Volume 0 (1981), Number 0 pp. 1–12 COMPUTER GRAPHICS *forum*

A Shape Grammar for Developing Glyph-based Visualizations

P. Karnick[†1] and S. Jeschke[1] and D. Cline[1] and A. Razdan[2] and E. Wentz[3] and P. Wonka[1]

[1]Department of Computer Science and Engineering, Arizona State University, Tempe USA
[2]Division of Computing Studies, Arizona State University Polytechnic, Mesa USA
[3]School of Geographical Sciences, Arizona State University, Tempe USA

Abstract

In this paper we address the question of how to quickly model glyph-based GIS visualizations. Our solution is based on using shape grammars to set up the different aspects of a visualization, including the geometric content of the visualization, methods for resolving layout conflicts and interaction methods. Our approach significantly increases modeling efficiency over similarly flexible systems currently in use.

Categories and Subject Descriptors (according to ACM CCS): I.3.2 [Computer Graphics]: Graphics Systems I.3.4 [Computer Graphics]: Graphics Utilities

1. Introduction

This paper describes a method for authoring interactive visualizations for the Geographic Information System (GIS) pipeline [Gah05]. The task of designing a GIS visualization usually means generating meaningful visual representations of primitives (typically points, lines, and polygons associated with attribute data). Various tools have been developed, both commercially (3D Analyst from ESRI, Creator Suite from Multigen-Paradigm, Google Maps, Microsoft Virtual Earth) and within the research community (GeoVista Studio, GeoViz) that address this issue.

Approaches to generating GIS visualizations can be broadly classified into two categories: (1) Simple and fast methods that require very limited visualization setup via a graphical user interface. Visualizations in this category can be generated in a few minutes, but flexible design is not possible. (2) Custom visualizations programmed with a low-level language like Java, Javabeans or C++. These visualizations can be customized, but the low level languages used do not offer many tools to simplify or speed up the customization process. There is thus a need for tools that allow flexibility in modeling, while producing GIS visualizations within reasonable time.

In this paper we describe a solution to the GIS Visualization modeling problem based on procedural modeling. The solution is aimed towards allowing domain experts to quickly generate complex visualizations of attribute data over a geospatial domain. We employ a shape grammar specified by a script-like language to author the visualizations. The shape grammar evolves a set of shapes [WWSR03, MWH*06] to represent GIS data. The shapes have geometric and numeric attributes. Production rules specify how a starting design (a set of shapes and attributes from the GIS database) can be evolved into a complex model complete with interaction possibilities. Our objective is to allow computer scientists, and more importantly, domain scientists with limited computer science background, to create and customize complex GIS visualizations. The main contributions of this paper are:

- This is the first paper to apply the concepts of procedural modeling to glyph based visualization of GIS data using shape grammars.
- Our method generates visualizations much faster than existing methods that rely on low-level programming.
- We demonstrate the effectiveness of our shape grammar with real-world examples.

† pushpak@asu.edu

Published by Blackwell Publishing, 9600 Garsington Road, Oxford OX4 2DQ, UK and 350 Main Street, Malden, MA 02148, USA.

2. Related Work and Preliminaries

2.1. Related Work in Geo-spatial Visualization

There is a large volume of work in visualization related to the presentation of geo-spatial data. Much of this work proposes new techniques for two and three-dimensional visualizations. Examples of two-dimensional visualizations include extended "focus+context" [FS04] and distorted two-dimensional maps [KNPS03]. Three-dimensional visualization of geo-spatial data has recently attracted much attention in research [DMK05, WKD*05]. Wood [Woo05] presents a case for using multi-scale 3D visualizations (levels-of-detail) for the display of multi-scale geographic data. The *GeoTime Information Visualization* system [KW04] displays temporal data over a geographic dataset to give the user a combined 3D view with time as one axis. Tominksi et. al. [TSWS05] describe a method to represent multivariate time-dependent data at proper geo-spatial coordinates. Their method relies on using the height as an axis for 3D polytopes, where each face describes the variations in one particular variable.

Data visualization and interaction has been a richly mined area of research (see [CMS99] for an extensive survey). However, there has been less effort in providing the backbone tools for modeling such visualizations and specifying modes of user interaction. One example is GlyphMaker [RAEM94], which includes an interactive editor to model glyphs. Another interesting approach to glyph modeling is the use of implicit surfaces [RDL98, DRC*99] or superquadrics [SHB*99]. In these approaches, multi-dimensional abstract data are mapped to the parameters of the mathematical surface description. Mackinlay [Mac86] describes a system to create 2D visualization objects such as pie charts and bar graphs.

There exist commercial and research solutions for modeling GIS visualizations. Commercial products include the 3D Analyst extension to ESRI's ArcGIS, ArchiCAD GDL and the Creator Suite from Multigen Paradigm. These tools provide a fully functional scripting environment that allows manipulation of GIS data. However, they do not support advanced real-time visualization options like our system, and do not handle large datasets (> 4 GB) robustly. In addition, there has been a parallel development of visualization toolkits for geo-spatial data in academia. Most related examples include DEVise [LRB*97], GeoVista Studio [TG02] and GeoViz, InfoVis Toolkit [Fek04], Improvise [Wea04], and Prefuse [HCL05].

InfoVis Toolkit and DEVise are excellent solutions for Information Visualization. However, they lack features to integrate geo-spatial data into their framework. GUI oriented tools [TG02, Wea04] for geo-spatial data visualization often perform a strict set of tasks very efficiently. However, if complete control over the modeling process is required, the GUI has to be complemented by low-level APIs and libraries [HCL05, TG02] such as Java or C++. This approach usually results in a time consuming modeling phase.

In addition, three-dimensional city model generation is an active research topic [PM01, KG03, HC05, MWH*06, LWW08]. Such tools provide an architectural solution for modeling cities. However, the main difference between these approaches and our method is that the above approaches strive for authentic city modeling in 3D, while our goal is to visualize abstract GIS attribute data over the same geospatial domain. We do not have a priori knowledge of the 3D geometry like buildings or landmarks, but work from the 2D footprint data or point data to place glyphs at the appropriate locations over the geospatial terrain.

2.2. Related Work in Procedural Modeling

Many procedural techniques were developed in the context of modeling plants and architecture. For plant modeling, Prusinkiewicz and Lindenmayer showed that impressive results can be achieved by using L-systems [PL91, PJM94]. In architecture, shape grammars [Sti75, Sti80] were successfully used for the construction and analysis of architectural design [DF81, Dua02]. Shape grammars and generative modeling can also be used to model architecture for computer graphics [WWSR03, MWH*06]. While we use many ideas of existing grammars in this paper, we augment our grammar with extra capabilities needed in the context of geo-spatial visualization.

2.3. Shape Grammars

Here we review the basic concepts of shape grammers, drawing much of the discussion from *CGA Shape* [MWH*06] and L-Systems [PL91]:

Shape: A shape grammar works by constructing a configuration of shapes, each consisting of a symbol (string), geometry, and numeric attributes. Shapes are identified by their symbols which are either *terminals* or *non-terminals*. The most important geometric attributes are the position P, three orthogonal vectors X, Y, and Z, describing a local coordinate system, and a size vector S. These attributes define an oriented bounding box in space called the *scope* (see figure 1).

Production process: A configuration is a finite set of basic shapes. The production process starts with an initial configuration A, called the *axiom*, and proceeds as follows: (1) Select an *active* shape containing a non-terminal B, (2) choose a production rule with B on the left hand side to compute a new set of active shapes B_{NEW} (the successor for B), (3) add the active shapes B_{NEW} to the configuration and mark the shape B as *inactive*. Finally, return to step (1). When the configuration contains no more active non-terminals, the production process terminates.

Rule priorities: Each of the rules is assigned a priority to determine the rule selection order in step (1). Priorities are

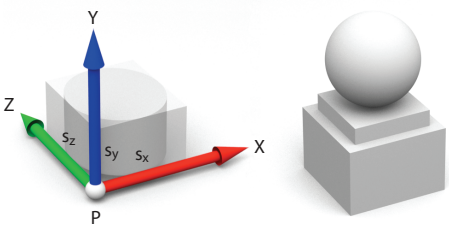


Figure 1: *Left: The scope of a shape. Right: A three-dimensional glyph composed of three shape primitives.*

specified by the user with the optional "PRIORITY" keyword. If this keyword is not specified for a rule, the rule inherits the priority level of a preceding rule that has a well-defined priority. The first rule in the file is assigned a default priority value 0. Any user defined priority value overrides this default priority value. Rules belonging to the same priority level are processed before any rule from subsequent priority levels is applied. This guarantees that the derivation proceeds from low detail to high detail. Rules within a priority group are processed in the order in which they are listed.

Notation: Production rules are defined in the following form:

PRIORITY *number*
id: predecessor : cond $\rightsquigarrow$ successor

where *id* is a unique identifier, *predecessor* is a non-terminal identifying a shape that is to be replaced with *successor*, *cond* is a guard (logical expression) that must be true in order for the rule to be applied, and *number* is the rule priority as defined by the "PRIORITY" keyword (optional). For example, the rule

1: well(depth) : **depth** $>$ **60** $\rightsquigarrow$ cylinder(9*depth/10) top(depth/10)

replaces the shape *well* with two shapes *cylinder* and *top*, if the parameter *depth* is greater than 60.

The condition (logical expression) acts as a switch for processing rules that begin with the same non-terminal and have the same priority level. Rules that begin with the same non-terminal and have the same priority are processed in the sequence that they occur in the file, and the first matching rule is applied for generation of next level of shapes. The "*default*" keyword is used to specify the action to be taken if all other specified conditions are not satisfied. For example, the second of the following rules

1: well(depth) : **depth** $>$ **60** $\rightsquigarrow$ cylinder(9*depth/10) top(depth/10)
2: well(depth) : *default* $\rightsquigarrow$ cube(9*depth/10)

creates a cube without a *top*, for the wells whose *depth* value is not greater than 60.

3. Overview

Figure 2 shows an overview of our system. We store the data to be visualized in a *GIS database*. A user creates a visualization using a *rule editor* and a *user interface editor*. The rule editor allows the specification of rules that define the modeling of geometry, appearance, and interaction. They are the focus of the paper and will be explained in the following sections. In contrast, the user interface editor directly defines the overall visualization method, for example, the number of views for overview and detail. Possible views are 2D orthographic, three-dimensional oblique, or three dimensional perspective. After the user has finished modeling, the *grammar engine* parses the rules and interfaces with the GIS database to create a visualization. The user can then navigate through the visualized data and apply interaction techniques such as focus + context.

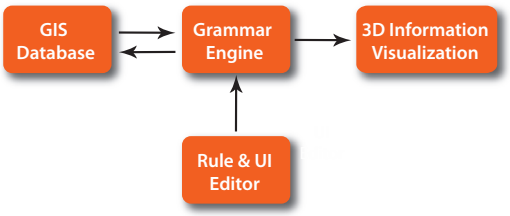


Figure 2: *The GIS Procedural Modeling System.*

Geo-spatial information is typically processed by GIS, and stored in layers. Each layer represents a particular geographic feature type, such as roads, city locations or land use boundaries. The data within each layer contains a set of *records*, each with spatial (point, poly-line or polygon) and abstract information attributes (multi-dimensional, geometric, or scalar attributes). Additionally, some layers are stored as regularly sampled raster grids, for example, the water depths used by the well glyphs in figure 8.

4. A Shape Grammar for Visualization Authoring

Our GIS shape grammar models many different aspects of a 2D or 3D visualization, including initialization, shape attributes, spatial placement of glyphs, glyph separation, representation instancing, and interaction.

4.1. High Level Grammar Definition

At the most abstract level, our grammar produces a visualization of a *scene*. A scene is composed of one or more independent datasets called *layers*, which often correspond directly to the data layers in the underlying GIS database. The grammar engine processes the scene layer-by-layer in sequence. Actual geometry and appearance definition occurs via *rules* that are attached to layers as child elements. Our grammar supports three types of rule definitions: (1) Geometric placement rules (GPR), (2) Visual descriptor rules (VDR) and (3) General rules (GR). The rules themselves are composed of atomic commands, similar to the shape grammar approach by Müller et al. [MWH*06].

The geometric placement rules consist of built-in commands to perform instantiation of geometry, geometric transformations and glyph separation. The visual descriptor rules define the look-and-feel of glyphs and other visualization elements. They consist of built-in commands as well as user defined attribute maps. Examples include commands that specify the texture and color attributes of glyph geometry. "General rules" do not specify the geometry or visual appearance directly. They serve as useful extensions to the scripting engine itself. They can be built-in rules, user defined commands or hooks for user interaction.

A formal description of our grammar is given below. We also include the typical commands available in our implementation. A detailed description of these commands follows the formal definition of the grammar in section 4.2. While the grammar specification is quite compact, we have found it sufficient to produce a variety of useful visualizations. This basic set of rules described below could also be extended to provide additional functionality.

Formal Shape Grammar Description:

1: $axiom \rightsquigarrow scene$
2: $scene \rightsquigarrow [\ layer\ ]^*$
3: $layer \rightsquigarrow rule^*$
4: $rule \rightsquigarrow [\ \text{GPR} \mid \text{VDR} \mid \text{GR}\ ]^*$
5: $GPR \rightsquigarrow$ **T(** $floatVal, floatVal, floatVal$ **)** |
S($floatVal, floatVal, floatVal$ **)** |
R($floatVal, floatVal, floatVal$ **)** |
I($shape\ id \mid filename$ **)** |
6: $VDR \rightsquigarrow$ **Color(** RGB(r, g, b, a) $\mid texture\ filename$ **)**
7: $GR \rightsquigarrow$ **Repeat(** $min\_index, max\_index, rule\ id$ **)** |
Repeat($count, rule\ id$ **)** |
OnLeftClick($action\ id$ **)**
8: $floatVal \rightsquigarrow symbol \mid number$
9: $symbol \rightsquigarrow$ **ReadSymbolTableValue(** $symbol$ **)**
10: $number \rightsquigarrow floating\ point\ number \mid$
HeightFieldIntersect($floatVal, floatVal, objId$ **)** |
Separate($objId, axis, units$ **)**

4.2. Grammar Usage

A typical visualization authoring session involves the following steps: (a) The user specifies the data to be imported into the visualization by linking specific non-terminal symbols to database tables. (b) The user writes grammar rules that create and position data glyphs according to their geospatial location. (c) The grammar engine processes the rules and generates the visualization.

The grammar may include rules to place shapes above a terrain layer or resolve collisions between shapes. Custom attribute maps can be used to define the visual appearance of the glyphs created in the previous step. Other rules specify interaction hooks in the visualization.

4.2.1. Initialization

Scene: The scene is the top-most logical component of our modeling hierarchy. It is denoted by the "*axiom*" keyword in the rule specification file. The non-terminal symbol that follows the keyword "*axiom*" is treated as the first non-terminal to be processed. Rule derivation begins from this point onward until suitable terminal symbols are generated. For example, the statement *"axiom: S"* specifies that the grammar derivation will begin with the symbol *S*.

Layers: GIS visualizations are often constructed as a set of layers. Our grammar uses the special keyword "*layer*" to denote the non-terminals that are to be treated as layers. This provides a conceptual link between the database layers and the visualization of those layers specified in the grammar.

Tables: Our system reads data from a PostGIS database. It is necessary to make a mapping between the non-terminals used in the grammar specification and their logical counterparts in the database. This is achieved with the help of the keyword "*table*". This keyword is followed by the table name in the database. Upon encountering this table name as a non-terminal, the system proceeds to derive the child rules for every record in the table. For example, the non-terminal "*well*" can be hooked to the well table in the database with the statement *"table well"*. The grammar engine processes the non-terminal "*well*" by extracting the entire well table from the database. It then instantiates a temporary "*well*" symbol for each row of the table and searches the rule list (in order of priority) for a rule with *well* as the left hand side and a conditional compatible with the well data. If such a rule exists, the rule is applied. Otherwise, the system discards the temporary well.

4.2.2. Shape Attributes

Attribute types: The Shape Grammar processes shapes with associated attributes, which can be of two types: (1) Geometric attributes such as position P, local coordinate axes X, Y, Z, size vector S, RGB color vector C, opacity, texture, and texture projection. The geometric attributes determine the visual representation of the shape. (2) Auxiliary attributes that are provided by the GIS database or generated during derivation. During the derivation of the grammar, attributes can be specified or changed by rules of the form:

$A \rightsquigarrow attribute($ $expression$ $)$

where *attribute* is the attribute to be set and *expession* is a valid mathematical expression of the correct type. Non-terminals corresponding to database tables inherit all attributes in the database as auxiliary attributes.

Attribute Maps: The goal of our grammar is a visualization of attribute data associated with a geospatial location. The attribute data may not be inherently geospatial in nature. For example, it may be a set of string literal values to categorize the table records into different groups. A mechanism to convert this "raw" information into visual characteristics is

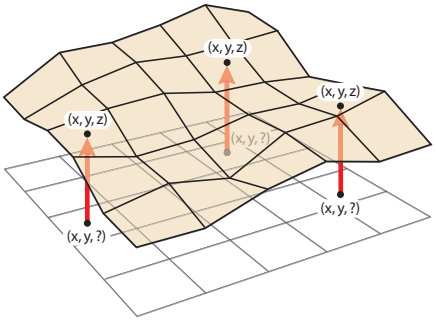


Figure 3: *HeightFieldIntersect example: The 2D wells are moved to the points where they lie on the terrain by calculating the intersection between the terrain and the well position.*

therefore needed. Attribute maps convert attribute data from their original type to another data type amenable to visualization. For example, the attribute map *ColorByWellType* in the well hydrological data example (section 5.1) transforms the well category to a color. It can be used anywhere a color would be used in the grammar, as in:

1: *wellAppearance* $\rightsquigarrow$ **Color**(*ColorByWellType*(*well.category*))

where *ColorByWellType* maps the discrete well group types (IRRIGATION, MUNICIPAL, etc.) to a set of colors. For discrete data, the attribute map is specified as a table in the input file. Numerical attributes can be converted to colors using the "*Gradient*" command, which maps a numerical range to a range of colors.

Mathematical expressions: Along with attribute maps, shape grammar rules may use mathematical expressions (+,-,*,/) as arguments. For example, the following rule scales the Y-coordinate of a well glyph by one fifth of the pumpage:

1: *wellAttributes* $\rightsquigarrow$ **S**(1, *well.pumpageCapacity*/5, 1)

4.2.3. Geometric Placement Rules (GPR)

Scope rules: Similar to L-systems we use scope rules to transform the scope of a shape. $T(t_x,t_y,t_z)$ translates the scope position, $Rx(angle)$, $Ry(angle)$ and $Rz(angle)$ rotate the coordinate system, and $S(s_x,s_y,s_z)$ scales the scope. We use '[' and ']' to push and pop the current scope on a stack. Any non-terminal symbol will be created with the current scope. Similarly, the command $I(objId)$ adds an instance of a geometric primitive with identifier *objId* within the current scope. Typical objects include cubes, spheres and cylinders, but any geometric model can be used. The example below illustrates the design of a simple glyph, similar to the one depicted in figure 1 right:

1: A $\rightsquigarrow$ [**T**(0,0,0) **S**(1,1,1) **I**("cube")]
[**T**(0,1,0) **S**(0.75,0.25,0.75) **I**("cube")]
[**T**(0,2,0) **S**(0.75,0.75,0.75) **I**("sphere")]

Intersection Queries: The *CGA Shape* grammar describes rules for translation, rotation, and scaling of its symbols. We extend this concept by introducing the rule *HeightFieldIntersect* that provides georeferencing for geometry placement. The *HeightFieldIntersect* command is a generic intersection command for calculating intersections between a 2D map point (specified by the x and y coordinates) and a 2.5D or 3D reference shape such as a height field or an isosurface, as shown in figure 3. The syntax for *HeightFieldIntersect* is

HeightFieldIntersect(*x*, *y*, *objID*)

where *x* and *y* are the x and y coordinates of the intersection, and *objID* is the id (a string) of the of the shape with which we wish to intersect. For example, given the well locations in 2D, and the underlying terrain layer denoted by the identifier *"Phoenix_terrain"*. The rule to place a well on this terrain is defined as:

1: *well* $\rightsquigarrow$ [**T**(*well.x*, *well.y*, *HeightFieldIntersect*(*well.x*, *well.y*, *"Phoenix_terrain"*)) *wellAppearance*]

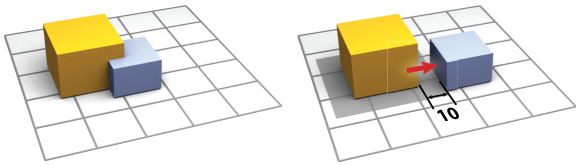


Figure 4: *Glyph Separation example. The* garage *(shown in blue) is moved along the X-axis so that it does not overlap with the* house *(shown in orange).*

Glyph Separation: We employ explicit glyph separation rules to handle overlapping shapes. Collisions are detected by comparing the bounding box of an incoming shape with shapes in the current derivation. The *Separate* command determines a vector to resolve collisions between the current scope and the scopes of other shapes in the derivation. It has syntax:

Separate(*B*, *direction*, *offset*)

where *B* is the shape we are comparing against, *direction* is the direction in which to resolve the collision, and *offset* is the desired separation distance. For example, the rule

1: *garage* $\rightsquigarrow$ **T**(*Separate*(*house*, (1,0,0), 10))

will move the symbol *garage* along the X-axis such that it does not occlude the *house* geometry (see figure 4) and lies 10 units away from its bounding box. Note that this does not cause a cascade of collision detection checks since each collision check must be explicitly mentioned in a rule. While this method does not account for all possible collisions, it provides a useful, fast and predictable way to resolve many overlaps. A more comprehensive collision detection scheme remains an area for future work; however, automatic layout with full collision resolution is known to be NP-complete [ECMS97].

4.2.4. Visual Descriptor Rules (VDR)

Color and texture: The "*Color*" command changes the color associated with a shape. *Color* is also used to apply a texture to a shape. For example, $Color(RGB(1,0,0))$ sets the color of the current scope to red, and $Color(texture$ "$image.jpg$"$)$ assigns the file "*image.jpg*" as a texture to the current scope. Colors can also be specified in other ways. Key words exist in the grammar for common colors such as *WHITE* and *BLACK*, and colors can be indexed through attribute maps, as described in section 4.2.2. In the future we would like to add other visual attributes to our system, such as glows for highlighting glyphs, animated textures, and various NPR shading styles.

At this point, we have enough rules to give a slightly larger example. The following rules create cylindrical glyphs for elements of the *well* table from the database:

1: *well* $\rightsquigarrow$ [**T**(*well.x*, *well.y*, *HeightFieldIntersect*(*well.x*, *well.y*, ("*Phoenix_terrain*")) *wellAppearance*]
2: *wellAppearance* $\rightsquigarrow$ **Color**(ColorByWellType(*well.wellType*)) *wellAttributes*
3: *wellAttributes* $\rightsquigarrow$ **S**(10, *well.pumpageCapacity*, 10) **I**("*cylinder.obj*")

The glyphs are placed above the terrain (rule 1), colored by their type (rule 2), and scaled according to pumpage capacity (rule 3). Figure 5 shows the result of applying these three rules.

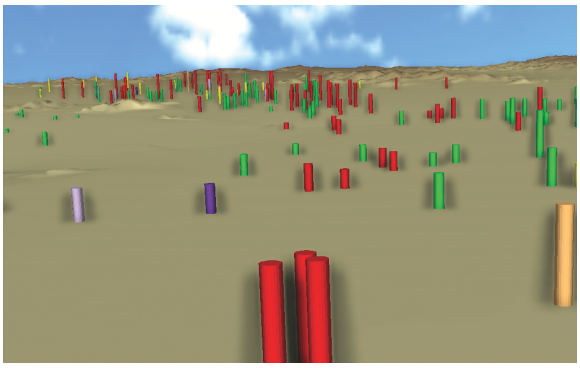


Figure 5: *The wells shown as cylinders scaled according to pumpage.*

4.2.5. General Rules

Repeat command: The *Repeat* command applies a derivation multiple times (similar to a "for" loop). The built-in variable "!index" defines the iteration number either as a range or as an absolute value (by convention the prefix "!" identifies internal variables):

A $\rightsquigarrow$ **Repeat**(*min_index*, *max_index*, B)
A $\rightsquigarrow$ **Repeat**(*repeat_count*, B)

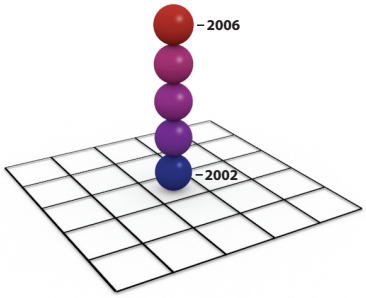


Figure 6: *Repeat command example showing a stacked glyph at the well location.*

where *A* and *B* are non-terminals. The *Repeat* command is useful to model time varying attributes of a geo-spatial location. For example, the variation in the annual pumpage for a well from 2002 to 2006, might be modeled using the rule:

1: pumpageValues $\rightsquigarrow$ **Repeat**("2002", "2006", pumpagePerMonth)
2: pumpagePerMonth $\rightsquigarrow$ [**T**(*well.x*, *well.y* * 2.5 * !*index*, *well.z*) pumpageAppearance]
3: pumpageAppearance $\rightsquigarrow$ **Color**(ColorFromPumpage(*well.monthly_pumpage*[!*index*])) pumpageAttributes
4: pumpageAttributes $\rightsquigarrow$ **S**(1,1,1) **I**("sphere.obj")

The rules above generate a "stack" of spheres at the well location (see figure 6) that show the annual pumpage values as colors.

Interaction Commands: The interaction commands define user interactivity with the generated model. They allow the grammar to set up hooks for processing user-interface events like mouse button clicks, key presses, etc. The derivation of the specified non-terminal defines the system's response to the interaction, and the derivation is deferred until the event occurs. For example, the rule

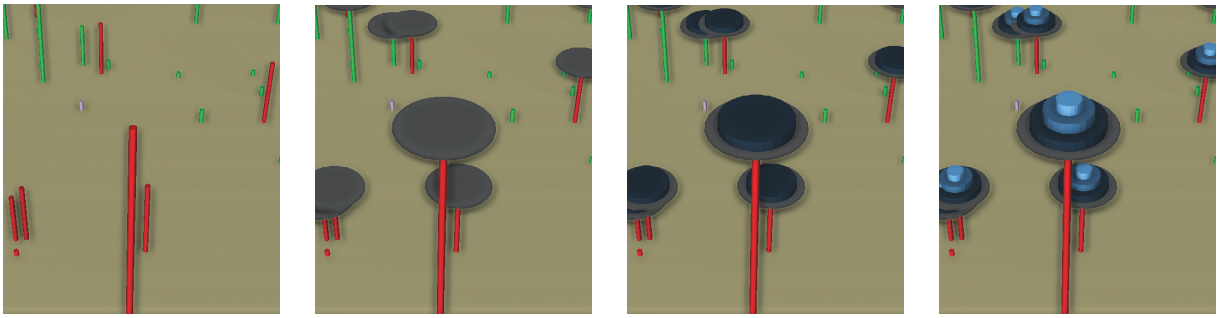
1: *well* $\rightsquigarrow$ *OnLeftClick*(*DisplayMetadata*(*well*))

calls the user defined routine *DisplayMetadata* that displays information about the well, once the user has clicked on it.

5. Visualization Examples

We chose three instances of real-world GIS problems from related research projects to test our system. For each we structure the description in three parts. First, we give a brief explanation of the problem domain. Second, we explain the visualization concepts employed for our solution. Third, we show how the solution can be modeled with our grammar rules.

Each of the examples demonstrates the ability of our framework to work with different kinds of GIS data. The first example processes 2D point data on a terrain (well locations), the second works with area features (parcels and lots), and the third example demonstrates visualization of time-varying data over a geospatial domain.



(a) (b) (c) (d)

Figure 7: *Step-by-step derivation of the glyphs as produced by the rules in Figure 11. (a) Create stem on well site based on well depth (rule 11). (b) Add cylinders to indicate pumpage (rule 12). (c,d) Add disks to indicate intersections with alluvial layers (rules 13-16).*

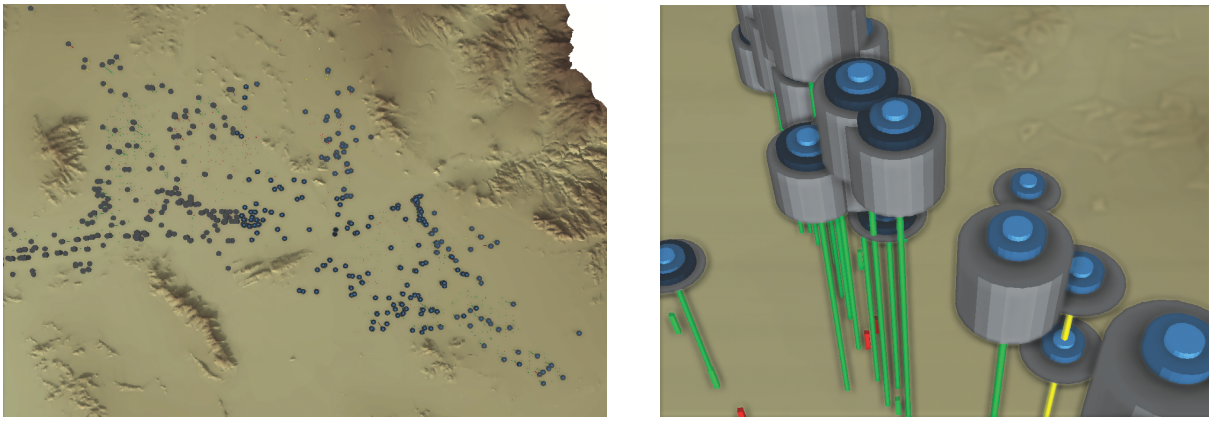


Figure 8: *(Left) Overview of the wells. (Right) A close-up 3D view of the well glyphs. The glyphs display the well depth as height above the terrain, pumpage as the volume of a cylinder, and alluvial layers intersected by the well as disks above the pumpage cylinder.*

5.1. Example 1: Hydrological Data

Problem Statement: The first example stems from an underground water management project in the greater Phoenix area. There are three distinct alluvial layers in which underground water is stored. Water is pumped via wells of different bore depth and pumpage capacity. Water management is typically concerned with maintaining and monitoring the water quality and levels in the three layers.

Uncontrolled pumping from localized regions in the aquifer can lead to land subsidence and have an adverse effect on the water quality, and structural stability of physical landmarks on the surface. Hence pumpage must be controlled carefully. The goal of our visualization is to provide water managers with a tool to analyze both local and global water supplies and the effect of pumping over time.

Visualization Solution: The challenge in this context is to visualize the wells and their attributes together with the alluvial layers and the terrain. A water manager needs to visually cluster the well data and detect negative impacts of pumpage to the water surface, both in terms of water depth and the gradient of the water surface at a well location. We construct three-dimensional glyphs to better visualize the involved surfaces, the well depth and the spatial relationships. The data used in this example consists of a layer of well data that includes (1) the well locations, (2) pumping capacity per well, (3) well types, and (4) the alluvial layers intersected by each of the wells. The alluvial layers themselves are available as height fields. The surrounding terrain is displayed as a height field with ground texture.

The fairly high number of wells leads to a cluttered visualization. Therefore, we employ selection strategies to represent only wells of interest as detailed glyphs while other wells are modeled as simple cylinders. In the example, we model wells with pumpage values lower than 1000, or greater than 40,000 gallons per day as detailed glyphs. The glyphs show the pumpage, bore depth and alluvial layers from which the wells pump in an easily vieiwable form.

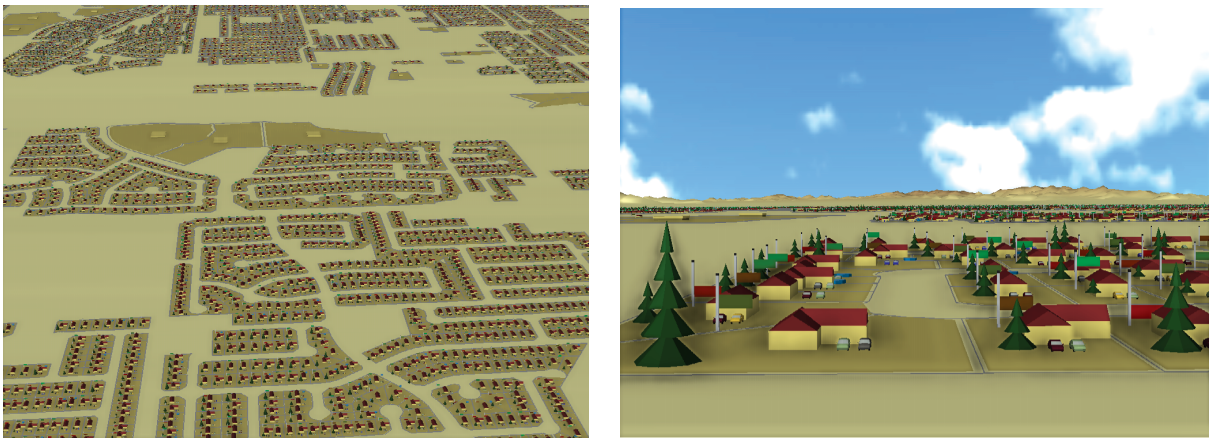


Figure 9: *Left: House glyphs visualization overview. Right: A closeup view that shows the houses in a neighborhood.*

Figure 7 shows the step-by-step derivation of the detailed well glyphs for this visualization. Figure 8 give screen shots of the complete visualization.

Grammar Description: Figure 11 shows the grammar rules used by our system to to create the well visualization example, along with descriptions of the rules. Much of the power of our system lies in its ability to blend database queries with grammatical derivations in arbitrary ways. This makes our technique more flexible than methods that only provide a few fixed mappings between database tables and the visual and geometric attributes of a visualization.

5.2. Example 2: Home Browsing

Problem Statement: This example is inspired by the HomeFinder [WS99] application and quality-of-life indices [BS81, Rog99]. The HomeFinder allowed novice users to make database queries based on data that could be relevant to purchasing a house, but provided only a simple visual representation of the query results. In this example, we attempt to give a richer visualization of real estate data than was provided by HomeFinder, incorporating quality-of-life indices as well as assessor's data for individual house lots. The main problem in visualizing such data lies in the fact that the metadata comes from disparate sources and needs to be combined in a meaningful fashion to enable the user to explore the data visually. We are not just looking for overall data trends. Rather, we want a user to be able to see the data and geographic context for the individual households within a neighborhood to help assess the suitability of a particular house for a potential buyer.

Visualization Solution: The challenge of this visualization is conveying attribute data, while placing them in a three-dimensional geo-spatial context. The proposed solution gives a good estimation of certain attributes from an overview, while allowing details to be seen in a view from near ground level (see figure 9). Thus it is possible to couple inherently three-dimensional aspects, such as building sizes, street width, and view of the mountains together with attribute data. The data used in this example is a mix of census data, assessor's data, and output of an agent-based urban simulation.

Grammar Description: Starting with a parcel, our detail view recursively generates glyphs for each contained lot. We map the following variables per lot: annual income, number of cars in the household, area of pool (if present), the improved fixed cash value of the lot and change in the improved fixed cash value over the previous year.

5.3. Example 3: Traffic Data for City of Tempe, AZ

Problem Statement: Traffic forecasting and accident analysis are two important processes in city development. Traditionally, cities publish annual accident reports, and these data are taken into account when planning new roads. Construction projects also contribute to this decision making process. Traditional means of interpreting such data involve deriving correlations between different data sources. Although tools exist to overlay the data in map views, providing extra information like traffic flow, presence of traffic lights etc. can be quite cumbersome. Our modeling framework can build 'scenarios' based on actual measured and simulated data.

Visualization Solution: Our model consists of two parts, (a) the static geometry and (b) the dynamic metadata which varies over time. We base our glyphs on the space-time path as described by Tominski et. al. [TSWS05] (Figure 10). The spheres at every intersection denote the accidents per month.

The second step in the visualization involves calculating the intersections given some geographical constraint (the so-called *map focus+context* [FS04]). For instance, we are interested in locating all the intersections within the city of Tempe. At every intersection, we also place a "cross" symbol that shows the traffic congestion at that intersection visually. The symbol is scaled proportional to the congestion,

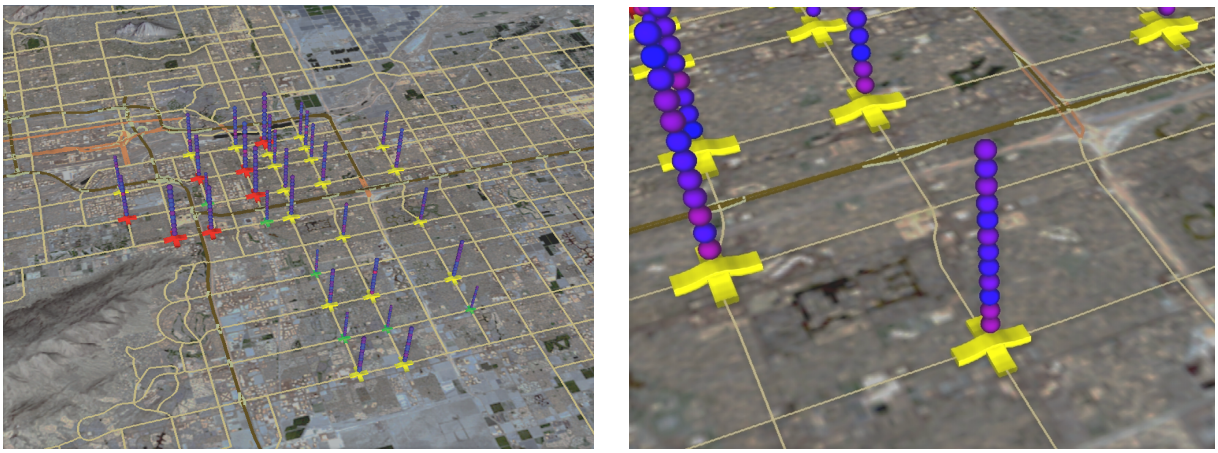


Figure 10: *An example of integrating static and dynamic data for traffic analysis. Data is visualized at the intersections within a given geospatial extent (in this case, city of Tempe, AZ).*

and it is colored with one of the three color values (Red, Yellow or Green) depending on the traffic congestion. Because of space constraints, we have not included the grammar rules for this example, but figure 10 shows screen shots of the visualization.

6. Discussion

Implementation: Our grammar engine is implemented in C++. We collected GIS data as shape files containing the geometry and associated metadata (as columns) from various projects using ArcGIS software. The GIS data are stored in a PostgreSQL database server and we use the PostGIS interface for communication. We use the OpenSceneGraph library as our rendering engine.

We have chosen to write a stand-alone prototype of our system, but an existing scripting language like Python or Visual BASIC could also be used as an underlying layer for our grammar framework. These scripting languages do not supply geospatial commands by default, and would therefore need to be extended with an implementation of our framework that supplies the special functionality of scope control, glyph generation and remote querying to a geo-spatial database.

Derivation Times: Table 1 shows the derivation time and polygon count for each of our examples. The timings were averaged over 15 runs of each example. While not real-time, these derivation times are fast enough to allow for iterative refinement of the grammar rules during visualization authoring.

Scripting vs. GUI interfaces: We believe that our modeling framework is a good fit for geo-spatial data exploration. While our system requires grammar specifications, this level of computer knowledge is required from most practitioners in geographic information systems (scripting is part of all major GIS systems and is required even for introductory GIS courses). Once a user becomes familiar with the scripting language, complex visualizations can be quickly created for a wide variety of projects. At the same time, some parts of the visualization authoring, in particular glyph design, could be aided by a GUI rule editor similar to the one described in [LWW08].

Example	Polygon count	Model generation time (secs)
Wells	2,024,342	8.4
Home Browsing	15,785,645	98.5
Roads	146,850	56

Table 1: *Derivation times for the examples. The second column shows the polygons generated in the modeling process, and the third column shows the CPU time (in seconds) taken by the engine to generate the scene. The timings are an average of 15 runs of each example.*

7. Conclusion

This paper proposes a new method for procedural modeling geo-spatial visualizations. We employ shape grammars that can produce detailed visual models using similar concepts from the *CGA Shape* grammar of Müller et. al. [MWH*06]. However, our system solves a different problem than the creation of visually plausible city model. Instead, we model the visualization of abstract metadata over a geospatial domain.

In general, a procedural modeling system has to be adapted to the problem at hand. Therefore, we adopted some of the basic ideas from previous work, such as the turtle style commands, but also added commands to access geo-

Rule	Description
layer terrain	Define the layers in the visualization
layer waterSurface	(terrain, waterSurface, well).
layer well	
axiom: S	Define the axiom of the grammar.
table well	Specifies "*well*" as a table in the GIS database. The non-terminal "*well*" initiates a database query.
PRIORITY 1 // Create Visualization Layers	
1: S ⇝ [terrain][waterSurface][well]	Derivation of the scene layers from the axiom.
2: terrain ⇝ **I**(*"terrain.obj"*)	Load the terrain object.
3: waterSurface ⇝ waterSurfaces	Load the water surface objects.
4: well : (*well.pumpage* < 1000 **OR** *well.pumpage* > 40000) ⇝ *OnLeftClick*(metaData) detailedGlyph	Rules 4 and 5 are processed for every record of the "*well*" table from the database. Rule 4 creates detailed glyphs for wells with pumpage between 1000 and 40000. Rule 5 creates simple glyphs for other wells.
5: well : **default** ⇝ *OnLeftClick*(metaData) overviewGlyph	
PRIORITY 2 // Load the water surface objects	
6: waterSurfaces ⇝ [waterSurf1] [waterSurf2] [waterSurf3]	
7: waterSurf1 ⇝ **I**(*"water_surface1.obj"*)	Load the individual water surfaces.
8: waterSurf2 ⇝ **I**(*"water_surface2.obj"*)	
9: waterSurf3 ⇝ **I**(*"water_surface3.obj"*)	
PRIORITY 3 // Create Overview Glyph	
10: overviewGlyph ⇝ **T(** *well.x*, *well.y*, *HeightFieldIntersect*(*well.x*, *well.y*, *"terrain"*)) **Color(** RGB(*ColorByWellType*(*well.category*))) **S**(50.0, 50.0, *well.pumpage*) **I**(*"cylinder.obj"*)	*HeightFieldIntersect* places each well on the correct *z*-value on the terrain. Color the well according to its type. Create a cylinder at the well location scaled by 50 units in *x* and *y* and by the annual pumpage value in *z*.
PRIORITY 4 // Begin Detailed Glyph	
11: detailedGlyph ⇝ **T(** *well.x*, *well.y*, *HeightFieldIntersect*(*well.x*, *well.y*, *"terrain"*)) **Color(** RGB(*ColorByWellType*(*well.category*))) **S(** 50, 50, 10.0 * *well.depthInMeters*,) **I**(*"cylinder.obj"*) level1	Begin derivation of the detailed glyph. Translate the glyph to the well location. Apply color based on the well type. Scale the well cylinder in *z* in proportion to its depth below the surface.
PRIORITY 5 // Detailed Glyph: Add Cylinder Based On Pumpage Value	Rule 12 places another cylinder on top.
12: level1 ⇝ **T(** 0, 0, 1.0) **Color(** RGB(127, 127, 127)) **S**(60.0, 60.0, *well.pumpage*) **I**(*"cylinder.obj"*) level2 level3 level4	Move (the scope) to the top of the well cylinder. Create a cylinder scaled proportional to the pumpage value
PRIORITY 6 // Detailed Glyph: Add Cylinders for Water Surface Intersections.	
13: level2 :*HeightFieldIntersect*(*well.x*, *well.y*, *"waterSurf1"*) **!= NULL** ⇝ RGB(28,50,69) topCylinder	Rules 13-15 create a stack of scaled cylinders based on what water surface the well intersects. The rules are executed serially, and the successor symbols are derived if the condition is true.
14: level3 :*HeightFieldIntersect*(*well.x*, *well.y*, *"waterSurf2"*) **!= NULL** ⇝ RGB(65,117,162) topCylinder	
15: level4 :*HeightFieldIntersect*(*well.x*, *well.y*, *"waterSurf3"*) **!= NULL** ⇝ RGB(103,187,255) topCylinder.	
PRIORITY 7 Add a Scaled Cylinder on top	
16: topCylinder ⇝ **T(** 0.0, 0.0, 1.0) **S(** 0.8, 0.8, 1.0) **I**(*"cylinder.obj"*)	
PRIORITY 8 // Display Metadata on Click	
17: metaData ⇝ *DisplayMetaDataForObject*(*well*)	Rule processed whenever the user clicks on a well.

Figure 11: *Rules for modeling glyphs for visualization of hydrological data for the Phoenix Metropolitan Area.*

spatial information, perform intersection queries, resolve glyph collision, and map abstract attributes to geometric entities. These new modeling commands and their application are the main contribution of this paper. We incorporated concepts from visualization and demonstrated that our modeling framework is capable of integrating several existing information visualization concepts, including information stacking, focus and context, level-of-detail, and temporal animation of attribute data. The result is a method that provides visualization designers with a tool set to define and extend visual representations of abstract attributes over a geo-spatial domain.

Our framework is flexible enough to be implemented on top of an existing scripting language like Python or VisualBASIC. This will allow domain scientists who are familiar with the existing languages to incorporate functionality provided by our toolkit into their existing applications seamlessly.

In the future, we would like to further test the effectiveness of our shape grammars with participation from domain scientists. We are also interested in expanding our solution to other GIS related problems such as the dynamic placement of labels on maps [BDY06, AHS05] and clustering algorithms for geo-spatial visualizations [Ray99].

table lot
Axiom: parcel

PRIORITY 1: // Create Lots
1: parcel ⇝ [lot]

PRIORITY 2: // Move To Lot Location Above Terrain
2: lot ⇝ **T(** *lot.centroid.x*, *lot.centroid.y*,
HeightFieldIntersect(*lot.centroid.x* , *lot.centroid.y*, *"terrain"*))
lotAppearance

PRIORITY 3: // Place Lot Boundary
3: lotAppearance ⇝ **Color(**RGB(0.42, 0.21, 0.05))
S(0.75, 0.75, 1.0) **I(** *lot.boundary*)
massModel
4: massModel: *lot.puc* **== 'SFR'** ⇝ houseModel
[cars] [pool] [cashValue] [priceFlag]
5: massModel: *lot.puc* **== 'APT'** ⇝ aptModel
6: houseModel: *lot.income* **> 120000.0** ⇝ **I(***"house_big.obj"***)**
7: houseModel: *default* ⇝ **I(***"house_small.obj"***)**
8: aptModel: ⇝ **I(***"apt_model.obj"***)**

PRIORITY 4: // Cars
9: cars: *lot.delta_x* **>** *lot.delta_y* ⇝ **T(**6.0, -5.0, 0.0) carPlacement
10: cars: *default* ⇝ **T(**2.5, -10.0, 0.0**)** carPlacement
11: carPlacement ⇝ [**Repeat(**1,*numberOfCars*, carModel **)**]
12: carModel ⇝ **T(***Separate*(*lot*, (1,0,0)))
I(*CAR*[RAND(1,5)])

PRIORITY 5: // Pool
13: pool:*lot.pool* **== true** ⇝ **S(** 2, *lot.poolArea*, 2)
T(*Separate*(*lot*,(1,1,0)))
Color(RGB(0.219, 0.863, 0.921) **)**
I(*"cylinder.obj"* **)**

PRIORITY 6: // Cash Value Glyphs
14: cashValue ⇝ **T(** *lot.deltax* * 0.25, *lot.deltay* * 0.25, 0.0)
[**Repeat(** 1, *getCountFromFCV*(*lot.fcv*), cashValueGlyph **)**]
15: cashValueGlyph ⇝ **S(** scale_values[!*index*] * 1.5 **)** **T(** 0,0,-2)
Color(RGB(0.145, 0.337, 0.0901) **)** **I(** *"cone.obj"* **)**

PRIORITY 7: // Change in FCV over previous year
16: priceFlag ⇝ **T(** *-lot.deltax**0.25, *lot.deltay**0.25, 0)
I(*"flag_base.obj"* **)** flag
17: flag ⇝ (*changeFactor* = (*lot.fcv*-*lot.prev_fcv*)/*lot.prev_fcv*)
T(0.0, *getHeightFromRange*(-1,1, *changeFactor*) , 0.0)
Color(RGB(*Gradient*(*RED*, *GREEN*, *changeFactor*)))
I(*"flag.obj"* **)**

Figure 12: *Grammar rules for the Home Browser example.*

References

[AHS05] ALI K., HARTMANN K., STROTHOTTE T.: Label layout for interactive 3D illustrations. *Journal of the WSCG 13* (January 2005).

[BDY06] BEEN K., DAICHES E., YAP C.: Dynamic map labeling. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (2006), 773–780.

[BS81] BOYER R., SAVAGEAU D.: *Places Rated Almanac: Your Guide to Finding the Best Places to live in America*. Rand McNally, Chicago, 1981.

[CMS99] CARD S., MACKINLAY J. D., SCHNEIDERMANN B.: *Readings in Information Visualization: Using Vision To Think*. Morgan Kaufmann Series in Interactive Technologies. Morgan Kauffmann Publishers, Inc., San Francisco, CA, 1999.

[DF81] DOWNING F., FLEMMING U.: The bungalows of buffalo. *Environment and Planning B 8* (1981), 269–293.

[DMK05] DYKES J., MACEACHREN A. M., KRAAK M.-J.: *Exploring Geovisualization*. International Cartographic Association. Elsevier Science, February 2005.

[DRC*99] DAVID E., RANDALL R., CHRISTOPHER S., PRADYUT P., JAMES K., D. R.: Procedural shape generation for multi-dimensional data visualization. In *Data Visualization* (Berlin, 1999), Springer-Verlag, pp. 3–12.

[Dua02] DUARTE J.: *Malagueira Grammar – towards a tool for customizing Alvaro Siza's mass houses at Malagueira*. PhD thesis, MIT School of Architecture and Planning, 2002.

[ECMS97] EDMONDSON S., CHRISTENSEN J., MARKS J., SHIEBER S.: A general cartographic labeling algorithm. *Cartographica 33*, 4 (1997), 13–23.

[Fek04] FEKETE J.-D.: The infovis toolkit. *infovis 00* (2004), 167–174.

[FS04] FUCHS G., SCHUMANN H.: Visualizing abstract

data on maps. In *Eighth International Conference on Information Visualization* (July 2004), pp. 139 – 144.

[Gah05] GAHEGAN M.: *Beyond Tools: Visual Support for the Entire Process of GIScience*. Elsevier Science, on behalf of International Cartographic Association, 2005, ch. 4, pp. 83 – 99.

[HC05] HAIST J., COORS V.: The W3DS-interface of Cityserver3D. In *Next Generation 3D City Models. Workshop Papers : Participant's Edition* (2005), Kolbe G., (Ed.), European Spatial Data Research (EuroSDR), pp. 63–67.

[HCL05] HEER J., CARD S. K., LANDAY J. A.: Prefuse: a toolkit for interactive information visualization. In *CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems* (New York, NY, USA, 2005), ACM Press, pp. 421–430.

[KG03] KOLBE T. H., GRÖGER G.: Towards unified 3D city models. In *Proceedings of the ISPRS Comm. IV Joint Workshop on Challenges in Geospatial Analysis, Integration and Visualization II* (Stuttgart, September 2003).

[KNPS03] KEIM D. A., NORTH S. C., PANSE C., SCHNEIDEWIND J.: Visualizing geographic information: Visualpoints vs Cartodraw. *Information Visualization 2*, 1 (2003), 58–67.

[KW04] KAPLER T., WRIGHT W.: Geotime information visualization. In *Proc. of the IEEE Symposium on Information Visualization* (Washington, DC, USA, 2004), IEEE Computer Society, pp. 25–32.

[LRB*97] LIVNY M., RAMAKRISHNAN R., BEYER K., CHEN G., DONJERKOVIC D., LAWANDE S., MYLLYMAKI J., WENGER K.: Devise: integrated querying and visual exploration of large datasets. In *Proc. of the 1997 ACM SIGMOD* (1997), ACM Press, pp. 301–312.

[LWW08] LIPP M., WONKA P., WIMMER M.: Interactive visual editing of grammars for procedural architecture. *ACM Transactions on Graphics 27*, 3 (2008).

[Mac86] MACKINLAY J.: Automating the design of graphical presentations of relational information. *ACM Trans. Graph. 5*, 2 (1986), 110–141.

[MWH*06] MÜLLER P., WONKA P., HAEGLER S., ULMER A., GOOL L. V.: Procedural modeling of buildings. *ACM Trans. Graph. 25*, 3 (2006), 614–623.

[PJM94] PRUSINKIEWICZ P., JAMES M., MĚCH R.: Synthetic topiary. In *Proceedings of ACM SIGGRAPH 94* (July 1994), Glassner A., (Ed.), ACM Press, pp. 351–358.

[PL91] PRUSINKIEWICZ P., LINDENMAYER A.: *The Algorithmic Beauty of Plants*. Springer Verlag, 1991.

[PM01] PARISH Y. I. H., MÜLLER P.: Procedural modeling of cities. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2001), ACM Press, pp. 301–308.

[RAEM94] RIBARSKY W., AYERS E., EBLE J., MUKHERJEA S.: Glyphmaker: Creating customized visualizations of complex data. *Computer 27*, 7 (1994), 57–64.

[Ray99] RAYSON J. K.: Aggregate towers: Scale sensitive visualization and decluttering of geospatial data. In *Proc. of the 1999 IEEE Symposium on Information Visualization* (1999), IEEE Computer Society, p. 92.

[RDL98] RANDALL R., DAVID E., L. S. J.: The shape of shakesphere: Visualizing text using implicit surfaces. In *IEEE Symposium on Information Visualization* (October 1998), IEEE Computer Society Press, pp. 121–129.

[Rog99] ROGERSON R. J.: Quality of life and city competitiveness. *Urban Studies 36*, 5-6 (1999), 969–985.

[SHB*99] SHAW C. D., HALL J. A., BLAHUT C., EBERT D. S., ROBERTS D. A.: Using shape to visualize multivariate data. In *Eighth ACM International Conference on Information and Knowledge Management* (New York, NY, USA, 1999), ACM Press, pp. 17–20.

[Sti75] STINY G.: *Pictorial and Formal Aspects of Shape and Shape Grammars*. Birkhauser Verlag, Basel, 1975.

[Sti80] STINY G.: Introduction to shape and shape grammars. *Environment and Planning B 7* (1980), 343–361.

[TG02] TAKATSUKA M., GAHEGAN M.: Geovista studio: A codeless visual programming environment for geoscientific data analysis and visualization. *Comput. Geosci. 28*, 10 (2002), 1131–1144.

[TSWS05] TOMINSKI C., SCHULZE-WOLLGAST P., SCHUMANN H.: 3D information visualization for time dependent data on maps. In *Proc. of 9th International Conf. on Information Visualization* (2005), pp. 175– 181.

[Wea04] WEAVER C.: Building highly-coordinated visualizations in Improvise. *infovis 00* (2004), 159–166.

[WKD*05] WOOD J., KIRSCHENBAUER S., DÖLLNER J., LOPES A., BODUM L.: *Using 3D in Visualization*. Elsevier Science, 2005, ch. 14, pp. 295 – 312.

[Woo05] WOOD J.: *Multim im parvo - Many Things in a Small Place*. Elsevier Science, 2005, ch. 15, pp. 313 – 324.

[WS99] WILLIAMSON C., SHNEIDERMAN B.: The dynamic homefinder: evaluating dynamic queries in a real-estate information exploration system. 125–139.

[WWSR03] WONKA P., WIMMER M., SILLION F., RIBARSKY W.: Instant architecture. *ACM Transactions on Graphics 22*, 3 (2003), 669–677.