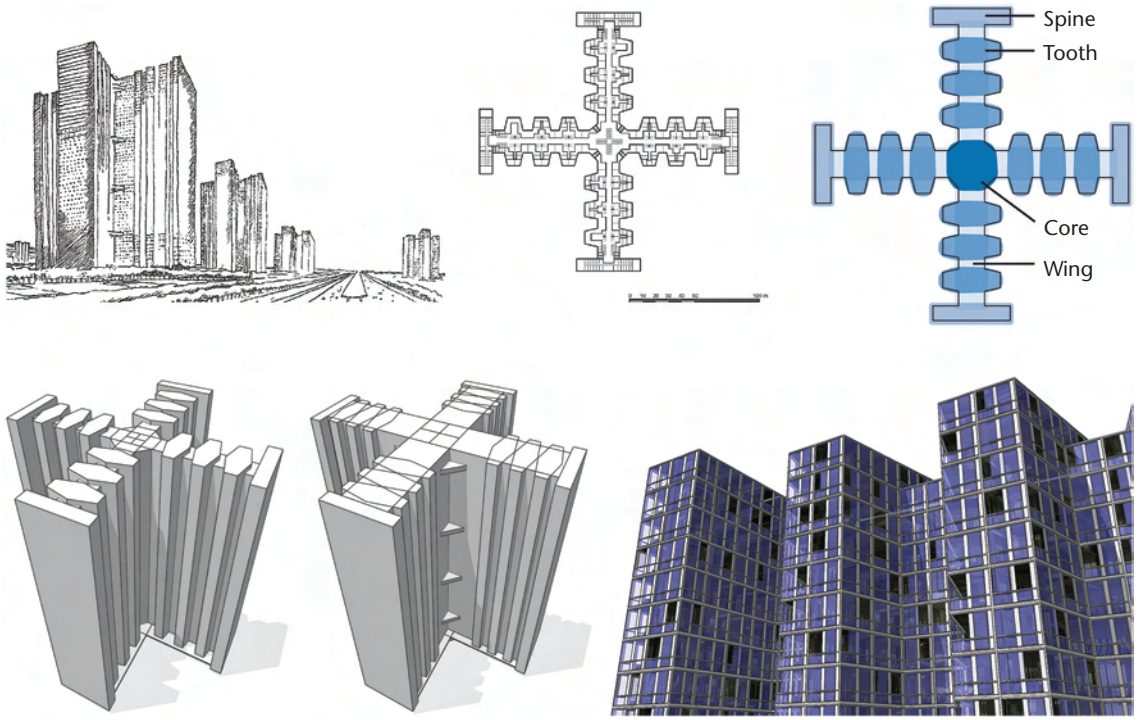

Figure 3. Top: Views of Le Corbusier's Cruciform Skyscraper and the simplified design structure. Bottom: 3D depictions generated with the CityEngine, including mass model variations created by changing parameters and a building with a facade applied.

experience with this procedure by building several simple buildings. Just a few well-modeled stochastic buildings can populate a complete city. For example, Parish and Müller's model of New York had only six different building types.[9]

A cultural heritage application

Experts and laymen study historical structures closely. Digital models of these elements of our cultural heritage are valuable tools for analysis, reconstruction, and virtual display. While the focus of cultural heritage digitization is still on 3D modeling of important major monuments such as the Parthenon, there's often additional demand for modeling larger settlements. Such settlements might be interesting or might only form the context for a monument.

While archeologists have detailed architectural knowledge of the monuments and settlements they study, they have little formal training in CAD or computer graphics modeling packages. Procedural-modeling tools such as the CityEngine can bridge this gap, providing a user-friendly, high-level interface and filling in detail where the archeological record is incomplete.

In close collaboration with archaeologists from Bonn, we recently reconstructed the ancient Mayan city of Xkipche in Mexico. We describe this project elsewhere.[16] Excavations provided detailed locations and descriptions of buildings, which were built primarily in one specific architectural style. Following the workflow we just described, we began analyzing the building and identified the most important parameters. While we could rely on some drawings and assistance from our archeological collaborators, we had no formal design description for the Xkipche style. So, we performed most of the analysis ourselves, with continual review and comment from archeologists, producing a building model encoded with 39 shape-grammar rules and 32 control parameters.

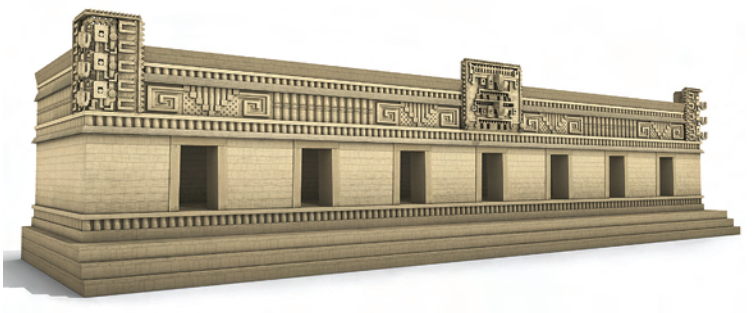

Figure 4. A Mayan building from Xkipche in Mexico generated with the CityEngine for a cultural heritage project.

To reconstruct the whole site, archaeologists imported their GIS data (footprints with metadata such as building height) into the CityEngine and selected buildings for 3D reconstruction. Archeologists then interactively defined the 32 parameters of those buildings while examining a 3D building preview. When the archeologists were satisfied with their reconstruction, they stored each building's parameters in the GIS database. The complete city can be generated and stored on disk at any time. Figure 4 shows a building model that we created with our system.

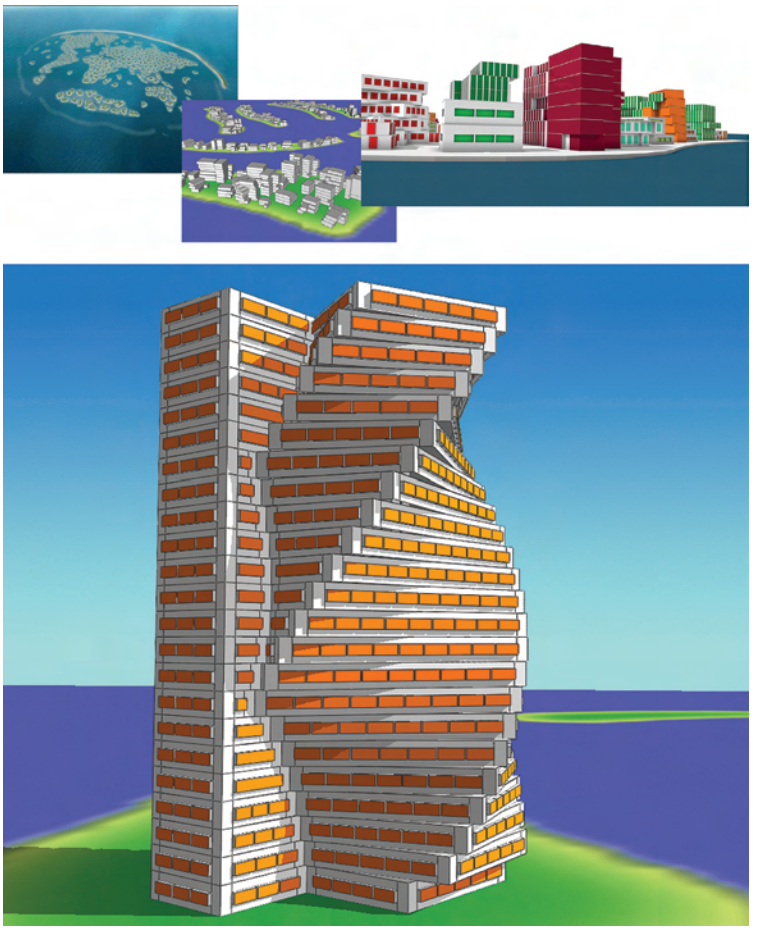

Figure 5. Some results from the Dubai World Islands urban planning project.

Use in urban planning

Urban master planning regulates, directs, and projects future development in cities. Long before any of the structures they envision are built, planners must produce impressive visualizations of them, which are used both in design competitions and in political decision making. Not only can procedural modeling simplify the production of such visualizations, it can also illustrate the remaining artistic possibilities through stochastic, procedural variation.

Urban plans must conform to numerous zoning regulations, the most important of which is the *building envelope*, which defines the volume wherein a building must be designed. Sometimes the envelope is the extruded property line, but setbacks in the form of angles at the line or distances from the line are common. Density is controlled using floor-area ratios: total floor area divided by property area. (Taller buildings will have higher ratios.) With the percentage of covered area on each envelope surface, regulations can ensure consistent facade alignments. Lighting rules limit the shadows buildings cast. For example, buildings over a certain height might not be permitted to cast a shadow on adjacent housing for more than 2 hours.

We implemented a simple prototype to visualize a master plan for the Dubai World Islands. Figure 5 shows some output from this prototype. We created rules by analyzing building sketches provided by urban planners who, like our archeologist collaborators, loaded the 2D plan into the CityEngine and interactively redefined buildings until they were satisfied. Ultimately, we would like to incorporate zoning regulations into the CityEngine as rules, so that all generated models will conform to them.

The CityEngine in film

The traditional production pipeline in the movie industry is broken down into clearly separated, sequential stages. The overall control lies with the director and producer. The art director supervises the first stage, preproduction, which develops initial drawings and 3D models. The second stage, if live action is included, is filming, which the director of photography supervises. The visual-effects supervisor leads postproduction, the last stage. Postproduction refines preproduction concepts and combines them with footage from filming to produce composited frames. Most computer graphics work occurs in postproduction and includes the creation of the final 3D textured models, animations, and lighting.

Because digital effects have become more prevalent, pre- and postproduction have become more integrated. Procedural modeling can strengthen this integration by providing a single, flexible, digital representation, saving both time and cost. For example, in preproduction, artists might describe initial designs using coarse urban layouts and 3D building mass models, augmented with a few detailed facade designs to convey overall appearance. As the designs mature, even major adjustments to building layout and shape can be made without losing facade detail and the work that defined it, because rules are defined within the context of larger-scale designs. Figure 6 shows how the CityEngine supported such changes during movie production.

Procedural urban modeling in racing games

Demand for procedural technology in game development is unique. Over the last decade, video games have grown into a large and lucrative sector of the software development and entertainment industries. The visual complexity of games released for modern game consoles such as Sony's PlayStation 3 and Microsoft's Xbox 360 now rivals the visual complexity of computer-generated films. This generates content demand with elements similar to and distinct from content demand for film. Here

we look at racing games, a genre that has a voracious appetite for new urban content, and how procedural modeling is meeting this demand. We focus on EA's Need for Speed (NFS) racing games, as one of the most significant examples of this genre.

Recent NFS games include more than 100 miles of roads in dense urban, suburban, and rural settings (see Figure 7). Environments vary from real race tracks and famous world locations to fictional locations designed to provide a fun driving experience. NFS worlds are filled with thousands of unique elements or artistic assets such as architectural buildings, objects, signs, lights, and organics such as trees, bushes, and grass.

Real-time display requirements (at least 30 frames per second) force game assets to conform to strict budgets, measured in the number of polygons, materials, draw calls, shader complexity, and texture sizes. The challenge of maintaining high visual complexity while conforming to these budgets is daunting, and differentiates video game production from film.

Procedural-modeling techniques offer tools that can speed up development of game assets. Game artists aren't looking for a one-button procedural solution. Instead, they're interested in procedural methods that help with tedious tasks and provide results that adjust to gaming constraints. Procedural methods should free artists to spend time creating and polishing, rather than performing mundane, repetitive, and time-consuming tasks.

Fitting procedural modeling to game development workflow

Procedural techniques must fit into established workflow and production processes. The process for EA's *NFS* games contains three primary stages: road, terrain, and building development.

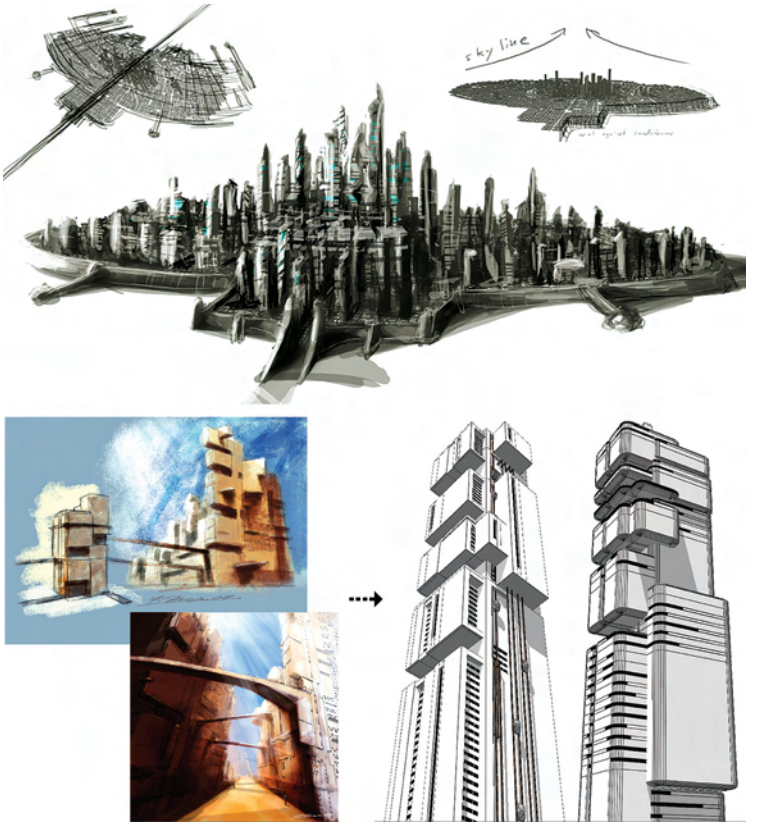

Figure 6. Some designs for a movie that used the CityEngine in pre- and postproduction. Top: overall city design. Bottom: Sketches for individual buildings and procedural building models derived from the design sketches. Conceptual images by Filip Krnja.

With roads, artists are limited to a few unique tileable textures. Road geometry requires regular tessellation and UV mapping that guarantees constant pixel density for all road elements: base surface, road lines, and details such as grime, potholes, and cracks. Maintaining pixel density is particularly challenging at intersections because roads change shape, require turns, and then widen. Our NFS road tool automates road creation

Figure 7. The world in the Need for Speed racing games includes visually complex and detailed models of hundreds of miles of roads in city, suburban, and rural settings.

(a) (b)

Figure 8.
(a) Architectural model.
(b) Reusable and tileable textures used in architectural modeling.

and lets artists lay out roads quickly, add tessellation, and apply UV mapping. Roads can change many times during production. They remain editable and retain UV mapping no matter how many modifications are made. Authoring sharp transitions from the road's edge to dirt or the surrounding environment using repeatable texture is particularly challenging. Unfortunately, existing procedural techniques are limited in this area and don't yet offer acceptable solutions.

Procedural methods for authoring natural terrains are well established and widely used. Authoring terrain in urban gaming environments remains difficult and includes these challenges:

- providing correct shapes, locations, and sizes of foundations for buildings;
- minimizing polygon density to meet memory budgets;
- minimizing unique textures; and
- having a uniform UV mapping across the entire terrain, including building footprints.

Artists populate the terrain with objects including trees and organics, road signs, light posts, fountains, waterfalls, and buildings. Some objects, such as shortcuts and hiding spots, involve gameplay, while others are only for viewing and aren't interactive.

Authoring buildings is one of the most time-consuming elements of the NFS production pipeline. Artists construct architectural models using large polygons and texture atlases with tileable and reusable textures (see Figure 8). They use surface materials to add unique shading properties such as reflections. They must generate normal and offset maps accurately for hundreds of buildings and objects. EA's artists currently have no procedural assistance for this work.

A wish list of procedural tools for game development

Procedural methods for placing world objects would be extremely helpful. To respect asset budgets, such tools must be smart enough to increase the density of objects close to the car camera and decrease the density of objects far from the car camera. Artists might want to eliminate parts of buildings that players will never see.

While organic objects such as trees greatly enhance the look and feel of racing games, rendering them is expensive. Existing procedural methods for modeling convincing trees require high-resolution textures and many polygons and can't meet game asset budgets. Constructing highly optimized organic objects by hand is tedious and time-consuming. Game artists are looking for procedural methods for modeling organic objects that meet asset budgets and yet remain convincing.

Many games are released on multiple hardware platforms and must reuse digital content. Each platform has unique strengths and weaknesses; accordingly, asset budgets differ widely across platforms. Procedural methods can improve content reusability by automating control of both model complexity and local model frequency. For example, a procedural tool might place highly detailed models close to a road and more coarsely detailed models farther from it.

Procedural urban modeling is becoming increasingly important in industrial practice but can still improve its fit to industrial workflows. Possible improvements include these:

- *Automated grammar learning*. One of the most challenging aspects of using grammar-based procedural modelers is producing the rules that generate the desired models. We've cited some initial work in automating this process,[17] but further work is required.
- *Visual-grammar interfaces*. Grammars are widely used in procedural urban modeling, but textual grammar interfaces aren't well suited for artists and designers who aren't fluent in scripting.
- *Rule libraries*. Modifying an existing rule set (such as a grammar) to meet new requirements is much simpler than analyzing a design and creating a new set from scratch. Creating libraries of useful rule sets representing complete analyses could lower this "bootstrapping barrier."
- *Site and structure integration*. Buildings are designed to fit their site, and sites are designed to fit buildings. Most procedural methods focus on sites or buildings in isolation.
- *Structure and shading integration*. Urban structures are built from certain common materials (such as paint and asphalt) and are used in a specific manner (for example, by pedestrians and cars). Integrating this information into

grammars and rule sets should make procedural approaches much more powerful.
- *Procedural detail control.* Especially for interactive applications such as games, procedural techniques should respect detail constraints such as asset budgets by adapting to parameters such as precomputed and real-time visibility.

Further research in these directions will help realize the full potential of procedural urban modeling.

References

1. P. Prusinkiewicz, A. Lindenmayer, and J. Hanan, "Development Models of Herbaceous Plants for Computer Imagery Purposes," *Proc. Siggraph*, ACM Press, 1988, pp. 141–150.
2. W.T. Reeves, "Particle Systems—A Technique for Modeling a Class of Fuzzy Objects," *Proc. Siggraph*, ACM Press, 1983, pp. 359–375.
3. K. Perlin, "An Image Synthesizer," *Proc. Siggraph*, ACM Press, 1985, pp. 287–296.
4. C.W. Reynolds, "Flocks, Herds and Schools: A Distributed Behavioral Model," *Proc. Siggraph*, ACM Press, 1987, pp. 25–34.
5. K. Sims, "Evolving Virtual Creatures," *Proc. Siggraph*, ACM Press, 1994, pp. 15–22.
6. A. Fournier, D. Fussell, and L. Carpenter, "Computer Rendering of Stochastic Models," *Comm. ACM*, vol. 25, no. 6, 1982, pp. 371–384.
7. F.K. Musgrave, C.E. Kolb, and R.S. Mace, "The Synthesis and Rendering of Eroded Fractal Terrains," *Proc. Siggraph*, ACM Press, 1989, pp. 41–50.
8. H. Zhou et al., "Terrain Synthesis from Digital Elevation Models," *IEEE Trans. Visualization and Computer Graphics*, vol. 13, no. 4, 2007, pp. 834–848.
9. Y.I.H. Parish and P. Müller, "Procedural Modeling of Cities," *Proc. Siggraph*, ACM Press, 2001, pp. 301–308.
10. T. Lechner et al., *Procedural Modeling of Land Use in Cities*, tech. report NWU-CS-04-38, Dept. Computer Science, Northwestern Univ., 2004.
11. C. Sexton and B. Watson, "Vectorization of Gridded Urban Land Use Data," *ACM Siggraph Posters*, ACM Press, 2007, p. 71.
12. G. Esch et al., *Interactive Procedural Street Modeling*, tech. report CS07-10-01, Dept. Computer Science, Oregon State Univ., 2007.
13. S. Greuter et al., "Real-Time Procedural Generation of 'Pseudo Infinite' Cities," *Proc. 1st Int'l Conf. Computer Graphics and Interactive Techniques in Australasia and South East Asia* (Graphite 03), ACM Press, 2003, pp. 87–94.
14. G. Stiny and J. Gips, "Shape Grammars and the Generative Specification of Painting and Sculpture," *Proc. IFIP Congress 71*, North-Holland, 1972, pp. 1460–1465.
15. P. Wonka et al., "Instant Architecture," *ACM Trans. Graphics* (Proc. Siggraph), vol. 22, no. 3, 2003, pp. 669–677.
16. P. Müller et al., "Procedural 3D Reconstruction of Puuc Buildings in Xkipche," *Proc. Eurographics Symp. Virtual Reality, Archaeology and Cultural Heritage* (VAST 06), Eurographics, 2006, pp. 139–146.
17. P. Müller et al., "Image-Based Procedural Modeling of Facades," *ACM Trans. Graphics* (Proc. Siggraph), vol. 26, no. 3, 2007, article no. 85.
18. D. Aliaga, P.A. Rosen, and D.R. Bekins, "Style Grammars for Interactive Visualization of Architecture," *IEEE Trans. Visualization and Computer Graphics*, vol. 13, no. 4, 2007, pp. 786–797.
19. P. Merrell, "Example-Based Model Synthesis," *Proc. 2007 Symp. Interactive 3D Graphics and Games* (I3D 07), ACM Press, 2007, pp. 105–112.
20. H. Pottmann et al., "Geometry of Multi-Layer Freeform Structures for Architecture," *ACM Trans. Graphics* (Proc. Siggraph), vol. 26, no. 3, 2007, article no. 65.
21. J. Smith et al., "Creating Models of Truss Structures with Optimization," *Proc. Siggraph*, ACM Press, 2002, pp. 295–301.
22. M. Harada, A. Witkin, and D. Baraff, "Interactive Physically-Based Manipulation of Discrete/Continuous Models," *Proc. Siggraph*, ACM Press, 1995, pp. 199–208.
23. J. Martin, "Procedural House Generation: A Method for Dynamically Generating Floor Plans," *Proc. Symp. Interactive 3D Graphics and Games: Posters*, ACM Press, 2006.
24. E. Hahn, P. Bose, and A. Whitehead, "Persistent Realtime Building Interior Generation," *Proc. 2006 ACM Siggraph Symp. Videogames*, ACM Press, 2006, pp. 179–186.
25. J. Legakis, J. Dorsey, and S. Gortler, "Feature-Based Cellular Texturing for Architectural Models," *Proc. Siggraph*, ACM Press, 2001, pp. 309–316.
26. J. Dorsey et al., "Modeling and Rendering of Weathered Stone," *Proc. Siggraph*, ACM Press, 1999, pp. 225–234.
27. Y. Chen et al., "Visual Simulation of Weathering by Y-Ton Tracing," *ACM Trans. Graphics* (Proc. Siggraph), vol. 24, no. 3, 2005, pp. 1127–1133.
28. G. Thomas and S. Donikian, "Modelling Virtual Cities Dedicated to Behavioural Animation," *Computer Graphics Forum*, vol. 19, no. 3, 2000, pp. 71–80.
29. J. Maïm et al., "Populating Ancient Pompeii with Crowds of Virtual Romans," *Proc. Eurographics Symp. Virtual Reality, Archaeology and Cultural Heritage* (VAST 07), Eurographics, 2007, pp. 109–116.
30. H. Köpf and G. Binding, *Bildwörterbuch der Architektur (Gebundene Ausgabe)* [Visual Dictionary of Architecture], Kröner, 2005 (in German).

31. P. Müller et al., "Procedural Modeling of Buildings," *ACM Trans Graphics* (Proc. Siggraph), 2006, vol. 25, no. 3, pp. 614–623.

Benjamin Watson *is an associate professor of computer science at North Carolina State University. His Design Graphics Lab focuses on the creation of meaning in imagery and spans the intersections between graphics and perception, design, and interaction. His work has been applied in digital entertainment, computer security, financial analysis, education, and medical assessment. Watson earned a doctorate in computer science at the Georgia Institute of Technology. He cochaired the Graphics Interface 2001, IEEE VR 2004, and ACM I3D 2006 conferences and was coprogram chair of I3D 2007. He's an ACM and senior IEEE member. Contact him at bwatson@ncsu.edu.*

Pascal Müller *is cofounder and CEO of Procedural Inc., a company specialized in software for the efficient creation of 3D buildings and cities. His main interests are procedural and image-based modeling, visual effects production, generative design, and architecture. During his PhD thesis at the Computer Vision Laboratory at ETH Zurich, Muller developed the CityEngine and published several scientific papers. Contact him at pascal.mueller@procedural.com.*

Peter Wonka *is an assistant professor in the Department of Computer Science and Engineering of Arizona State University. His research interests include real-time rendering, procedural urban modeling, and the application of computer graphics and visualization to various urban planning problems. Wonka received a PhD in computer science and an MS in urban planning from the Vienna University of Technology. He's a member of the Partnership in Research and Spatial Modeling lab. Contact him at pwonka@gmail.com.*

Chris Sexton *is a researcher at the Johns Hopkins University Applied Physics Laboratory. His research interests center on urban modeling and capture. Sexton received an MS in computer science from North Carolina State University. Contact him at cgsexton@gmail.com.*

Oleg Veryovka *is a technical director at Electronic Arts. He leads development of computer graphics software and tools for the best-selling* Need for Speed *series of computer games. His research areas are texture control in image halftoning, nonphotorealistic and stylized rendering, preservation of image detail, and image quality measures. Veryovka received a PhD in computer graphics from the University of Alberta. Contact him at olegv@ea.com.*

Andy Fuller *is an associate computer graphics supervisor at Electronic Arts. He has worked full time in the computer game development industry since 1994 (including porting the original* Need for Speed *onto the Sega Saturn). Fuller received a two-year civil engineering degree from Bellingham Technical College and studied at the Seattle Art Institute. Contact him at afuller@ea.com.*

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