URBAN DESIGN AND PROCEDURAL MODELING

Peter Wonka
Arizona State University
http://www.public.asu.edu/~pwonka

Pascal Müller
ETH Zurich
http://www.vision.ethz.ch/pmueller

Ben Watson
NC State University
http://cde.ncsu.edu/watson

Andy Fuller
Electronic Arts, Inc.
http://www.ea.com

Problem: The Content Challenge

- *More and better content is desperately needed*
  - Display capabilities improve (next-gen platforms etc)
  - Audience expectations grow
- Architectural content like cities and buildings is of high importance – but very complex!
- What tools and techniques exist for the *efficient* design and creation of 3D urban environments?

Summary

This course will explain procedural modeling techniques to create urban models for computer games and movies. It will quickly survey previously published techniques and theory, and then delve into their use in graphics practice: tool demonstrations, fitting to real and artistic data, the needs of gaming, and preparation for rendering.

Creating digital content remains a significant challenge in the entertainment industry, especially for urban environments, which are among the largest and most complex. As display capabilities improve and audience expectations grow, procedural modeling techniques are becoming an increasingly important supplement to traditional modeling software. Attendees of this course will learn how techniques for generated urban models can be used in practice, and where these tools still fall short.

In our first session, we will begin with a survey of existing tools, and new developments in the field. These include shape grammars, generative meshes, and procedural textures. Preparing models of this size and complexity for rendering remains a significant challenge. We will survey available techniques and options for rendering, including occlusion culling, imposters, and LOD, as well as how to prepare these urban models for rendering in Maya and Renderman. We continue with an in depth, interactive demonstration of the CityEngine, published previously in two SIGGRAPH papers. We will show how to use the CityEngine for both archeological and architectural applications.

In our second session, we examine design of urban environments in the large. We examine real-life city form, and how planning attempts to shape that form. We also study how urban development can be simulated, and how simulation can be controlled and polished to meet artistic needs. Next, we will study the problem of urban content from the game industry's perspective. Each of Electronic Arts' frequently issued editions of Need For Speed requires several urban environments. We will study the artistic and industry constraints that must shape these environments, how existing procedural tools help EA develop them, and how the tools should improve in the future to meet EA's needs.

Solution: Automatic Content Creation

- Procedural generation of architectural 3D content: tools to “simulate cities and program buildings”
- Encoding the structural, spatial and functional complexity of cities and buildings for CG
- Goal is the efficient creation of detailed 3D models of urban environments at low cost
- Many applications in entertainment, simulation, archaeology, architecture, …

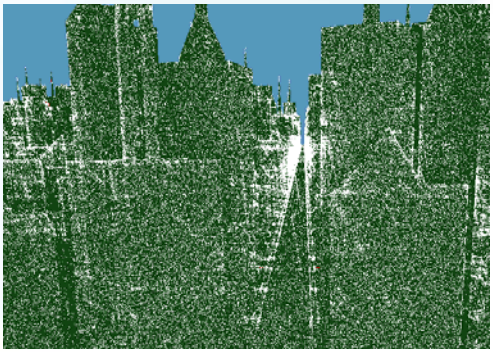
Intended Audience & Prerequisites

- The course is intended for producers and digital set artists who use and researchers who develop computer graphics modeling tools for creating architectural 3D content
- Knowledge of basic computer graphics principles, such as triangle meshes, hierarchical data structures, texture mapping, ray tracing, and procedural modeling techniques such as particle and L-systems is required.
- Level of Difficulty: Intermediate

Overview Part I

Peter Wonka: **State of the Art**

- What is the state of the art in procedural modeling and rendering of urban environments?



Peter Wonka joined the CSE faculty of Arizona State University as Assistant Professor in 2004 after two years as a post-doctorate researcher at the Georgia Institute of Technology. He received his Ph.D. in computer science from the Vienna University of Technology in 2001 and a masters degree in urban planning in 2002. His research interests include various topics in computer graphics, especially real-time rendering, procedural urban modeling and the application of computer graphics and visualization to various urban planning problems. Peter Wonka is a member of the PRISM lab.

Overview Part II

Pascal Müller: **Applied Procedural Modeling**

- How does procedural modeling work in practice?

URBAN DESIGN AND PROCEDURAL MODELING
P. WONKA, P. MUELLER, B. WATSON, A. FULLER

SIGGRAPH2007

Pascal Mueller is PhD candidate and research assistant at the Computer Vision Lab of the ETH Zurich, Switzerland. His main interests lie in the field of computer graphics: procedural modeling, generative design, animation, visual effects production pipelines and computer-aided media art. He developed the architectural modeling tool CityEngine and is co-developer of the multimedia engine Soundium. He has published various scientific papers including SIGGRAPH, and his body of artistic work includes videos, short movies, over fifty live visuals performances, and several interactive installations exhibited in museums like the Ars Electronica Center. Pascal Mueller received a master degree in computer science from ETH Zurich in 2001. For two years, he worked as a 3D artist and technical director for the Swiss production company Central Pictures.

Overview Part III

Andy Fuller: **Urban Modeling in Games**

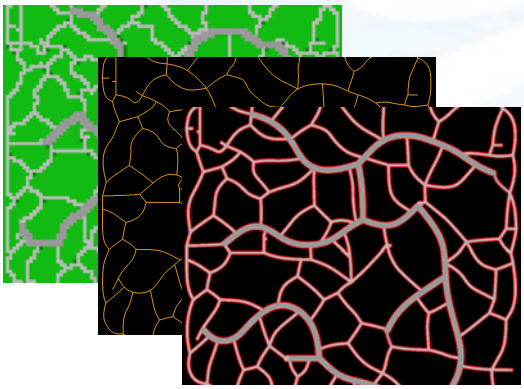
- What are the industrial practices to design and model a virtual city for a game?

Andy Fuller is currently working as an Associate Computer Graphic Supervisor at Electronic Arts. He worked full time in the computer game development industry since 1994 (including porting the original Need For Speed onto the Sega Saturn). In a former life he worked as a free lance artist and illustrator, and studied art at the Seattle Art Institute.

Overview Part IV

Ben Watson: **Real and Virtual Urban Design**

- How can urban simulation techniques be used in computer graphics?



Benjamin Watson is Associate Professor of Computer Science at North Carolina State University. His research focuses on design graphics: computer imagery as and about designed objects, and spans adaptive display and 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 co-chaired the Graphics Interface 2001, IEEE VR 2004 and ACM I3D 2006 conferences, and is co-program chair of I3D 2007. Watson is an ACM and senior IEEE member. He earned his doctorate at Georgia Tech's GVU Center.

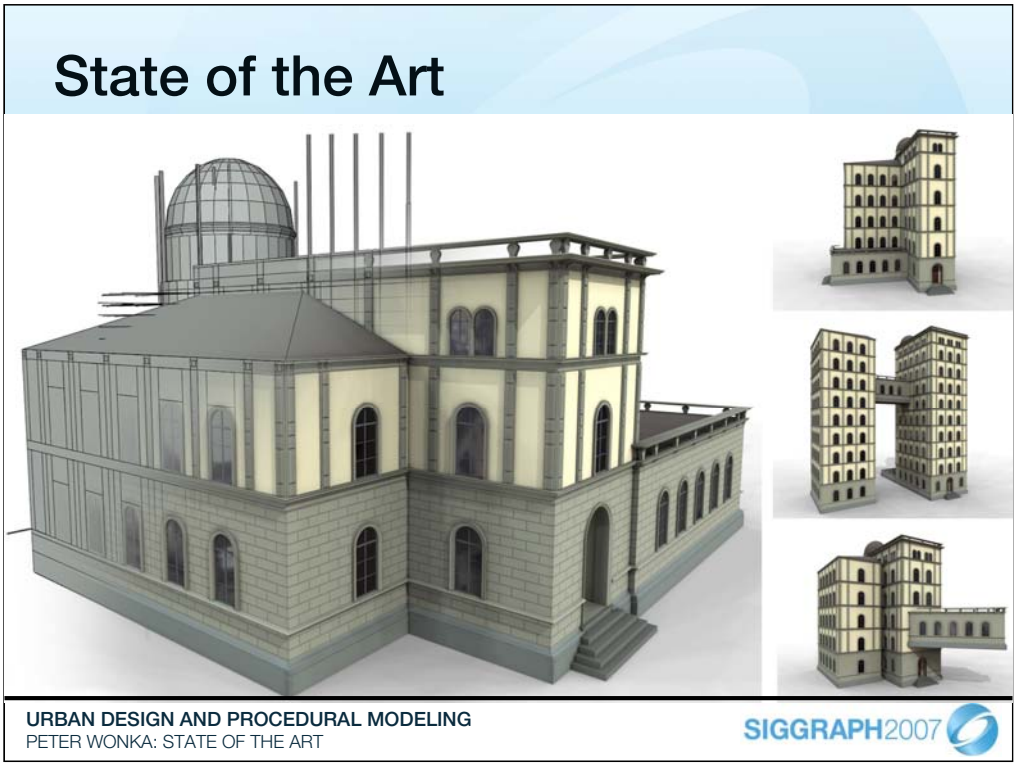
Schedule

8:30 - 8:45	**Intro** (B. Watson)
8:45 - 9:30	**State of the Art** (P. Wonka)
9:30 - 10:15	**Applied Procedural Modeling** (P. Müller)
10:15 - 10:30	**Break**
10:30 - 11:15	**Urban Modeling in Games** (A. Fuller)
11:15 - 12:00	**Real and Virtual Urban Design** (B. Watson)
12:00 - 12:15	**Open Discussion** (All)

State of the Art in Modeling and Rendering

Peter Wonka

Arizona State University



2) Procedural modeling of buildings – Peter Wonka

This section will explain selected parts of the state of the art of procedural modeling.

Outline – 1/3

- Introduction to Architecture
- Stiny‘s Shape Grammar
- Modeling Street Networks
- Modeling Facades and Building Shells
- Procedural Techniques for Reconstruction

At the beginning I will give some short ideas about architecture and how architecture uses grammars.

When modeling a city from scratch, it might be a good idea to start with the street network and the creation of parcels.

A building consists of multiple parts. We will consider three main parts in this course: Modeling of building shells, modeling of roofs, and modeling of room layouts. The most important part for outdoor views is the building shell. We will introduce a shape grammar for buildings to model these shells.

I will also briefly mention two approaches to create procedural models from images.

Outline – 2/3

- Cellular Textures
- Generative Mesh Modeling
- Roofs
- Room Layouts

To augment existing models with complex brick patterns cellular texturing can be used.

Generative Mesh Modeling will allow to specify geometric details. Typically, this results in a much lower level language and specification.

The modeling of roofs is fairly tricky and needs more complex computations such as the straight skeleton.

such as smooth curved surfaces that are difficult to model with grammars.

The inside of buildings consists of the room layouts and the placement of furniture.

This can be done with grammars or with optimization routines.

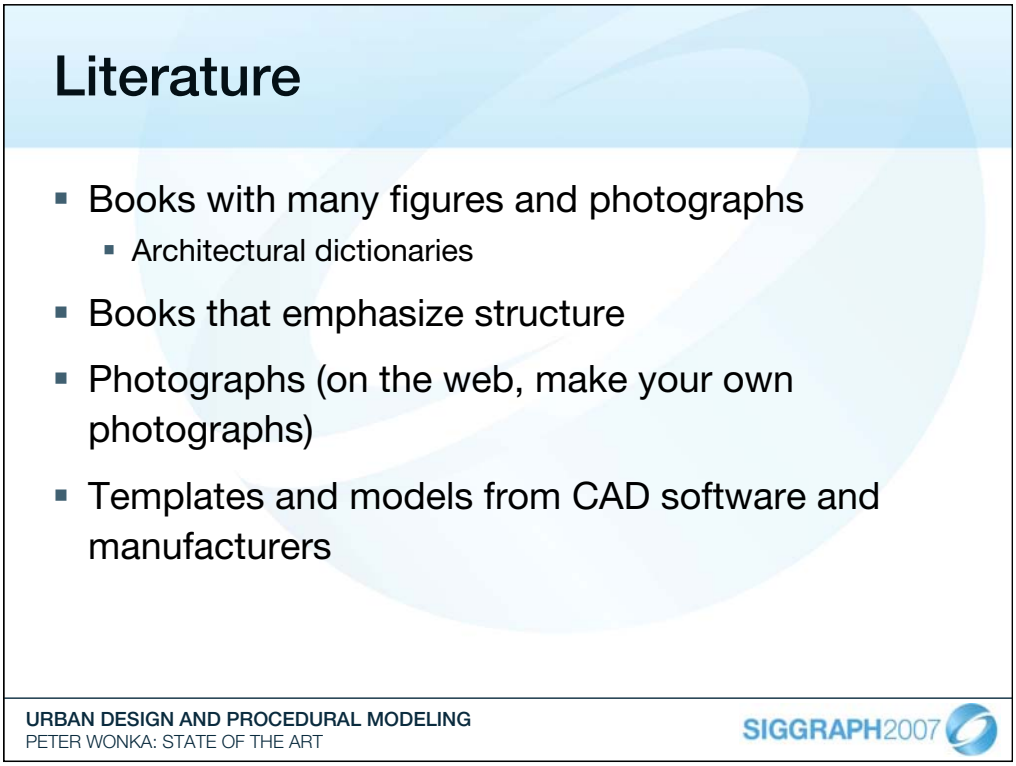
Outline – 3/3

- Lighting
- Textureing
- Rendering Techniques
  - Monte-Carlo Ray Tracing
  - Renderman
  - Real-time Rendering with GPUs
  - Real-time Ray Tracing

Personally, I found it important to combine modeling with a strong rendering solution. If the rendering is bad, the model will never look good.

The parts about the rendering aim at discussing the interface between modeling and rendering rather then an isolated rendering discussion.

Introduction to Architecture



I personally would recommend reading books that have many labeled figures and images.

Bildwoerterbuch der Architektur. Koepf and Binding. [Koepf]
This is my favorite book. Unfortunately this book is in German. A similar book is

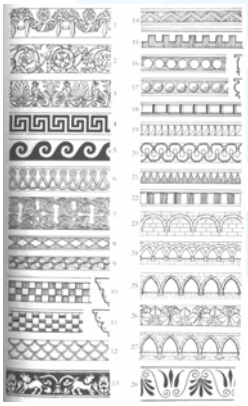
A Visual Dictionary of Architecture. Francis D. K. Ching. [Ching]
This books is not nearly as extensive though.

Baustilkunde. Wilfried Koch (recommended by Sven Havemann)

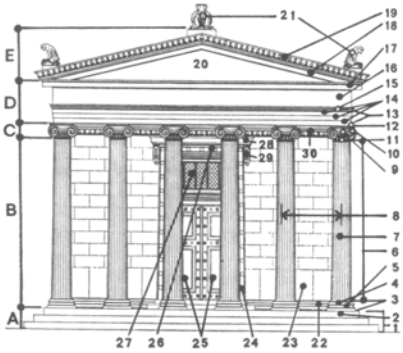
Example

- Bildwoerterbuch der Architektur. Koepf and Binding:

frieze

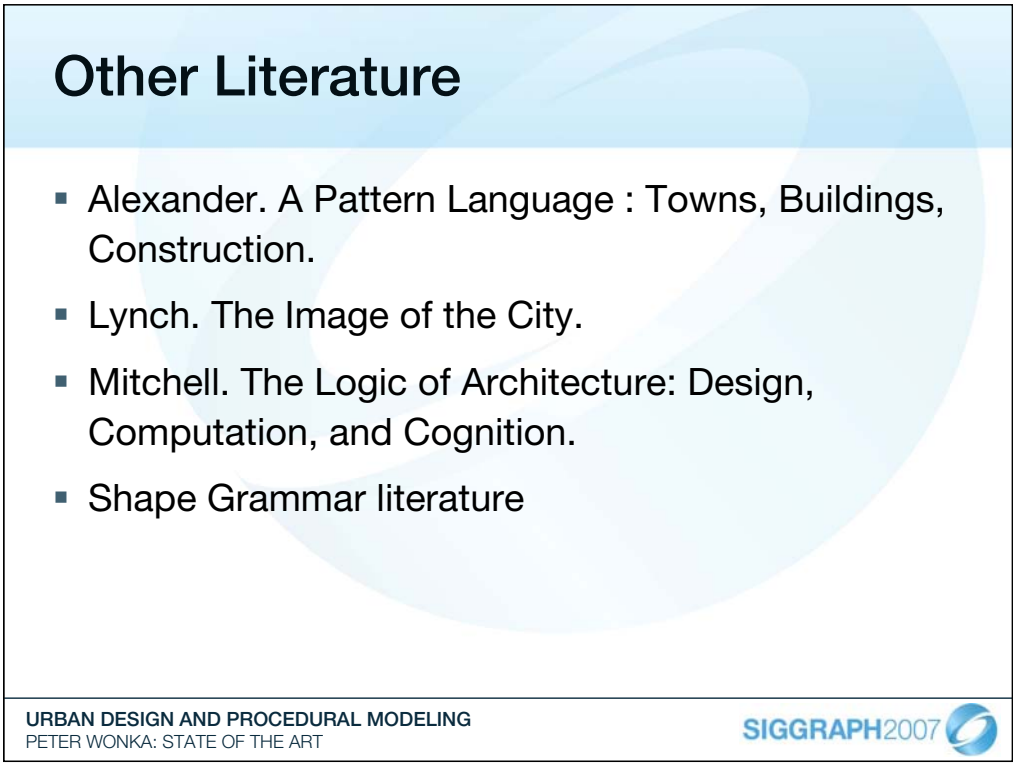


Ionic Temple Order



These type of entries are helpful to learn about the structure and variations in architecture. For example, the entry for frieze in Koepf's book gives a short description of the term and then includes a figure an several variations. This is a good starting point for a texture library for example.

The image on the right shows the ionic temple order. The figure together with the labels are an excellent tool to infer structure.



These are some of the more popular references in computer graphics to architectural and urban planning literature.

While these are great books, it is essential that the basics are studied first.

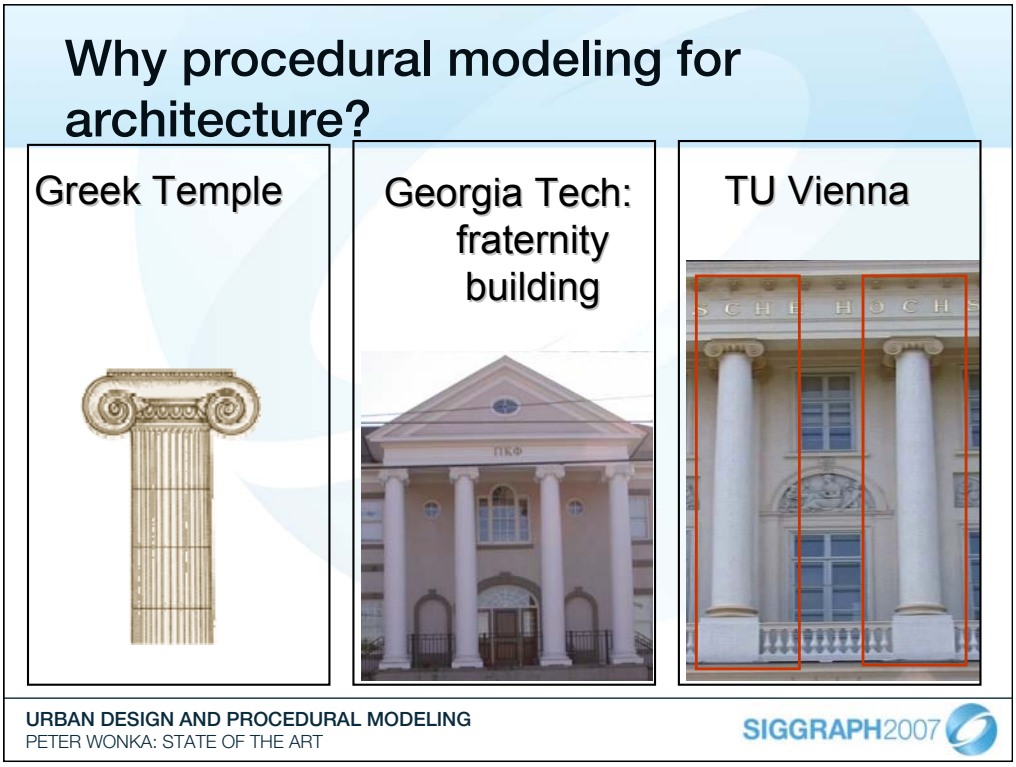
[Alexander77] ALEXANDER, C., ISHIKAWA, S., AND SILVERSTEIN, M. 1977. A Pattern Language: Towns, Buildings, Construction. Oxford University Press, New York.

This book is a very popular reference, but it is very abstract. I think it is difficult to get useful design rules from this book that are helpful for procedural modeling.

I therefore suggest to start with other books first.

[Lynch] Kevin Lynch. The Image of the City.

This book is very interesting to read. It can be used as an inspiration for city design, but it is also not specific enough to directly translate into rules for procedural modeling.



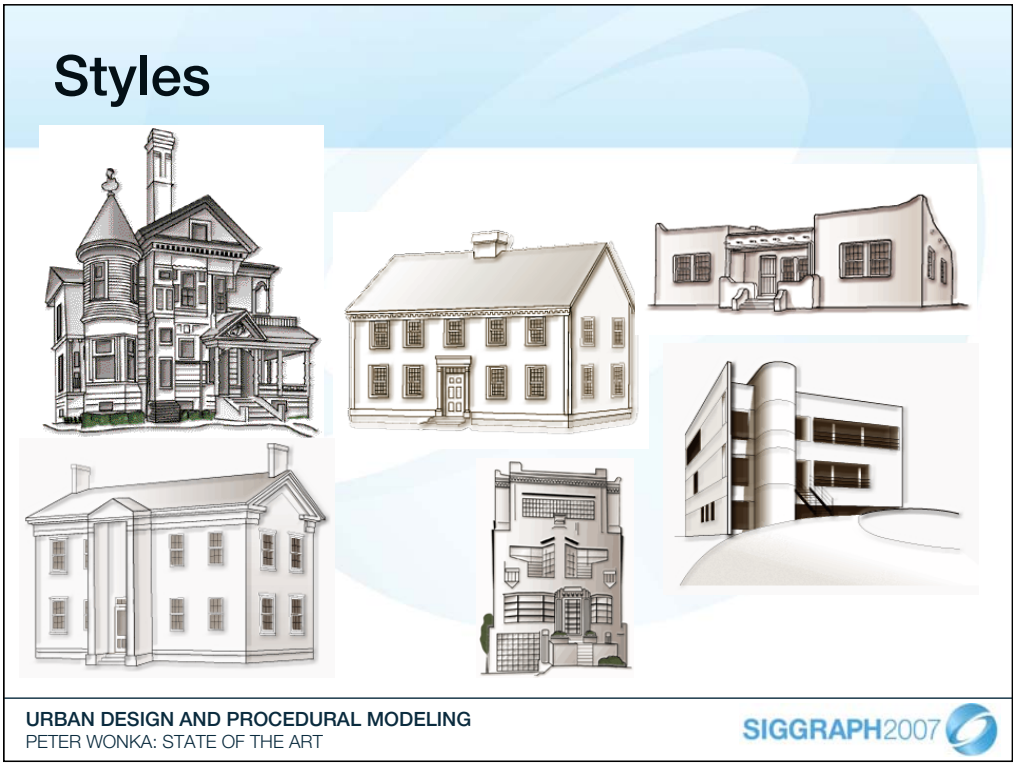
The left image shows a ionic column of a Greek temple.

The middle image is a photograph of a fraternity building on the Georgia Tech campus. It is probably one of the nicest fraternity buildings on campus.

The right image is the main building of the Vienna University of Technology.

Similar architectural elements occur in different times, in different styles, and in different countries. The similarity between buildings of the same city at the same time is even higher.

This makes procedural modeling an efficient tool, because rules can be reused and many variations of a single building can be created.

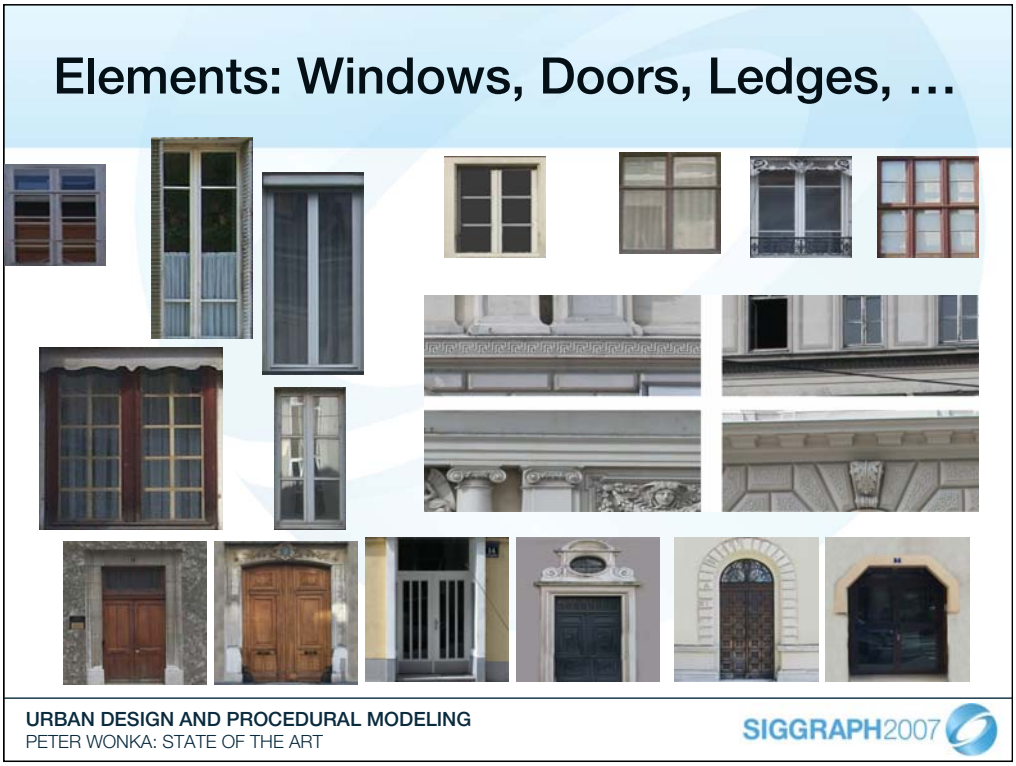


These images are from the webpage: http://www.realtor.org/rmomag.nsf/pages/arch21

(I actually like this page because you can get a 5 minute overview over a large range of styles,

e.g. Art Deco, California Bungalow, Cape Cod, Colonial, Contemporary, Craftsman, Creole,

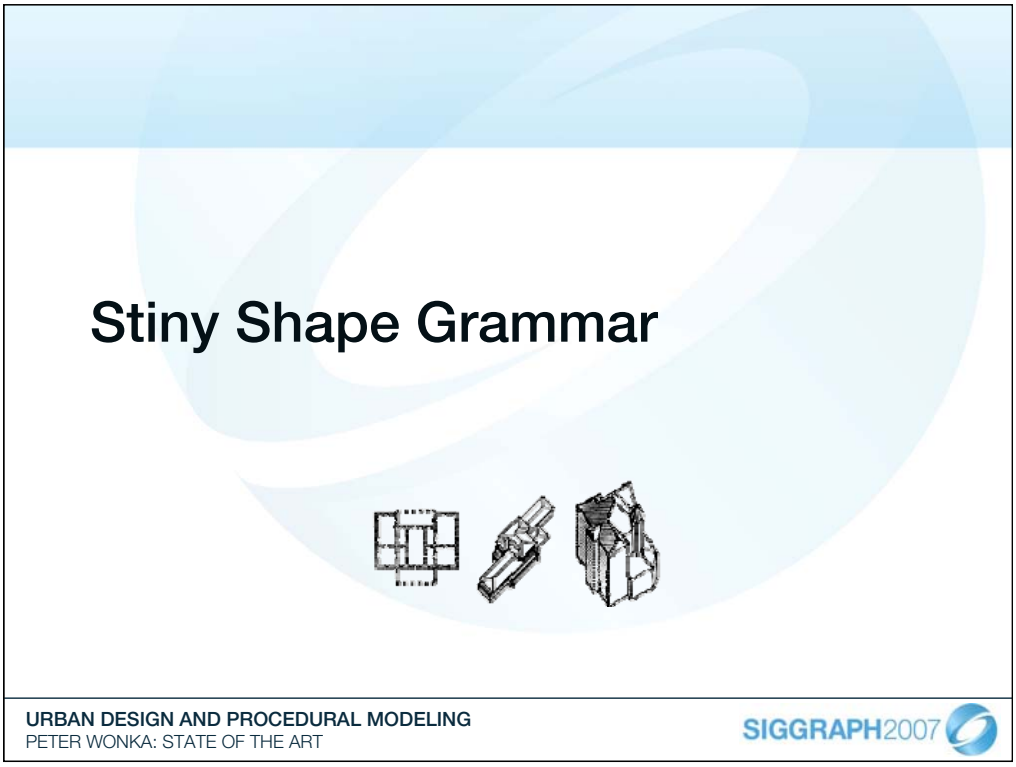
Dutch Colonial, Federal, French Provincial, Georgian, …



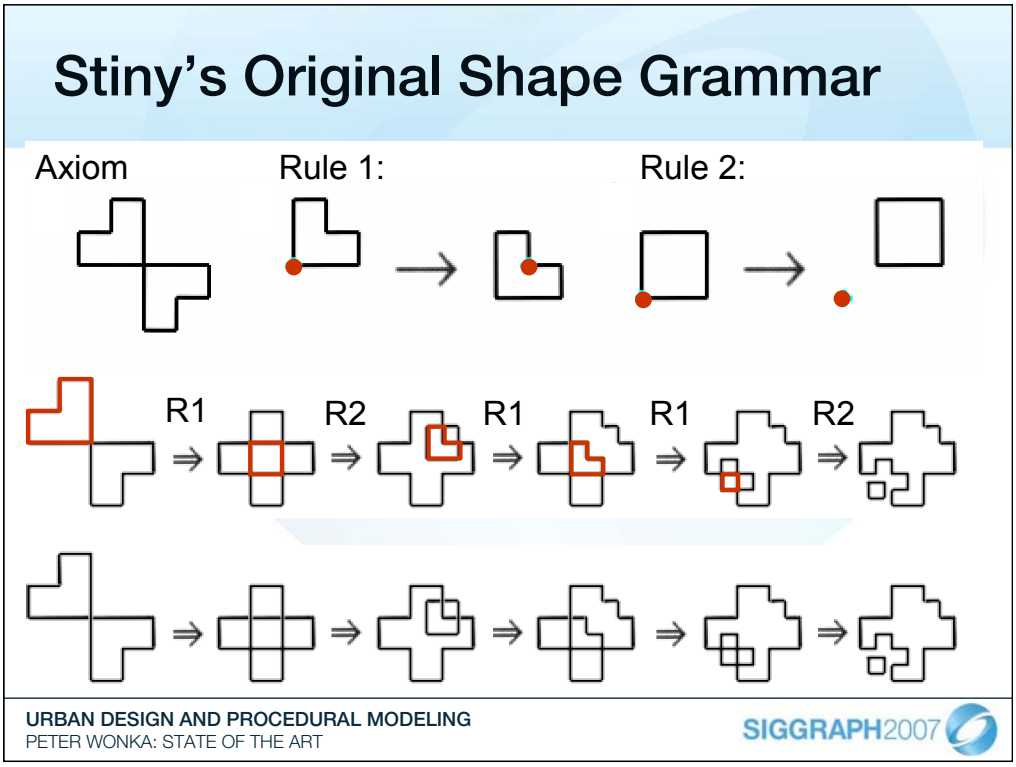
Unfortunately, there is a rather extensive list of architectural elements. I first started to model European architecture from Austria and France using

the following elements: windows, doors, ledges, quoins, window gratings, awnings, window ornaments, pilasters, and stairs.

Another interesting website might be http://freenet.buffalo.edu/bah/a/DCTNRY/vocab.html. Unfortunately not very extensive, but free.



[Stiny80] Stiny. Introduction to Shape and Shape Grammars. Environment and Planning B. volume . pages 343 – 361. 1980



This example introduces Stiny's origninal shape grammar. A more formal description will follow.

The shape grammar operates on arrangements of lines. The starting point is the shape in the top left corner.

Two rules are given: rule one and rule two.

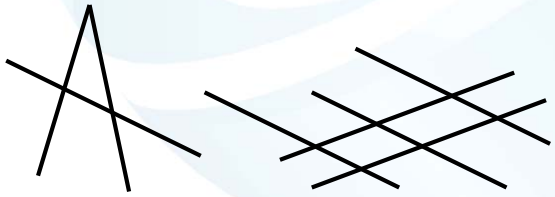
In the middle row the slide shows the derivation of one example. Which rule was chosen for the derivation is written over the arrow:

R1 means rule one and R2 means rule two.

The red lines highlight the subshape that is selected for replacement by the grammar.

Shape

- **Definition:** A **shape** is a limited arrangement of straight lines (defined in a Cartesian coordinate system with real axis and an associated Euclidian metric)



URBAN DESIGN AND PROCEDURAL MODELING
PETER WONKA: STATE OF THE ART
SIGGRAPH2007

I would recommend reading the paper:

Stiny. Introduction to Shape and Shape Grammars. Environment and Planning B. volume . pages 343 – 361. 1980

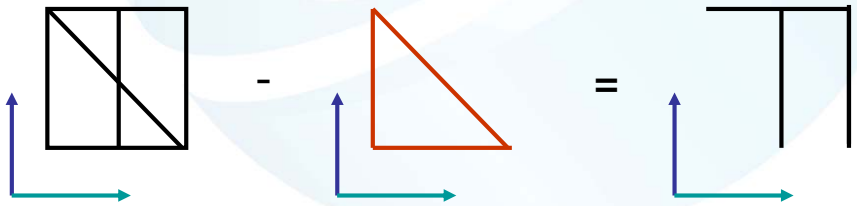
This paper includes a “formal” definition of the shape grammar formalism including:

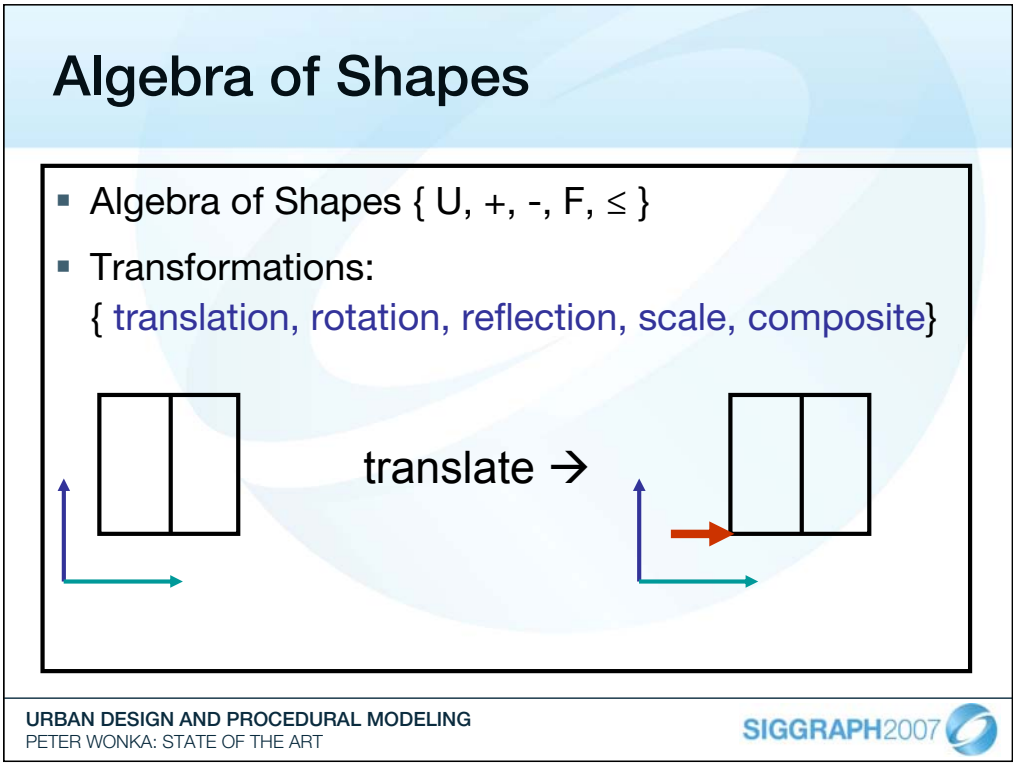
definition of shape; the shape grammar formalism including parametric shape grammars

For the outline of the grammar I will use a mixture between the description in this paper and the Version given in Instant Architecture. Siggraph 2003.

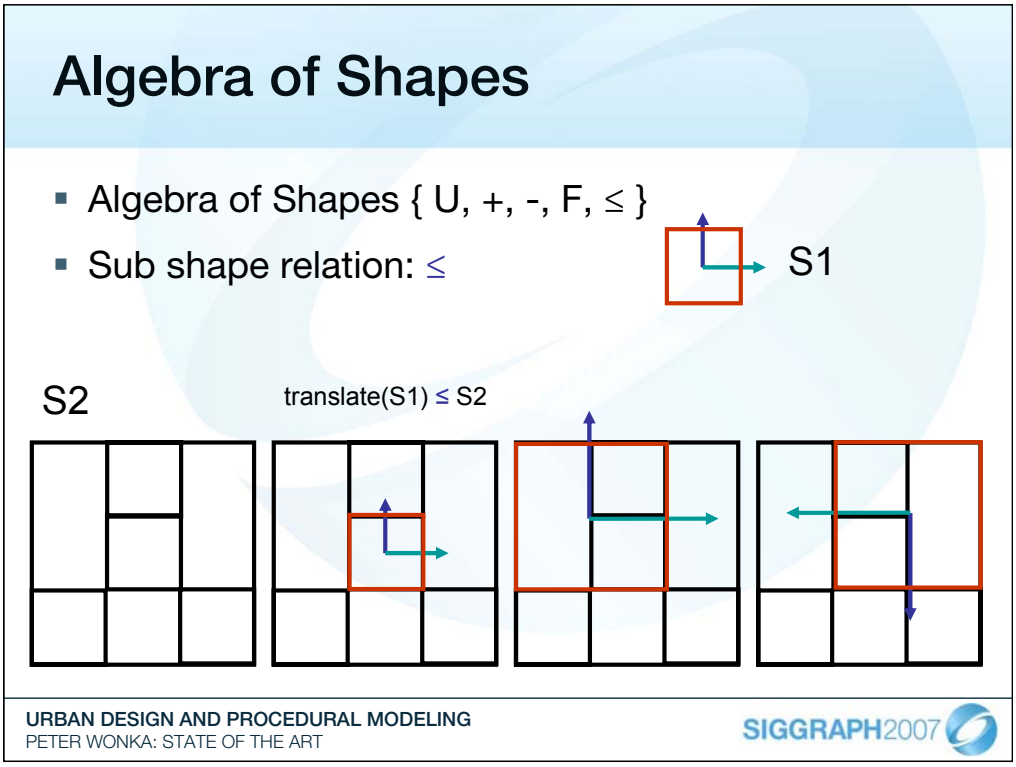
Algebra of Shapes

- Algebra of Shapes { U, +, -, F, ≤ }
- Operations: + and -





Stiny mentions the transformations: translation, rotation, reflection, scale, and any composite transformation between the two. The list can be extended or shortened for given applications.



This figure shows how a shape S1 can match different subshapes of S2.

The first example shows a matching with a translation, the second example a matching with translation and scaling,

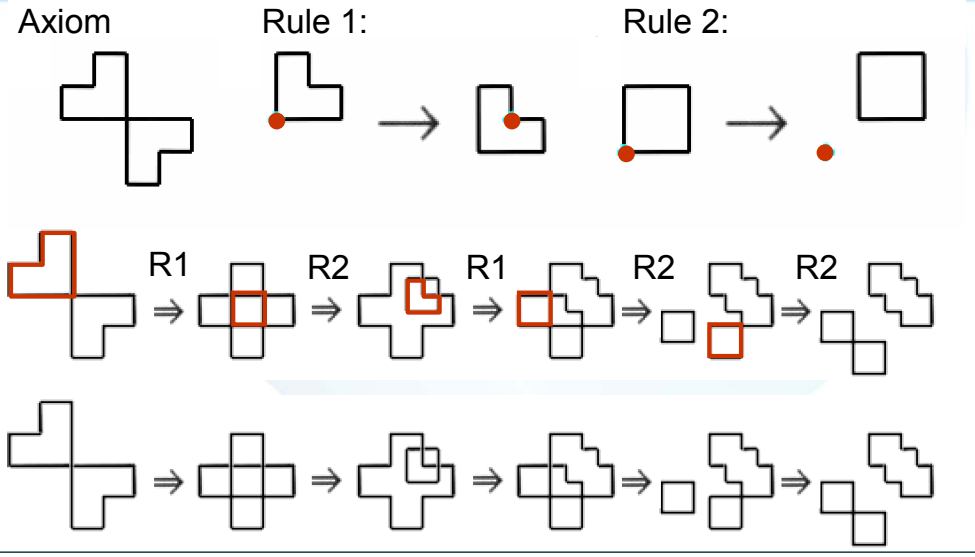
the third example uses rotation, scaling and translation. The two arrows within S1 are a local coordinate system to visualize

the transformations.

Shape Grammar

- Algebra of Shapes { U, +, -, F, ≤ }
- Rules: A → B (A and B are shapes)

Stiny's Original Shape Grammar



Architecture vs. Computer Graphics

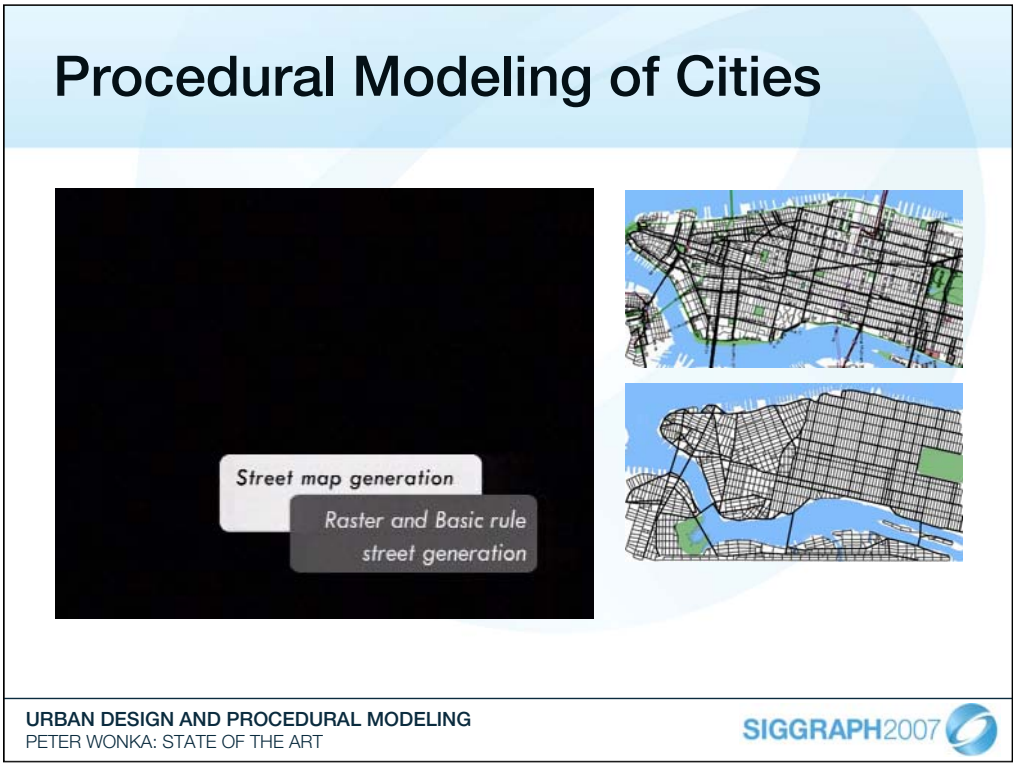
- Architecture:
  - emergence of shapes
  - create unexpected designs
  - Goal: outstanding designs
- Computer graphics:
  - automatic modeling
  - many mediocre designs
  - assist in creating great designs

Architecture and Computer Graphics have fairly different goals:

Architects typically like to create great designs that become famous, that include something creative, ...

In Computer Graphics (semi-)automatic modeling tools are important.

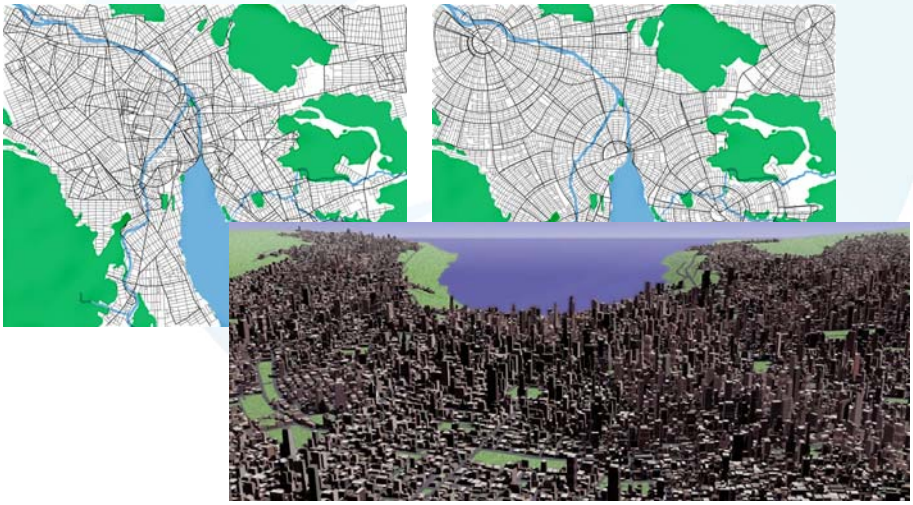
Modeling Street Networks

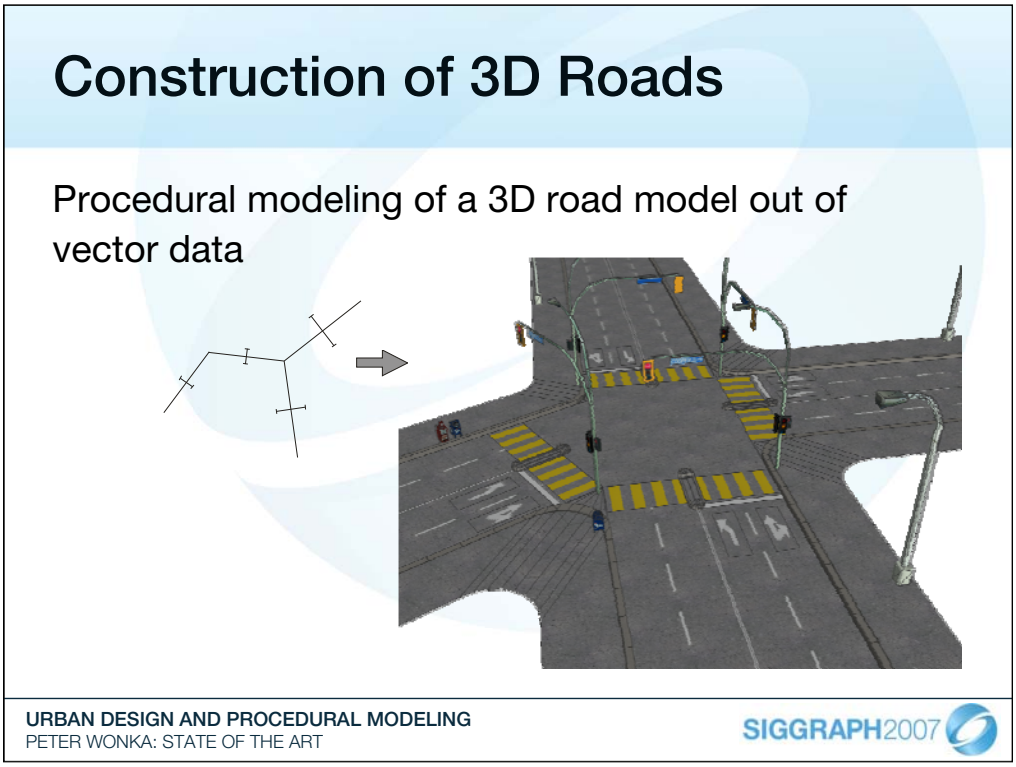


[Parish01] Parish and Müller. Procedural Moderling of Buildings. Siggraph 2001.

The idea of street modeling is to grow a graph.

Example: Virtual Cities



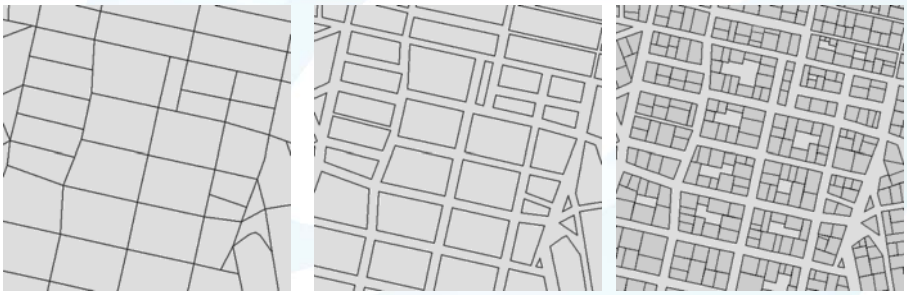


After a street graph is generated, it is still important to create the actual geometry. This can also be done using some procedural techniques.

The main idea is to use attributes of the cross section of a road. Such a system is described in:

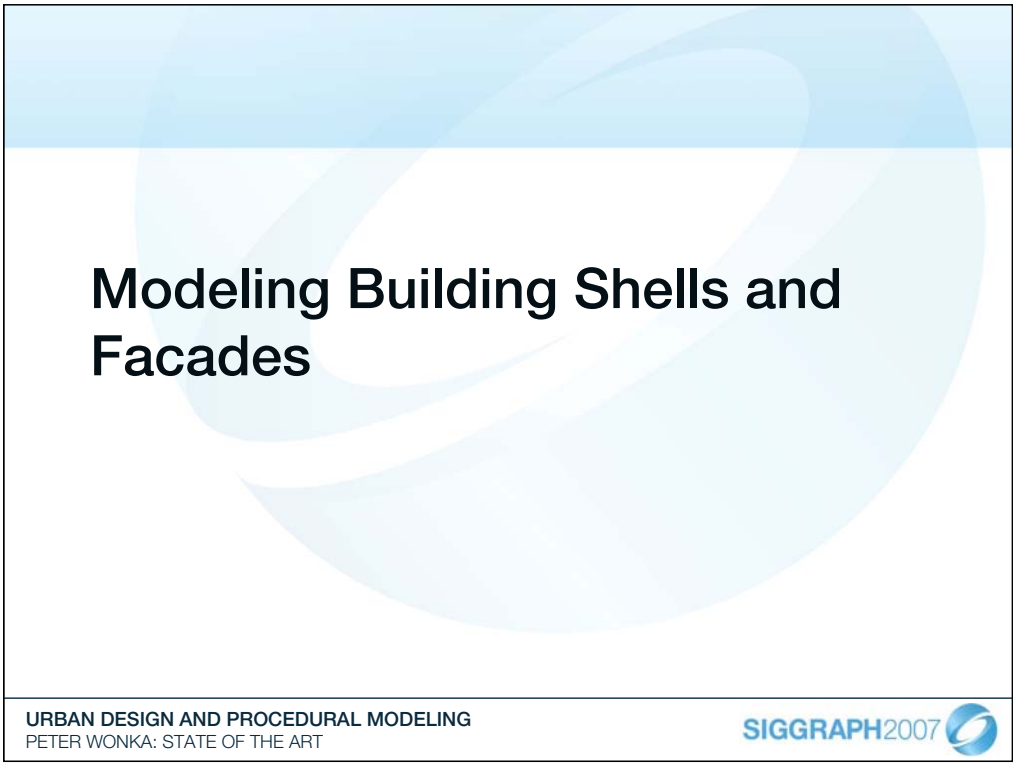
THOMAS, G., AND DONIKIAN, S. 2000. Modelling virtual cities dedicated to behavioural animation. *Computer Graphics Forum (Proc. Eurographics '00) 19*, 3 (Aug.), 71–80.

Division into Lots



Lot area depends on:

- Land use map and population density
- Access to street and type of street
- Geometric constraints



The main references for this section are:

[Wonka03] Wonka, Wimmer, Sillion, and Ribarsky. Instant Architecture. Siggraph 2003.

[Müller06] Müller, Wonka, Haegler, Ulmer, Van Gool. Procedural Modeling of Buildings. Siggraph 2006.

Modeling Architecture with Grammars

Procedural modeling using grammars (Version 1)

- String
- *Rules* (String replacement)
- *Derivation* (until the resulting string contains only terminal symbols)
- *Geometrical interpretation* of the terminal symbols

Procedural modeling using grammars (Version 2)

- Set of Shapes
- *Rules* (take one shape and replace it with other shapes)
- *Derivation* (until the resulting set contains only terminal shapes)
- *Geometrical interpretation* of the terminal shapes

Vocabulary: formal elements which make up its repertory

Syntax: system of relationships

iteratively generate a design by creating more and more detail

CGA Shape Grammar (SG 2006)

A shape grammar for the procedural modeling of CG architecture.

- Human-readable rules (notation like L-systems)
- Consistent design of mass models and facades
- Not restricted to axis aligned shapes
- Context sensitive rules to specify shape interactions

Not restricted to axis: e.g. Roofs or arbitrary footprints!

Defining the most important shape rules, the concise notation, and giving modeling examples

Based on Shape Grammars (Stiny & Gips, 1971)

- Analysis / creation of architectural designs
- Derivation usually done manually

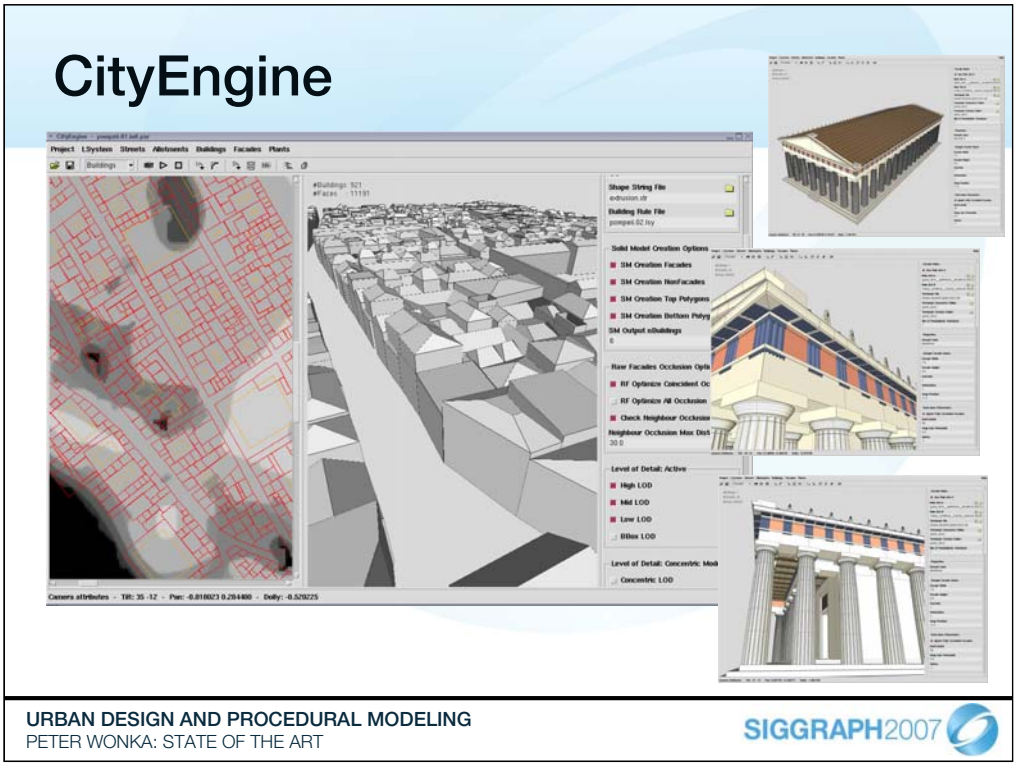
CGA Shape

Production process:

- Iteratively evolve a design by creating more and more details
- Sequential application (like Chomsky grammars)

URBAN DESIGN AND PROCEDURAL MODELING
PETER WONKA: STATE OF THE ART
SIGGRAPH2007

While parallel grammars like L-systems are suited to capture *growth* over time, a sequential application of rules allows for the characterization of *structure* i.e. the spatial distribution of features and components [Prusinkiewicz et al. 2001]. Therefore, *CGA Shape* is a sequential grammar (similar to Chomsky grammars).



Most models have been created using a modeling tool called CityEngine. It is proprietary software developed by Pascal MÜller.

City Engine includes a shape grammar for Architecture: CGA shape.

This shape grammar is an extension of the grammar presented

in the paper Instant Architecture. CGA shape is described in the paper: Procedural Modeling of Buildings.

Most of the following slides refer to this modeling system.

Modeling with Grammars: Two Sides

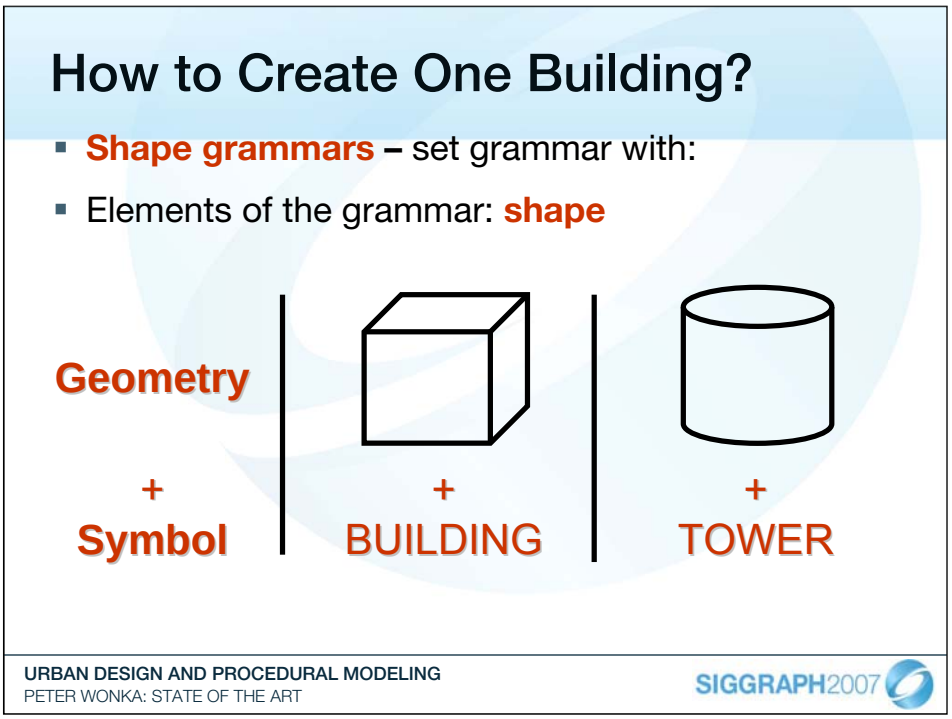
- Framework: e.g. syntax and semantic of a grammar
  - rule ::= id : predecessor : cond → successor : probability
  - cond ::= …
- Actual design knowledge: e.g. rules

For modeling with grammars it is important to recognize two sides of the problem: First, you need a framework that is powerful enough to encode architectural design knowledge. Second, the rules that represent the architectural design knowledge themselves. These two aspects are closely related and it is necessary to have both to validate the framework.

The first part is actually very tricky, because the best framework is probably not the most powerful one. I think an extremely powerful framework would be C++.

All architectural concepts could be encoded in C++ classes somehow and then the code is compiled and executed.

At the end you get a model.



The basic idea of the grammar is to iteratively replace one shape with a set of other shapes. The derivation of the shape grammar will therefore give a shape tree. Another implementation would just work on a set of shapes without storing the tree explicitly.

The main difference to L-Systems is that we do not derive strings but shapes. While a rule is similar to L-Systems, so that a left hand side of the rule is replaced with the right hand side we do not replace strings, but interpret the right hand side as command. The right hand side encodes how the shape is to be replaced.

Rule Fromat

- *id* : *pred* : *cond* → *successor*
- Example:

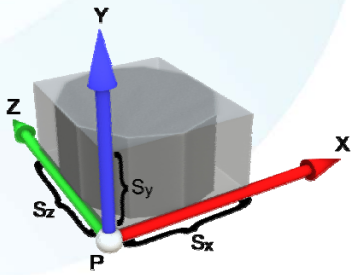
  1: *fac*(*h*) : *h* > 9 → *floor*(*h*/3) *floor*(*h*/3) *floor*(*h*/3)
- **id** – an integer identifying the rule
- **pred** – text string = symbol of the shape to be replaced
- **cond**: condition on the parameters of the shape
- **successor**: shapes to replace the predecessor

Shape Rules

- Notation:

  id: predecessor : condition ⇢ successor : prob

- A shape consists of:
  - Symbol (string)
  - Geometry (polygonal mesh)
  - Oriented bounding box called *scope* (numeric attributes)



Shape with symbol *predecessor* is replaced by shape with symbol *successor* if *condition* is satisfied and with probability *prob (in this just a renaming)*

- Parametric, conditional, stochastic...

Axiom is usually a bounding box, a parcel or a footprint.

(If footprint, you can extrude it via S(..))

numeric attributes can be accessed via grammar

Rule Types

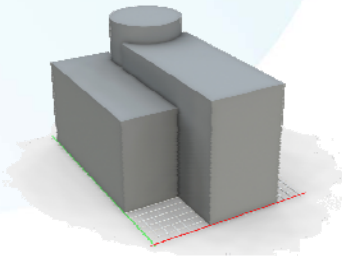
- Transformation Rules
- Split Rule
- Repeat Rule
- Scope Rules
- Component Split
- Occlusion
- Snaplines
- …

Basic Shape Operations

- Insertion: *I(objId)*
- Transformations: $T(t_x,t_y,t_z)$, $S(s_x,s_y,s_z)$, $Rx(\alpha)$..
- Branching: *[...]*

Simple example:

1: A → [T(0,0,6) S(8,10,18) I(cube)]
T(6,0,0) S(7,13,18) I(cube)
T(0,0,16) S(8,15,8) I(cylinder)



URBAN DESIGN AND PROCEDURAL MODELING
PETER WONKA: STATE OF THE ART
SIGGRAPH2007

Insertion: insertion of terminal shapes (We used Maya to generate geometry, such as roof bricks, the capitals, and window grills.)

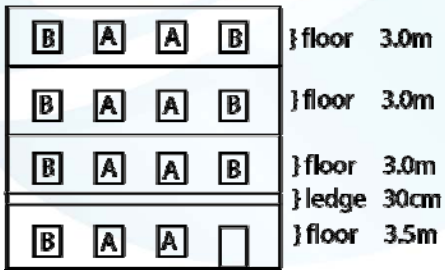
Branching: push pop shape state during rule application

Unlike l-systems, rules are directly interpreted and resulting shapes are stored in shape derivation tree

The Subdivision Split

Example:

1: facade →
Subdiv(Y,3.5,0.3,3,3,3){ floor | ledge | floor | floor | floor }

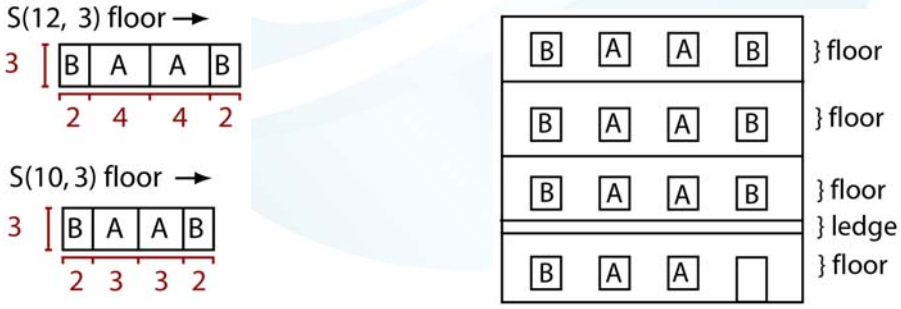


The first parameter of the *Subdiv* describes the split axis (*x*, *y* or *z*) and the remaining parameters describe the split sizes. Between the delimiter *{* and *}* a list of components is given, separated by |. Please note that we choose this notation to improve the Readability

The obvious problem is that this rule would only be valid for an object with y-size 15.5.
Therefore we need a mechanism to scale rules: relative coordinates!

Scaling Split Rules

- *floor* → *Subdiv("X",2,1r,1r,2){ B | A | A | B }*
- *façade* → *Subdiv("Y",3.5,0.3,1r,1r,1r)*
  (fl1 | ledge | fl2 | fl2 | fl2 }



To scale the split rule we use the modifier "r". "r" stands for relative and contrasts "a" for absolute.

We do not write "a" though, it is implicitly assumed. The idea is that after all the absolute length

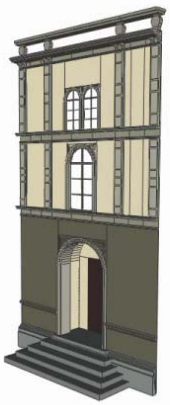
are subtracted from the splitting axis size, the remaining length is split according proportional to the r values

The Repeat Split

Example:

1: floor ⇢ Repeat(X,2){ window }

- Create as many *window* elements of approximate size 2 as there is space

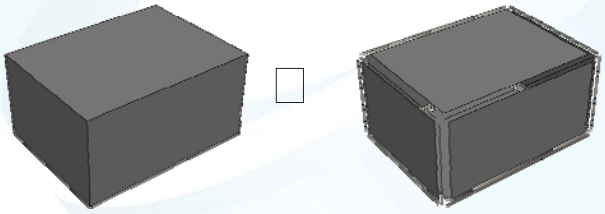


- Size-independent („Repeat“) and

The Component Split

Format:

id: A ⇢ Comp(type,params){ B | C | ... | Z }



Example:

1: solid ⇢ Comp("sidefaces"){ facade }

URBAN DESIGN AND PROCEDURAL MODELING
PETER WONKA: STATE OF THE ART
SIGGRAPH2007

The basic idea of the component split is to break down a shape based on its geometry.

Means you can create new shapes out of the components of the geometrical mesh and assign symbols to them.

Very powerful split and most important: it reparameterizes the scope (coordinate system..)

Where *type* identifies the type of the component split with associated parameters *param* (if any)

This is a simple example, but you can also access single edges, or single components or assign different symbols to each face etc...

Modeling with Split Grammars Example

- Take Photograph
- Create abstraction

This was my starting point for modeling.

Take a photograph and reconstruct the given facade.

The grammar has to be powerful enough, so that the grammar works for a façade geometry of (almost)

any given size.

The idea of the grammar is to have a different **firstfloor** and then as many **floors** as there is space.

This makes the grammar independent of the height of the building.

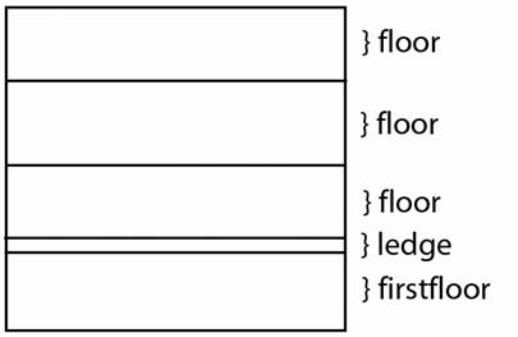
The grammar only makes sense for limited dimensions.

It is not reasonable to use the grammar for buildings with 100 stories or facades

of width 200m. We ignore this aspect at the beginning.

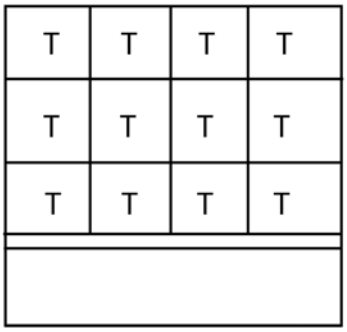
Modeling with Split Grammars Example

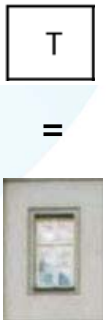
- *Facade* → *Subdiv*(“*Y*”,3.5,*0.3*,1*r*){ *firstfloor* | *ledge* | *floors*}
- *Floors* → *Repeat*(“*Y*”,*3*){*floor*}



Modeling with Split Grammars Example

- floor → Repeat("X",tile_width){ Tile }



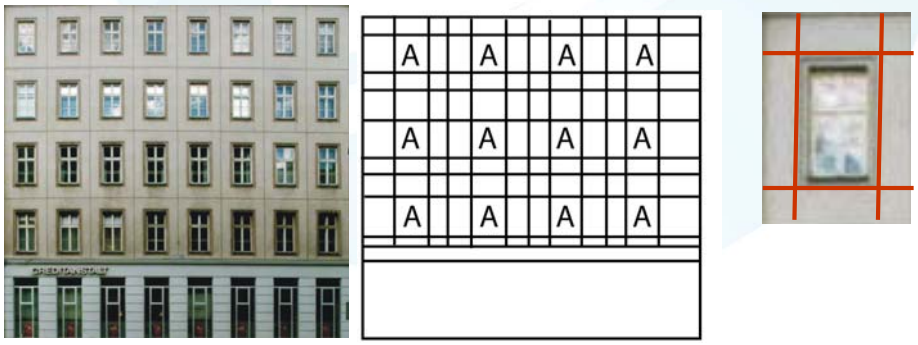


URBAN DESIGN AND PROCEDURAL MODELING
PETER WONKA: STATE OF THE ART
SIGGRAPH2007

I used the concept of "Tile" a lot in my first models. A tile is a window and the surrounding wall or a door and the surrounding wall.

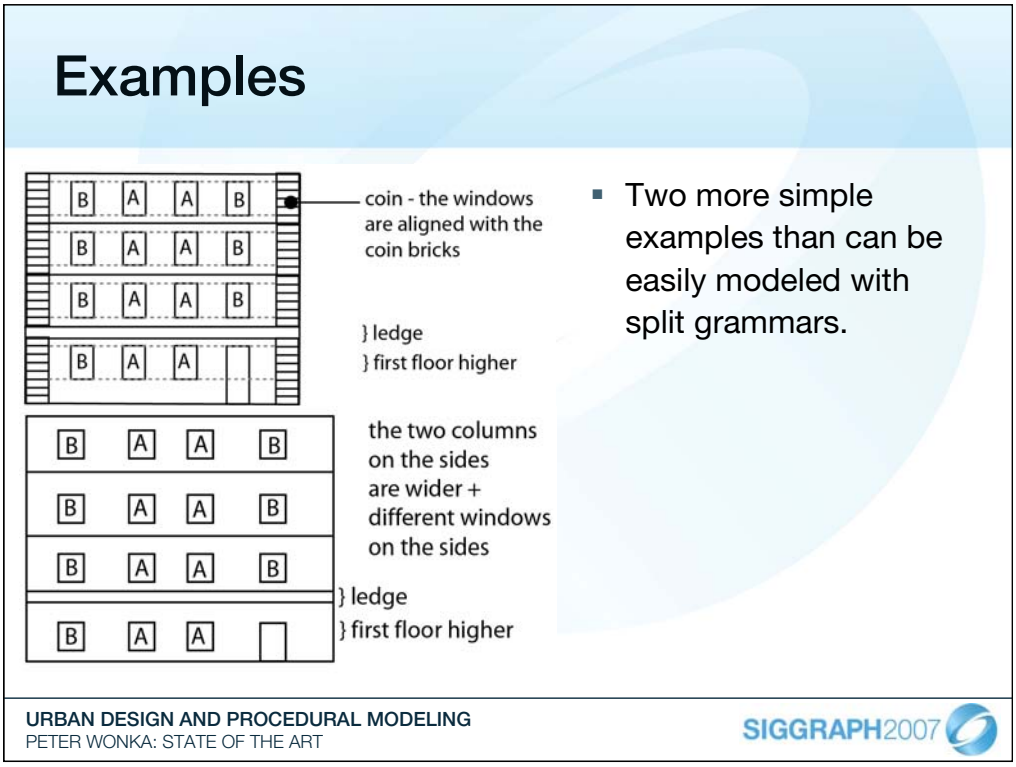
Modeling with Split Grammars Example

- Tile → Subdiv("XY", …){ Wall | Wall | … | A | Wall | … }



I also used a split to separate the wall from the doors or windows. That works well for many cases, but the wall splits into too many unnecessary elements.

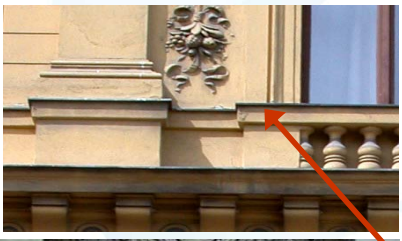
It is possible to split differently with less subdivisions, but I found this one sufficient for most cases.



Note: It is not necessary to split into floors first. As noted by various authors the façade structure is often similar to a matrix.

Therefore, it is possible to split into floors (rows) as well as columns.

Limitations of Split Grammars



- Complex geometry
- Features across multiple architectural elements
- Complex alignment

Shape Interaction: Motivation

- *Problem:*
  The volumes are not aware of each other ⇢ unwanted intersections

URBAN DESIGN AND PROCEDURAL MODELING
PETER WONKA: STATE OF THE ART
SIGGRAPH2007

So now, we have a grammar to create appealing mass models, can break them down into facades via component split and subdivide the facade to create detailed models. But then we have a problem... the volumes are not aware of each other: Stochastically assembled solids lead to windows unwanted intersections which will cut windows (or other elements) in unnatural ways…

Shape Interaction: Motivation

- *Solution:*
  Test the spatial overlap and align elements to important lines

So now, we have a grammar to create appealing mass models, can break them down into facades via component split and subdivide the facade to create detailed models. But then we have a problem... the volumes are not aware of each other: Stochastically assembled solids lead to windows unwanted intersections which will cut windows (or other elements) in unnatural ways…

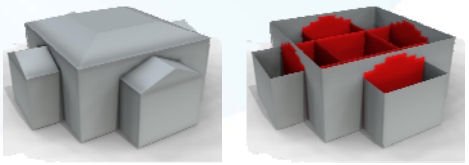
Testing the spatial overlap and align elements to nearby important lines

Shape Interaction: Occlusion

Example:

1: tile : Shape.occlusion == "none" ⇢ door

- Return values are either *"none"*, *"partial"* or *"full"*
- Shape-geometries or scope-boxes can be queried



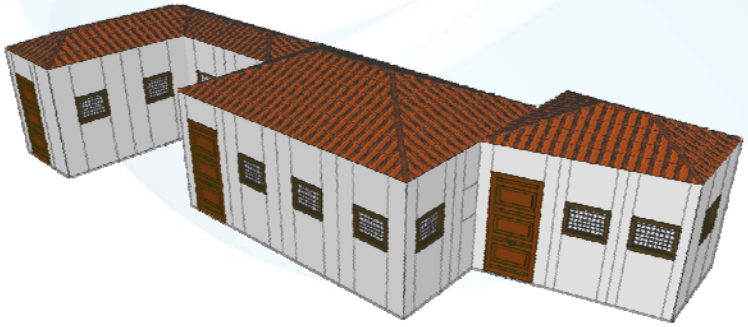
An occlusion query tests for intersections between shapes.

The label defines what shapes should be queried: either the name of a shape or a flag like *"all"*

Shape-geometry or scope-box can be queried

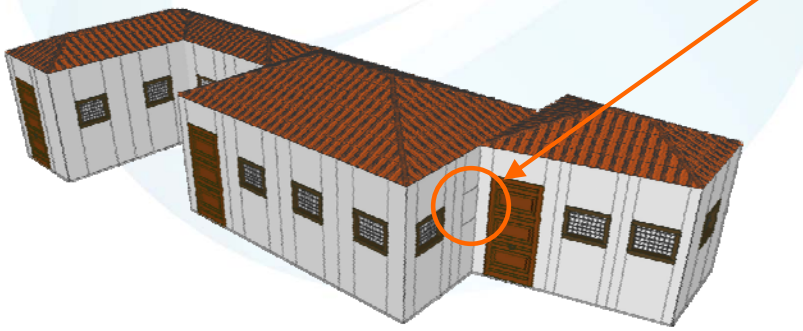
Occlusion Example

6: *tile : Shape.occlusion == "none"* ⇢ *window*
7: *tile : Shape.occlusion == "part"* ⇢ *wall*
8: *tile : Shape.occlusion == "full"* ⇢ ε



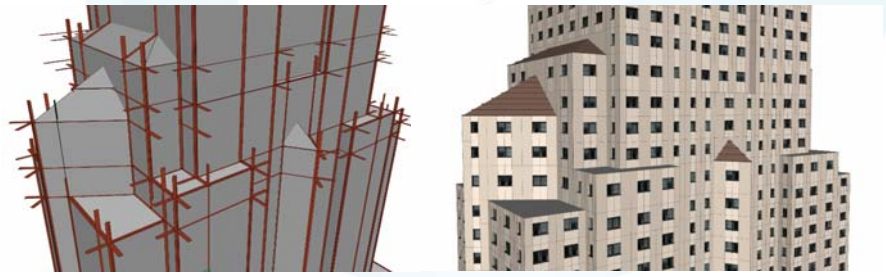
Occlusion Example

6: *tile : Shape.occlusion == "none"* ⇢ *window*
7: *tile : Shape.occlusion == "part"* ⇢ *wall*
8: *tile : Shape.occlusion == "full"* ⇢ ε



Shape Interaction: Snapping

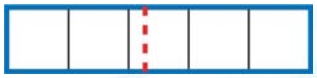
- Snap lines can be generated manually by the user
- .. and are created automatically on intersections



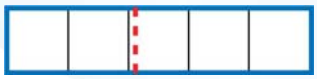
testing nearby important lines and planes in the shape configuration to align the elements

Shape Interaction: Snapping

- Subdivision split:
  1: floor ⇢ Subdiv(X,1r,1r,1r,1r){ B | B | B | B | B }



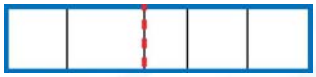
- Repeat split:
  1: floor ⇢ Repeat(X,0.2r){ B }



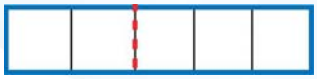
testing nearby important lines and planes in the shape configuration

Shape Interaction: Snapping

- Subdivision split:
  1: floor ⇢ Subdiv(XS,1r,1r,1r,1r){ B | B | B | B | B }

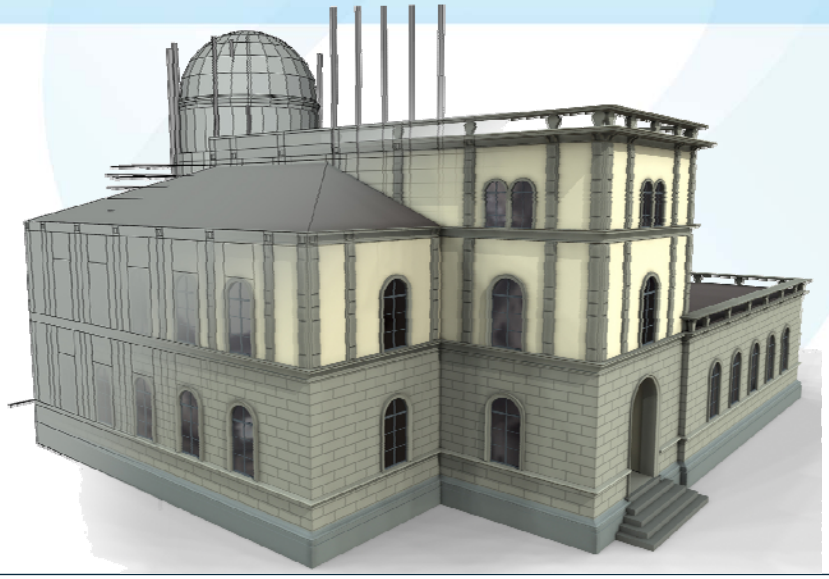


- Repeat split:
  1: floor ⇢ Repeat(XS,0.2r){ B }



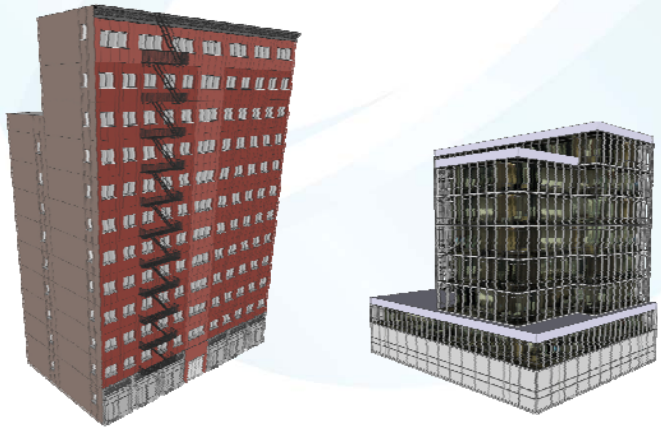
testing nearby important lines and planes in the shape configuration

Example



Snapping Example

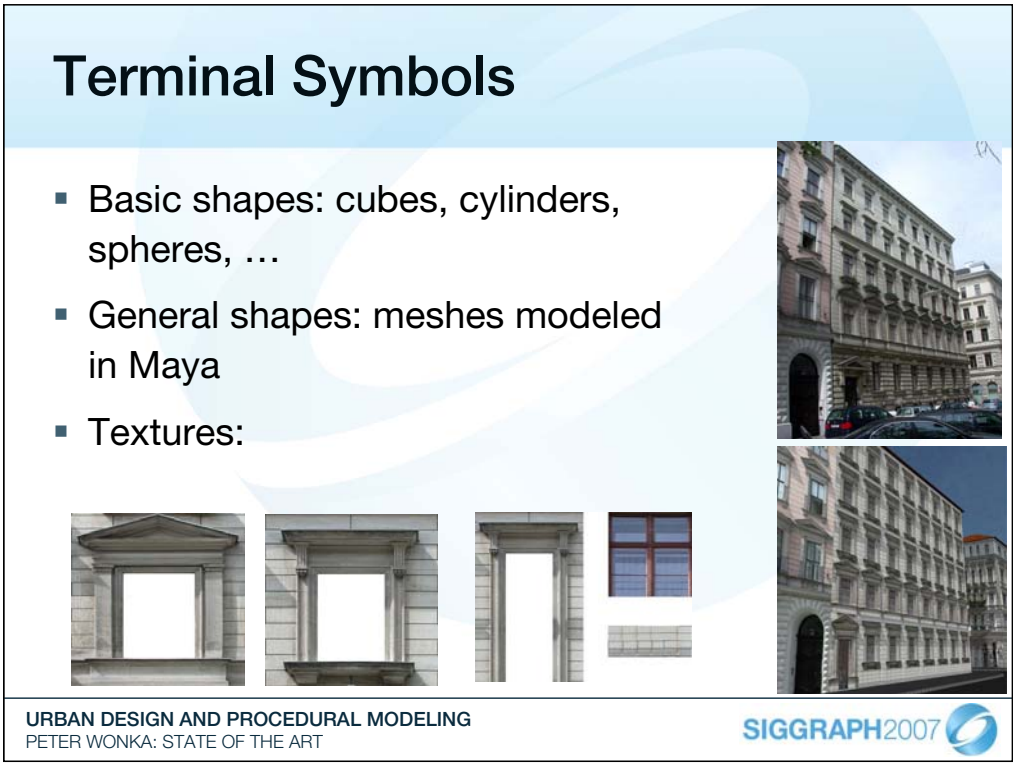
- Main application are office and high-rise buildings



High rise buildings and office buildings

Grammar Structure: Simple Temple

- temple → Subdiv("Y", …, … } { podium | columns | entablature | roof }
- columns → Repeat("X", …){ column }
- column → Subdiv("Y", …){ base | shaft | capital }
- base ⇢ I(corinthian_base)
- shaft ⇢ I(corinthian_shaft)
- capital ⇢I(corinthian_capital)
- entablature → Subdiv("Y", …)
  { architrave | frieze | cornice }
- architrave → Repeat("X", …) { architrave_tile }
- frieze → I(frieze)
- cornice → Repeat("X", …){ I(cornice_tile) }



It would be nice to model everything procedurally.

However, a trained modeler is much more efficient using Maya to model details.

In CGA shape

We chose to model most ornaments in Maya (such as Corinthian capitals).

Another idea would be to include generative mesh modeling (see next section)

I started out using texture blocks as a modeling primitive [Wonka2002]. The problem was that the texture blocks were not

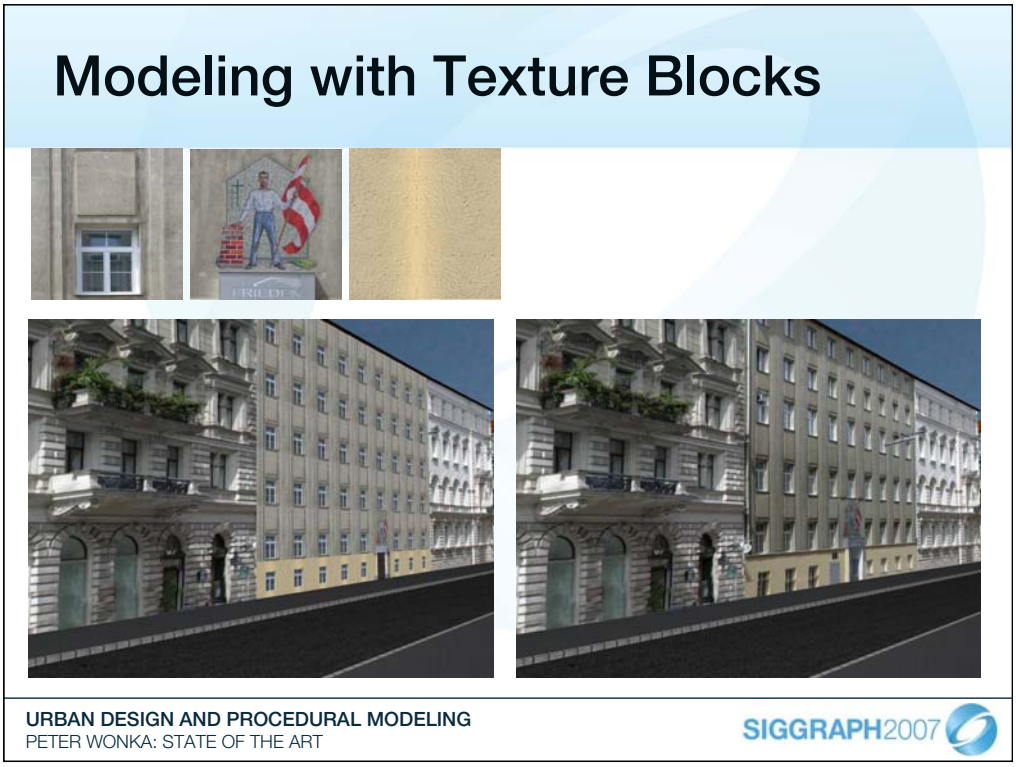
very configurable. It was very difficult to mix different blocks that did not come from the same façade.

A window block (the wall and the ornament surrounding the window) had to come from the same building as a door block.

So I could make variations in ordering window blocks from the same façade and changing the door locations.

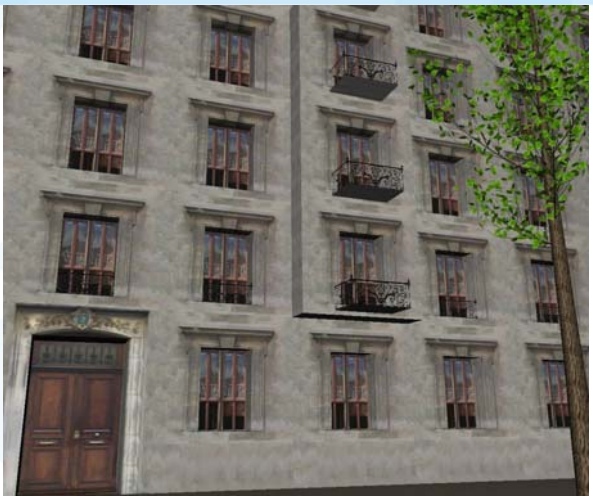
I could change exchange windows and doors more freely, but it was difficult.

[Wonka02] Wonka. Digitale Bausteine zur Fassadenmodellierung. Diplomarbeit, Institut fuer oertliche Raumplanung, Vienna University of Technology. 2002



Another example of reconstruction using texture blocks.

Procedural Modeling with Texture Blocks



Modeling Variations

Modeling Variations

- Add stochastic rule selection
- Add parameters
- Problems: (many rules → chaos)

Architectural Design

- Random selection cannot reproduce **design**
  - Example: horizontal and vertical coherence

URBAN DESIGN AND PROCEDURAL MODELING
PETER WONKA: STATE OF THE ART
SIGGRAPH2007

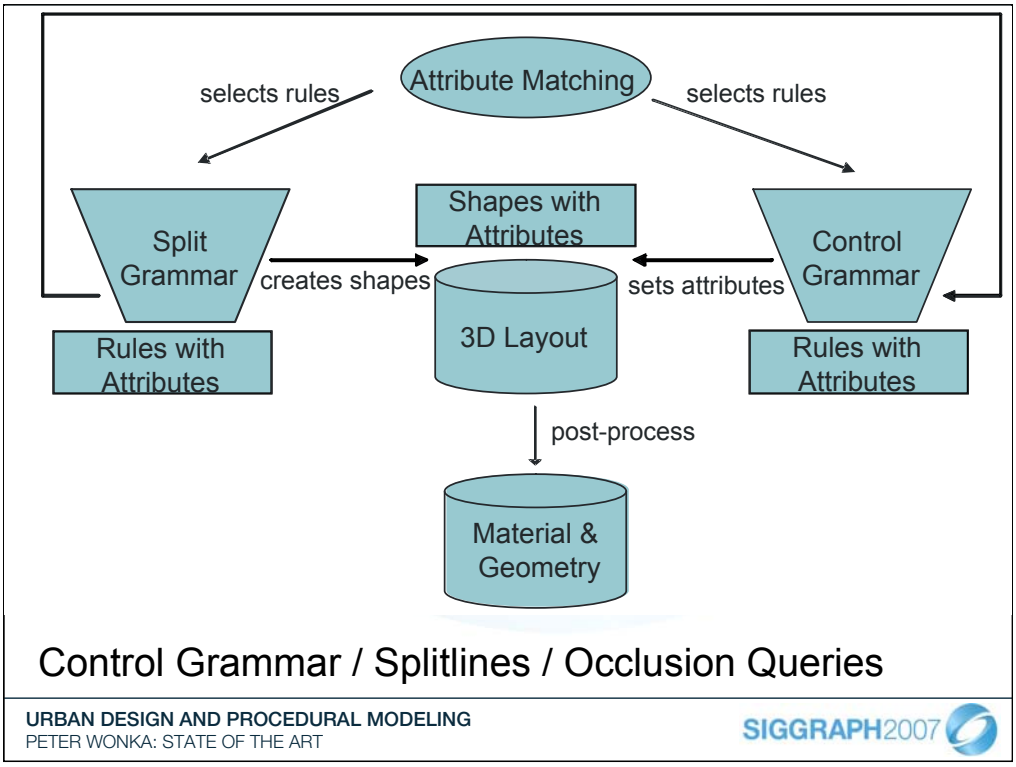
The problem of design becomes quickly a problem if stochastic rule selection is used. Typically, many elements are different in some aspects, but similar in others.

For example all windows in a row (floor) might share the same ornament on top of the window, but all windows in a column have the same width.

While these are trivial examples, there are typically several of these concepts interleaving. One of my main motivations was the paper: The city is not a tree by Christopher Alexander.

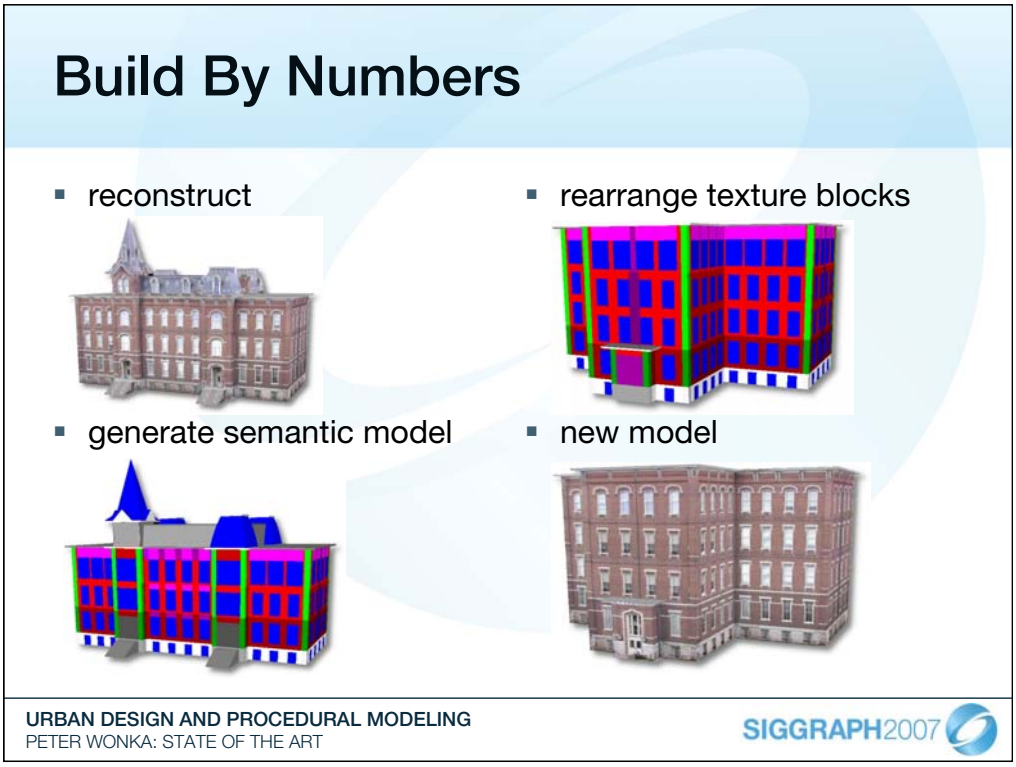
Alexander argues that most designs should not be constructed in a simple top down, tree like manner. Unfortunately, that

greatly conflicts with the goal of keeping the design simple.



CGA shape uses the control grammar introduced in Instant Architecture and additionally Snap lines and Occlusion Queries.

Procedural Techniques for Reconstruction



Bekins and Aliaga showed a very exiting framework that includes modeling with texture blocks [Bekins2005].

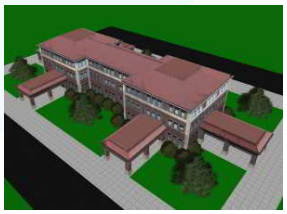
[Bekins05] D. Bekins, D. Aliaga, "Build-by-Number: Rearranging the Real World to Visualize Novel Architectural Spaces", IEEE Visualization, October, 2005

Abstract:

We combine image-based capturing and rendering with procedural modeling techniques to allow the creation of novel structures in the style of real-world structures. Starting with a simple model recovered from a sparse image set, the model is divided into feature regions, such as doorways, windows, and brick. These feature regions essentially comprise a mapping from model space to image space, and can be recombined to texture a novel model. Procedural rules for the growth and reorganization of the model are automatically derived to allow for very fast editing and design. Further, the redundancies marked by the feature labeling can be used to perform automatic occlusion replacement and color equalization in the finished scene, which is rendered using view-dependent texture mapping on standard graphics hardware. Results using four captured scenes show that a great variety of novel structures can be created very quickly once a captured scene is available, and rendered with a degree of realism comparable to the original scene.

Build By Numbers

- Captured model (left)
- Use semantics for color correction and occlusion removal (right)
- Another example:



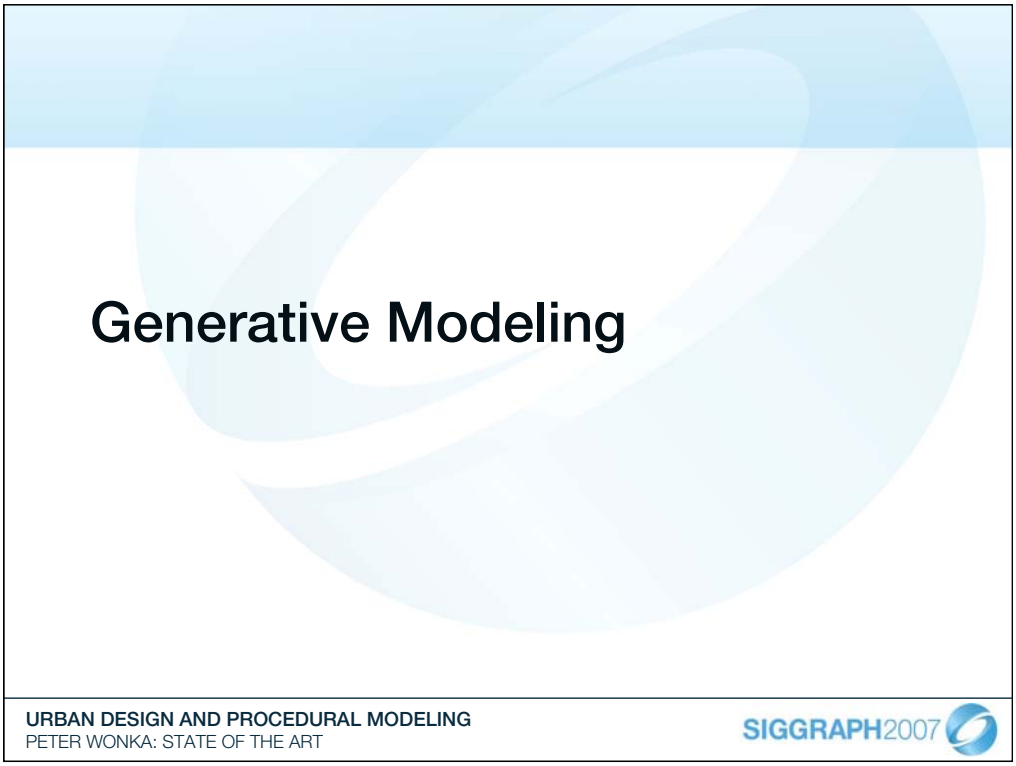
Modeling Facades from Single Images

- Input Image

- Procedural Model

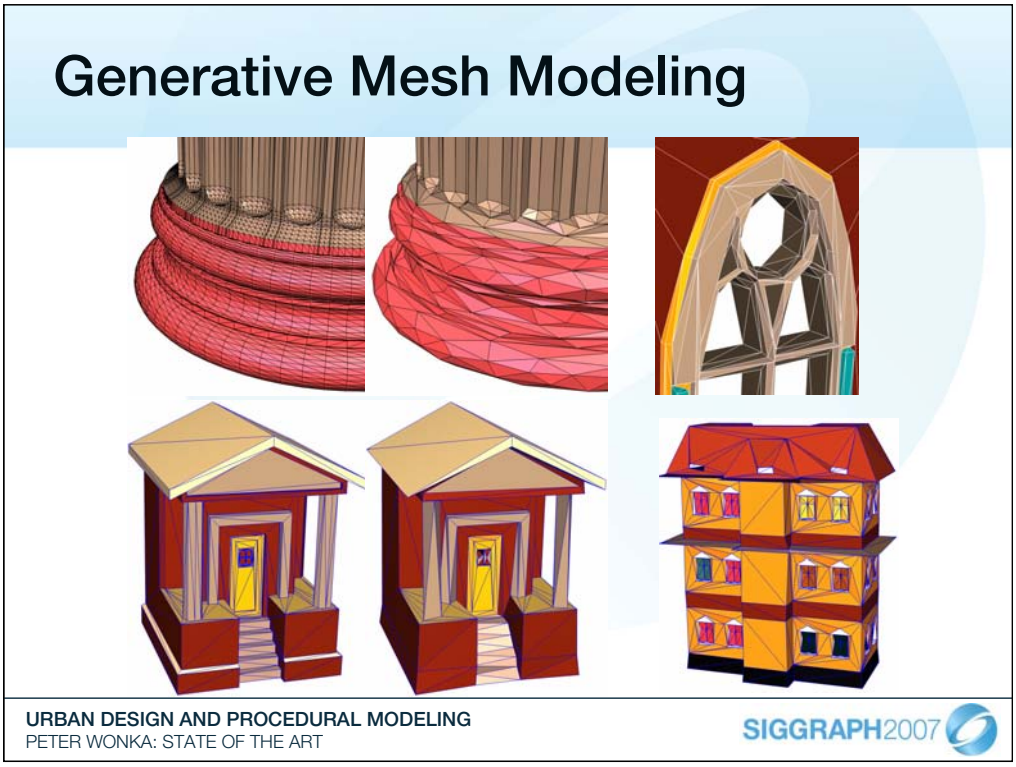
In contrast to the previous approach this work focuses more on automatic image segmentation and interpretation. In contrast, the previous paper focused on rearranging image parts that are segmented by hand.

Image-based Procedural Modeling of Facades. Pascal Müller, Gang Zeng (Eidgenössische Technische Hochschule Zürich), Peter Wonka (Arizona State University), Luc Van Gool (Eidgenössische Technische Hochschule Zürich and K.U. Leuven). conditionally accepted to Siggraph 2007.



website: www.generative-modeling.org

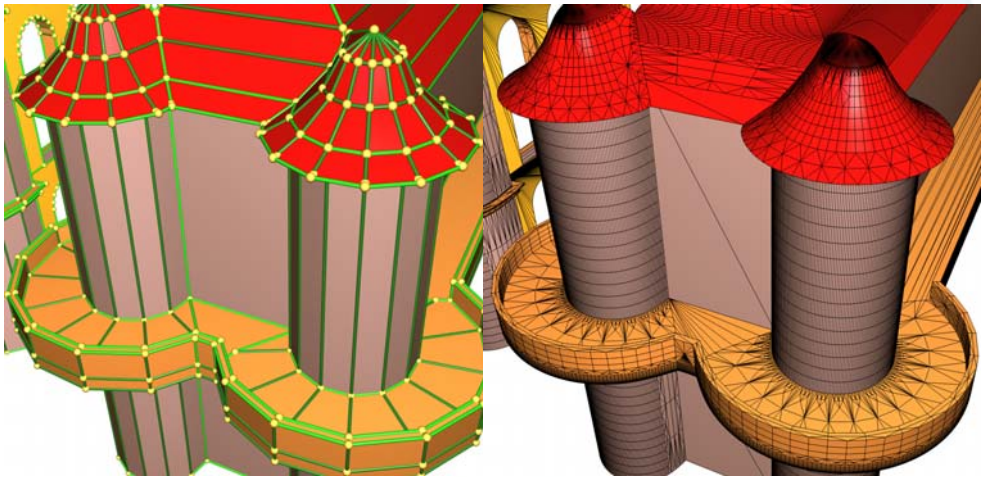
[Havemann05] Sven Havemann. Generative Mesh Modeling. PhD Thesis. TU Braunschweig.

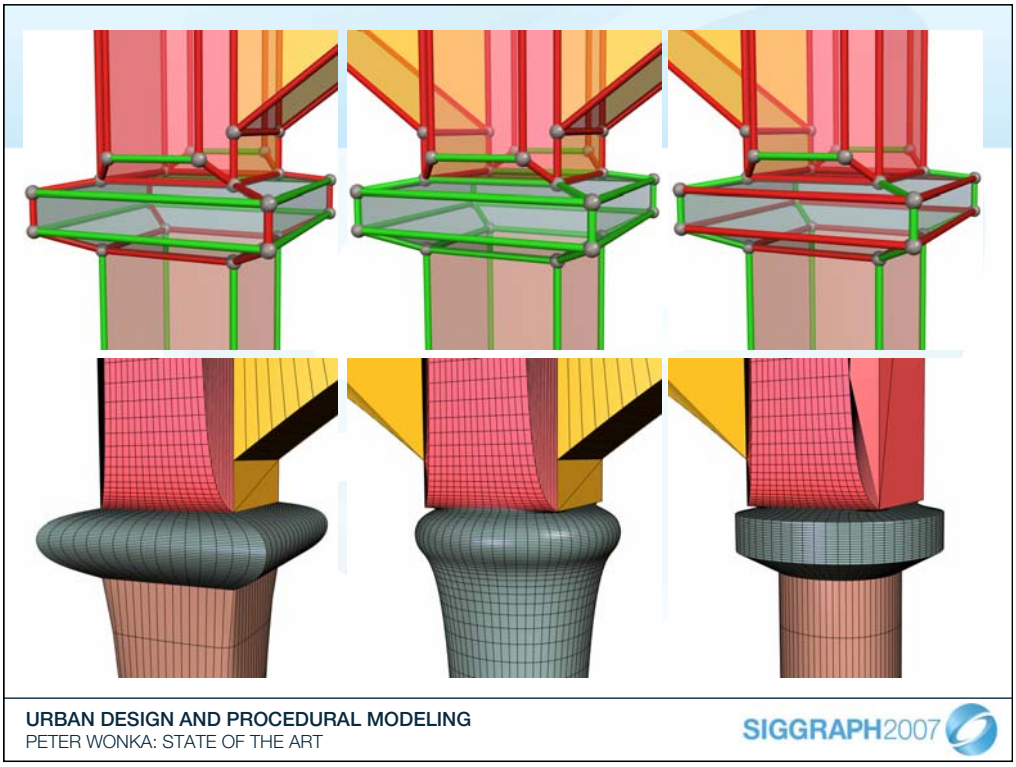


Generative Mesh Modeling can help to create details on basic meshes generated by a grammar. The grammar is not as well suited for complex ornaments and curved surfaces.

Generative Mesh Modeling can provide much additional detail. More information can be found in Sven Havemann's PhD thesis [Havemann].

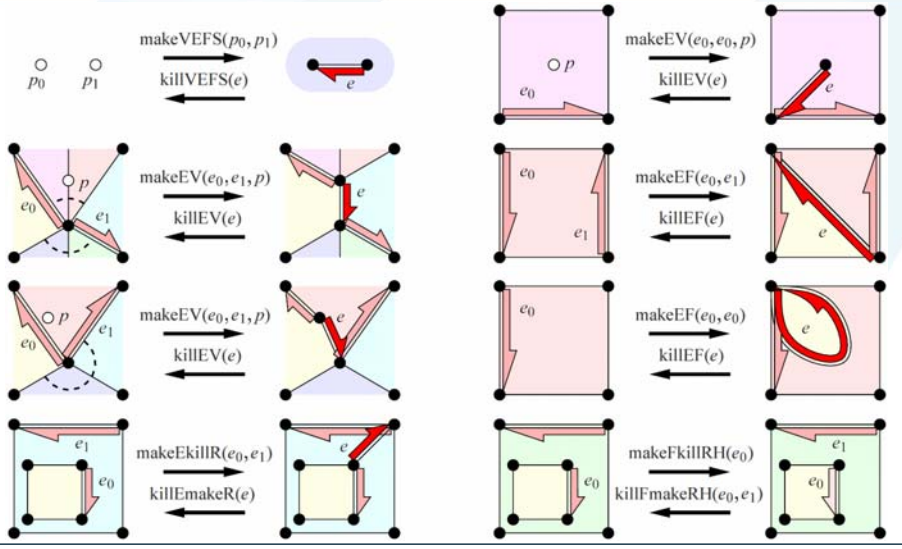
Generative Mesh Modeling





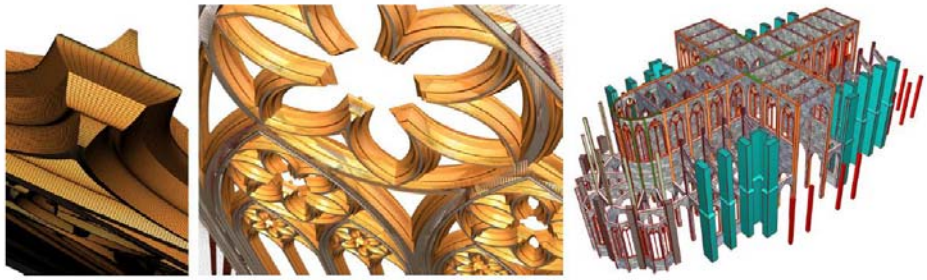
All column capitals have an identical control mesh, only the sharpness flags of the edges are different. A range of different shapes can be achieved by combining smooth transitions with sharp creases in various ways.

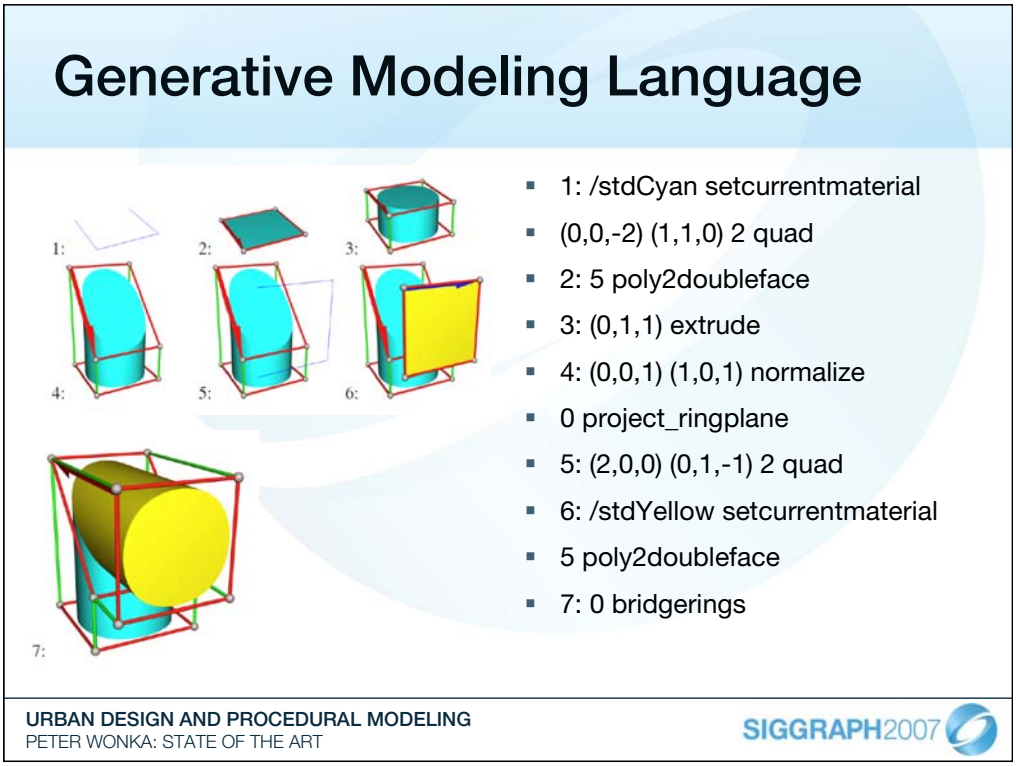
Euler Operations



Generative Modeling Language

- “Postscript for Meshes”
- Stack-based mesh modeling
- Operators take parameters from the stack





GML example for modeling with a stack-based language. 1: Parameters of the quad operator are the

midpoint (0,0,-2) and extension (1,1,0) of the quadrangle and 2 as a mode flag. The quadrangle is put on the stack

as an array of four points. 2: The polygon is converted to a mesh face, pushing a halfedge. 3: The extrude operator

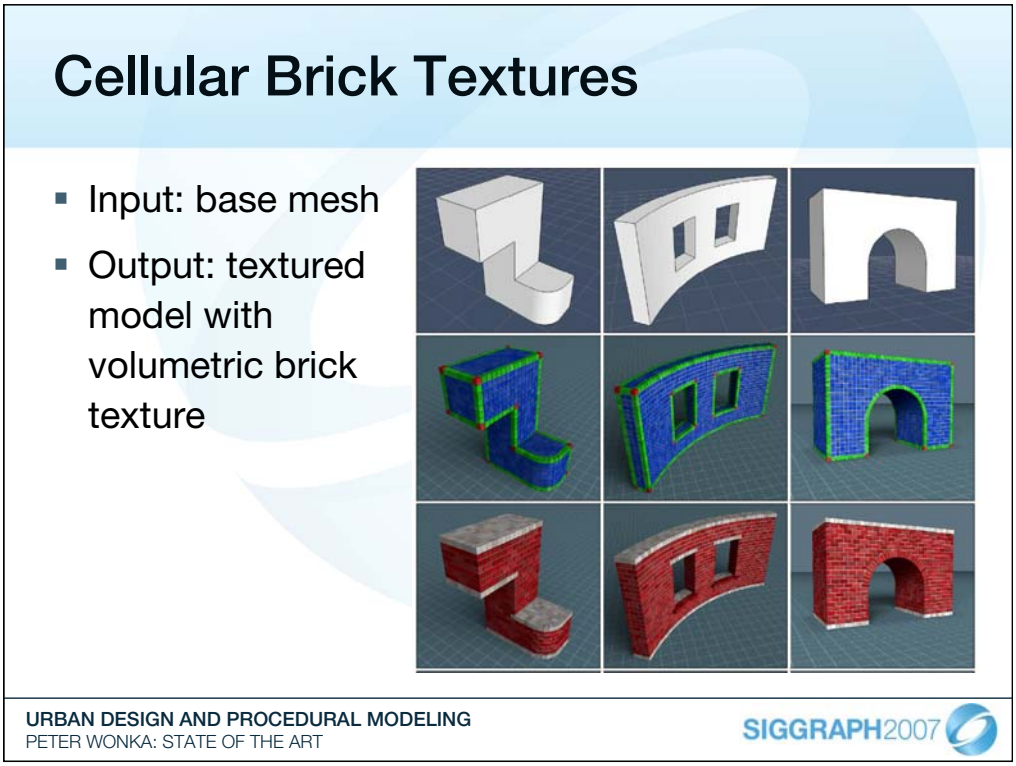
expects a halfedge and extension vector, and pushes the halfedge of the resulting face. 4: The face is moved by

projecting it in *z*-direction (0,0,1) onto a plane. 5,6: A second quad face is created. 7: The quad faces are bridged

with smooth edges. Thanks to combined B-reps the red/green mesh is sufficient to define the rounded pipe.

Modeling Gothic Windows

Cellular Textures



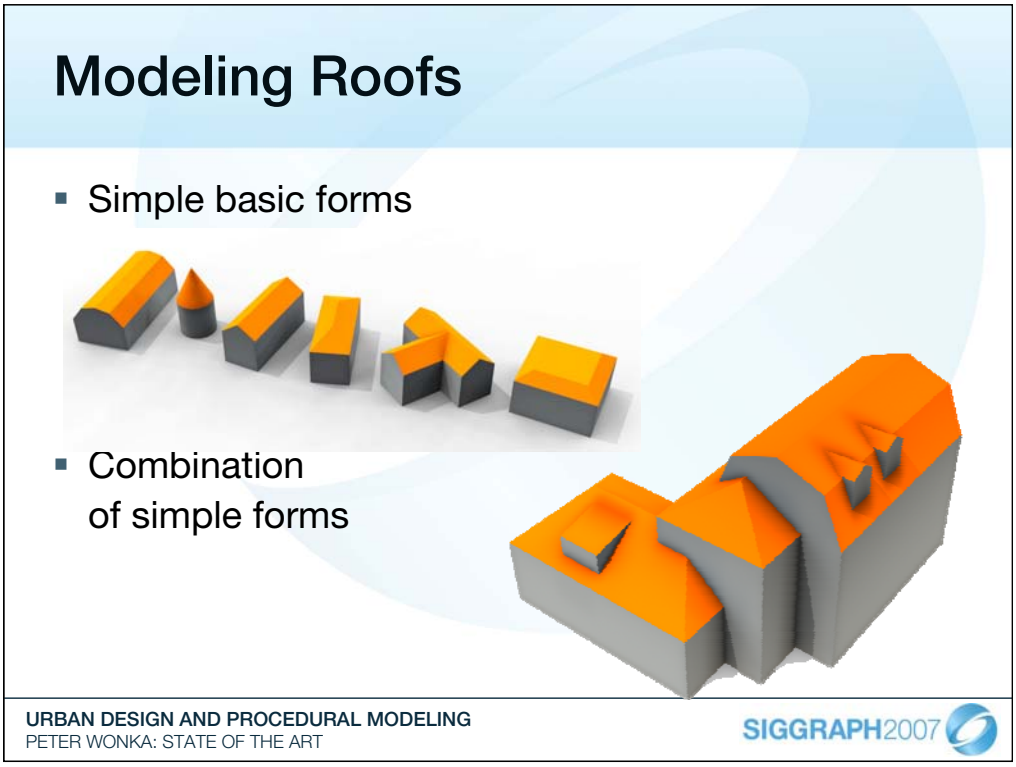
A very nice technique for augmenting a model with volumetric brick textures is described by

[Legakis01] Legakis, Dorsey, and Gortler. Feature-Based Cellular Texturing for Architectural Models. Siggraph 2001.

Abstract of the paper:

Cellular patterns are all around us, in masonry, tiling, shingles, and many other materials. Such patterns, especially in architectural settings, are influenced by geometric features of the underlying shape. Bricks turn corners, stones frame windows and doorways, and patterns on disconnected portions of a building align to achieve a particular aesthetic goal. We present a strategy for feature-based cellular texturing, where the resulting texture is derived from both patterns of cells and the geometry to which they are applied. As part of this strategy, we perform texturing operations on features in a welldefined order that simplifies the interdependence between cells of adjacent patterns. *Occupancy maps* are used to indicate which regions of a feature are already occupied by cells of its neighbors, and which regions remain to be textured. We also introduce the notion of a *pattern generator* — the cellular texturing analogy of a shader used in local illumination — and show how several can be used together to build complex textures. We present results obtained with an implementation of this strategy and discuss details of some example pattern generators.

Modeling Roofs



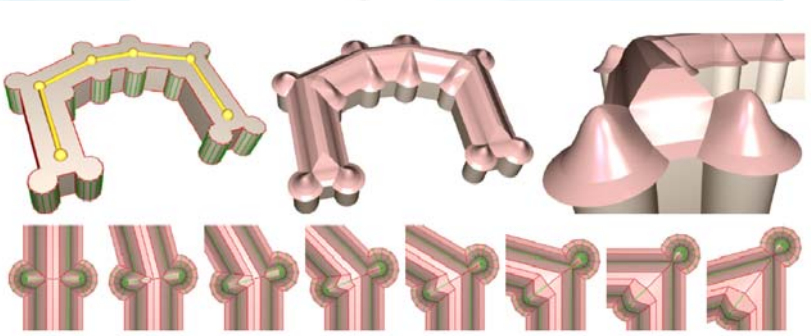
[Müller06] describes a method to procedurally generate roofs using a combination of basic roof shapes using the shape grammar

CGA shape. Individual bricks can also be generated by the grammar.

Modeling Roofs

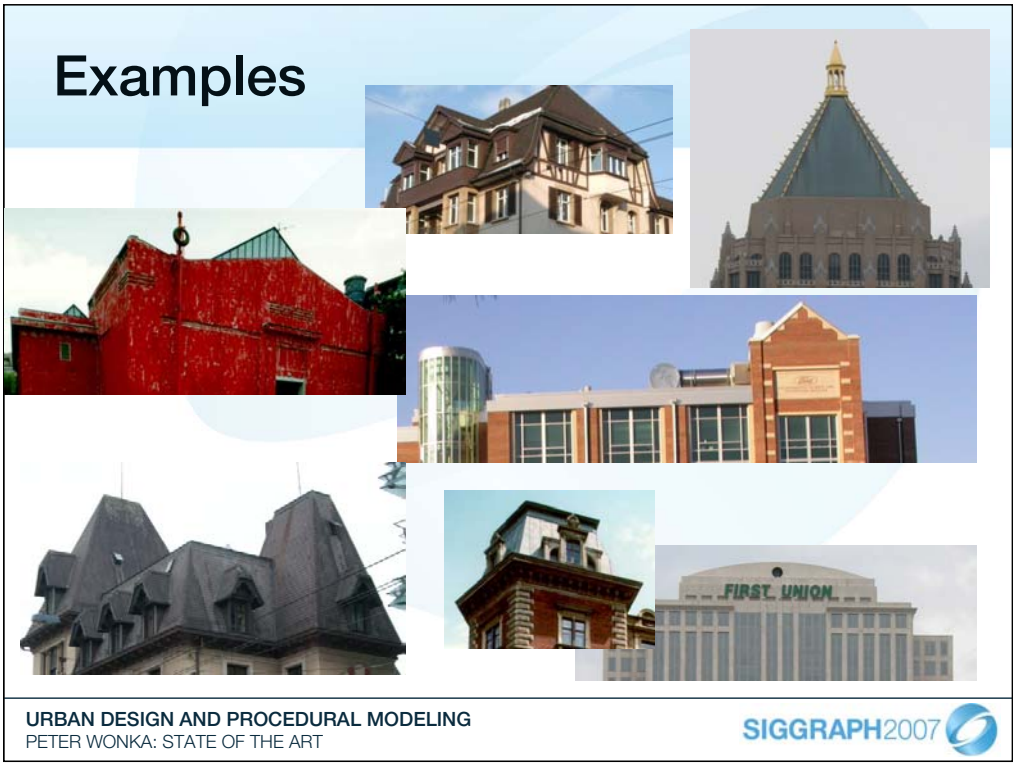
Roof Construction Algorithms

- Straight Skeleton [Aichholzer95][Eppstein99]
- Example from Sven Havemann's PhD thesis:



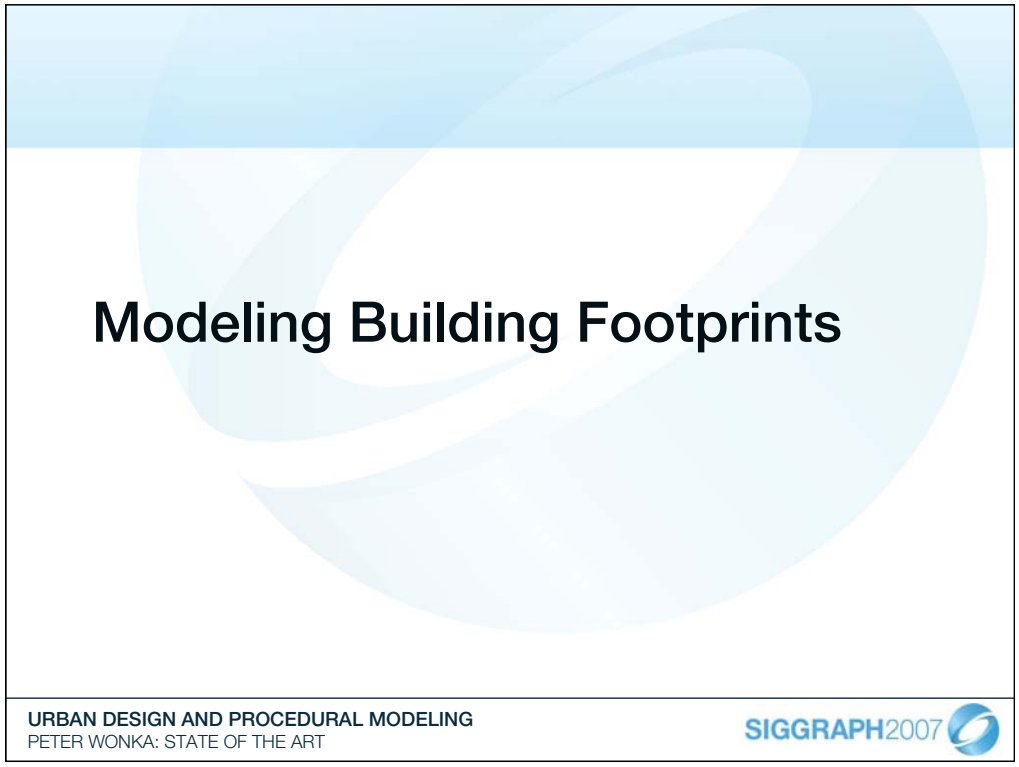
[Aichholzer 1995] Aichholzer, Aurenhammer, Alberts, and Gaertner. A novel type of skeleton for polygons. J.UCS: Journal of Universal Computer Science 1, 12 (1995), 752–761.

[Eppstein99] Eppstein, Erickson. 1999. Raising roofs, crashing cycles, and playing pool: applications of a data structure for finding pairwise interactions. In Proceedings of the 14th Annual Symposium on Computational Geometry, ACM Press, 58–67.



Some real-life images show that there are still interesting challenges. Some of these roofs are hard to generate procedurally.

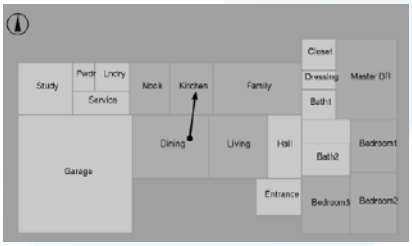
One of the main problems is the complex arrangement of different architectural elements in a consistent manner.



[Harada95] Harada, Witkin, and Baraff. Interactive physically-based manipulation of discrete/continuous models. Siggraph 1995.

Floorplan Modeling

- Given: Floorplan Model
- Problem: How can we interactively manipulate the floorplan by clicking on rooms
  and changing room sizes and location?



Harada et al. [Harada1995] describe a method for interactive floor plan manipulation.

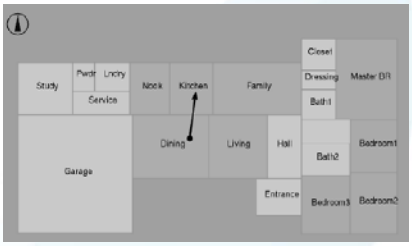
A large part of this paper is motivated by the shape grammar literature.

The main topic of the paper is to mix continuous and discrete changes during the interaction.

Using the system, the user can pull on either a wall or the center
of a room to change the position and size of a room.
When the user pulls on an interior wall, the exterior
walls of the layout remain fixed. When the user pulls on an exterior
wall, the entire layout is scaled.

Continuous / Discrete Interaction

- Continuous:
  - controlled by geometric constraints and forces
  - typically smaller changes
- Discrete:
  - when continuous changes do not work any more
  - typically larger changes
  - changes in structure



The interaction works as follows:

The user applies forces

The forces work in the continuous interaction mode apply forces and goto 1

3) The forces are under the threshold suggesting discrete changes goto 1

4) Trigger discrete search

The architecture is encoded as a sub region tree. The following discrete operations were implemented

(Note: verbatim quote from the paper):

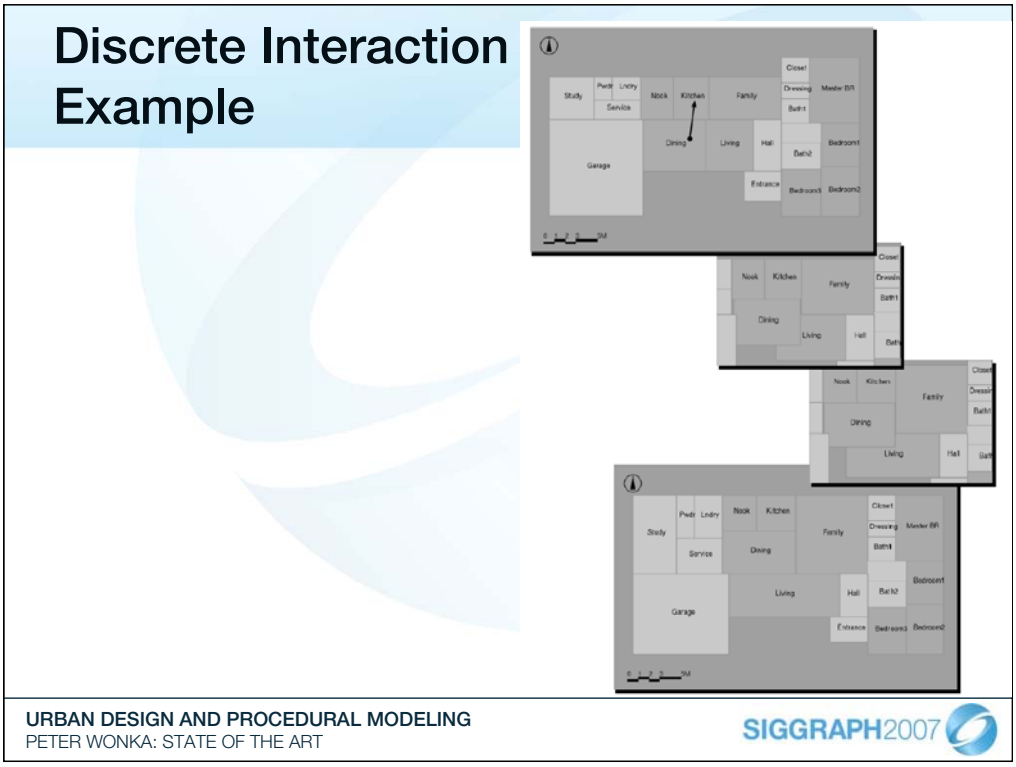
The move transformation takes as its arguments a node to be moved, the new node under which it should be placed, and an integer giving its position in the list of siblings at the new location. The target node is moved to the new location. If it is a nonterminal node, the subtree under it moves as well.

The swap transformation takes as arguments two nodes, and exchanges their positions in the subregion tree, equivalent to two move operations.

The group transformation takes as arguments two nodes, which must be spatially adjacent. The subregion tree is rearranged to make the nodes become siblings in the subregion tree. We use this operation when four subregions meet at or near one point to change the order of subdivisions (i.e., from vertical first and horizontal second to horizontal first and vertical second, or vice versa).

The rotate transformation takes a single node as its argument. It does not change the structure of the subregion tree. Rather, it swaps the x and y values of the minimum and maximum dimensions

assigned to the corresponding rectangle. This operation is only applicable to leaf nodes.



Architectural room layout.
(top) The user pulls on the dining room, and a discrete change occurs.
(middle) Two intermediate frames from the resulting animated transition.
(bottom) The new configuration.

The Interface to Rendering

This image was selected for the inner cover of the Siggraph 2006 proceedings (special issue of ACM TOG).

A lot of effort had to be invested in the rendering. Creating shaders and setting up lighting is fairly difficult.

Also: Can you find the teapot?

Textures

- Dirt Map / Noise Maps
- Diffuse / Specular Maps
- Displacement Maps

There are several texture / parameter maps that are necessary for rendering. Therefore the models need to have multiple texture coordinates for each polygon.

How to setup lighting

- Cheap OpenGL Version
  - Directional Light (sun)
  - Local light at viewer position (OpenGL)

I started rendering models with OpenGL like this.

How to setup lighting

- Ambient Occlusion
  - distance cutoff similar to average building height
- Image-based lighting
  - Good for single building
- Sky light models

A good starting point for image-based lighting:

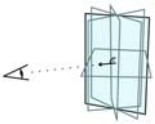
@inproceedings{280864, author = {Paul Debevec}, title = {Rendering synthetic objects into real scenes: bridging traditional and image-based graphics with global illumination and high dynamic range photography}, booktitle = {SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques}, year = {1998}, isbn = {0-89791-999-8}, pages = {189--198}, doi = {http://doi.acm.org/10.1145/280814.280864}, publisher = {ACM Press}, address = {New York, NY, USA}, }

A good starting reference for Sky light models

A Practical Analytic Model for Daylight, by Preetham et al. SIGGRAPH 99

RenderMan Usage

- Massive city models
  - Brute force AO via renderfarm of 60 Linux boxes (via NFS)
  - Delayed read archives (one per building)
  - Binary and compressed RIBs (one per building per lod)
  - Several UV-sets per mesh
- Procedural facade shaders via DSOs
- Vegetation
  - Tree library (Xfrog)
  - Low LoD with Billboard fans (++)



Alfred is king

Experiences with RenderMan 12:

- Did not like large-scale poly models (required huge amount of memory i.e. frames have to be ‘sliced’ etc)
- Baking of AO problematic for unstructured meshes (brickmaps as well as point clouds)

Real-time Ray Tracing

The next three rendering methods are all exiting possibilities for rendering the generated city models. We hope to get some more experience over the summer.

Real-time GPU Rendering

TODO

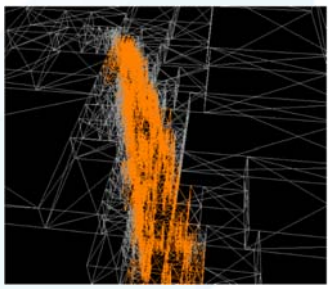
Monte-Carlo Ray Tracing

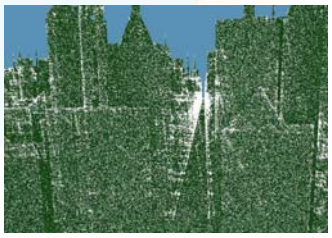
TODO

Acceleration Strategies

Visibility

- Visibility is very important when view point is at eye height (walking, driving, …)
  - online:
    - occlusion queries
    - [Bittner 2004], [Guthe 2006]
  - offline
    - pre-compute potentially visible sets
    - [Wonka 2006]





There are too many references to list. I selected the ones that I am most familiar with and that are fairly recent.

If the city is modeled on a flat terrain these techniques are almost sufficient by themselves to achieve high frame rates.

[Bittner 2004] Coherent Hierarchical Culling: Hardware Occlusion Queries Made Useful

Jiri Bittner, Michael Wimmer, Harald Piringer, Werner Purgathofer.

Computer Graphics Forum (Proc. Eurographics 2004) 23(3):615-624, September 2004

[Guthe 2006] Near Optimal Hierarchical Culling: Performance Driven Use of Hardware Occlusion Queries

Michael Guthe, Ákos Balázs, and Reinhard Klein.

Eurographics Symposium on Rendering 2006.

[Wonka 2006] Guided Visibility Sampling

Peter Wonka, Michael Wimmer, Kaichi Zhou, Stefan Maierhofer, Gerd Hesina, Alexander Reshetov

ACM Transactions on Graphics, 25(3):494-502, July 2006. (Siggraph Proceedings)

Requirements for Visibility

- Spatial sorting in hierarchical data structure
  - e.g. kd-tree
- Occluder selection
  - compute LODs as occluder

Image-based Simplification

- Replace geometry with textured polygons
  - Which geometry should be replaced?
  - How to place the polygons?
- Problems
  - Relighting
  - Interaction with dynamic objects
  - Memory consumption

Two examples of impostor placement are:

[Jeschke 2005] Automatic Impostor Placement for Guaranteed Frame Rates and Low Memory Requirements

Stefan Jeschke, Michael Wimmer, Heidrun Schumann, Werner Purgathofer

In Proceedings of ACM SIGGRAPH 2005 Symposium on Interactive 3D Graphics and Games, pages 103-110. April 2005.

[Decoret 1999] Multi-layered Impostors for Accelerated Rendering

Xavier Décoret, Gernot Schaufler, François Sillion, Julie Dorsey

Computer Graphics Forum (Proc. of Eurographics '99) - Sep 1999

The following paper is probably the best start:

Billboard Clouds for Extreme Model Simplification

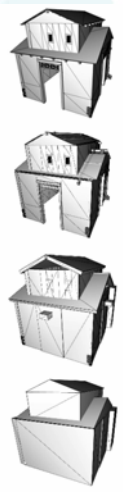
Xavier Décoret, Frédo Durand, François Sillion, Julie Dorsey

Proceedings of the ACM Siggraph - 2003

These simplification techniques pose little requirements on the models. They can work with almost any model.

Geometric Simplification

- Automatic generation of LoD's via semantics
  - How to include semantic information?
  - How can geometric simplification work for buildings?



URBAN DESIGN AND PROCEDURAL MODELING
PETER WONKA: STATE OF THE ART
SIGGRAPH2007

Traditional geometric mesh simplification is difficult. The models that are created are not nice 2-manifolds.

I would claim that all existing simplification methods are not very successful with a large variety of building models.

In this section we demonstrate practical procedural modeling concepts. First we explain the workflow within our tool called CityEngine and give general tips for the organization of city projects, architectural rulesets and terminal element models. Then we show in three different stages how architectural real-world structures can be encoded: (1) how to model single buildings, (2) how building parameters can be found, and (3) how to model buildings with citywide variation. Afterwards we will show example applications in archaeology, architecture and entertainment. The background image shows a rendering of our procedurally reconstructed Pompeii.

Contents

- The CityEngine System
- Grammar-based Encoding of Buildings
- Parameterization of Buildings
- Stochastic Modeling
- Reconstruction of Archaeological Sites
- Applications in Architecture
- Procedural Modeling in Entertainment

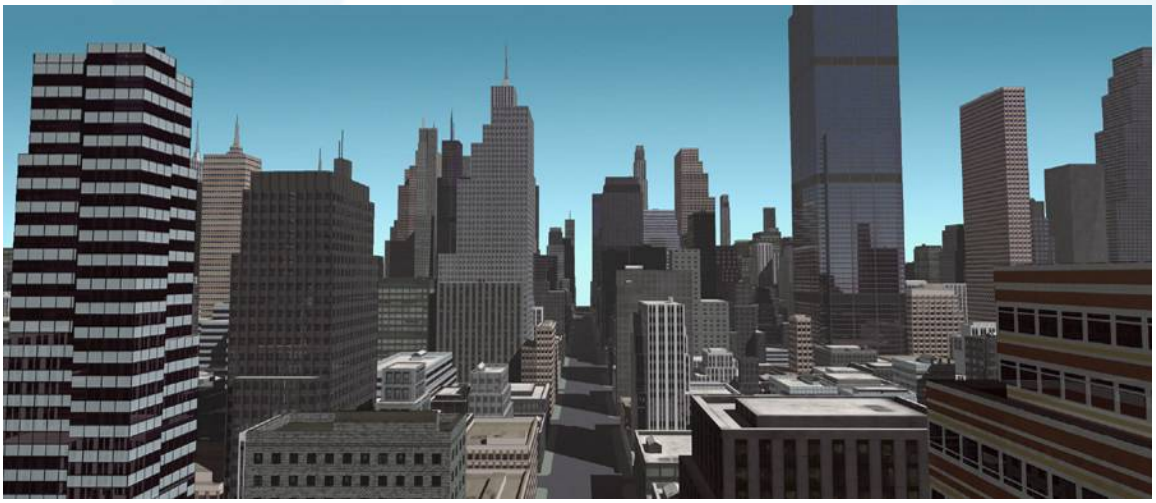
Note:

Although this course part is based on the CityEngine, the presented concepts can be applied in a general manner. For some real-world scenarios, e.g. developing one game, full blown solutions like the CityEngine are often not needed. For example, a shape grammar implementation may not to be needed and the procedural building appearances can be implemented directly in the source code instead.

Generally speaking, the CGA Shape grammar can serve as a generic research tool to formulate architectural procedural modeling problems and solutions in computer graphics.

The *CityEngine* System

Software tool for the procedural generation of detailed large-scale urban environments



Motivation and Goals:

- Encoding the structural, spatial and functional complexity of cities and buildings
- Efficient creation of detailed building models at low cost
- Many applications in entertainment, simulation, archaeology and architecture

The CityEngine system, developed at ETH Zürich, is able to:

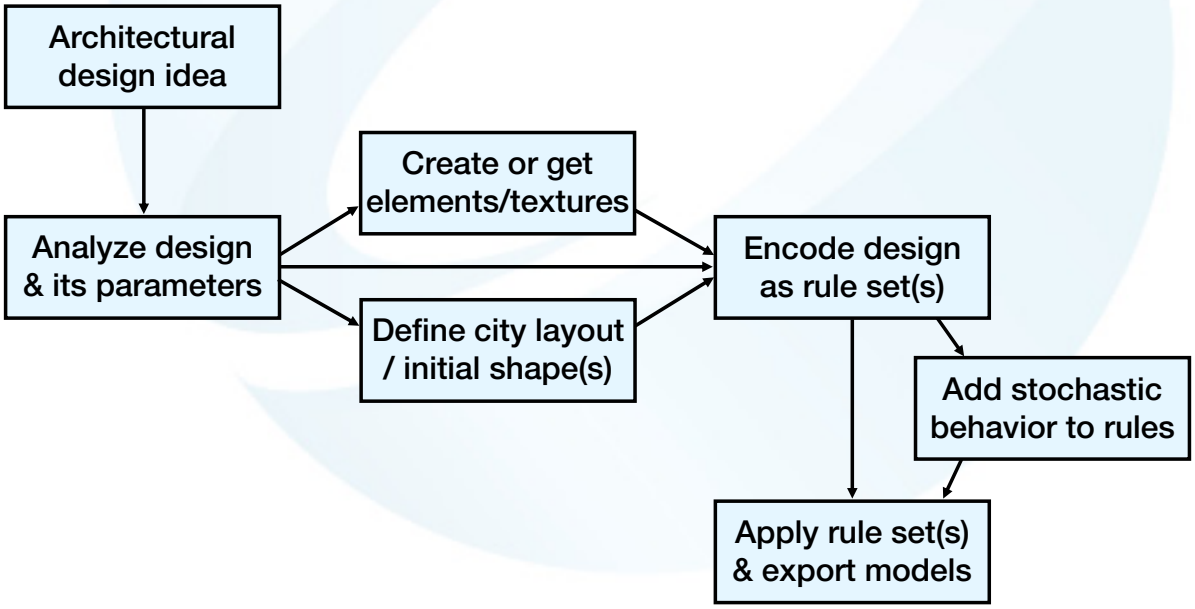
- Create unique large-scale city models as well as very detailed building models through procedural techniques.
- Evaluate and interpret all kinds of architectural rules. Different libraries are already available, e.g. a rule set for classical architecture.
- Perform rule-based distribution of (urban) vegetation.
- Interactively explore and alternate models due to a sophisticated user-interface.
- Import and export of GIS data and meshes in a variety of formats.

CityEngine references:

- P. Müller, P. Wonka, S. Haegler, A. Ulmer and L. Van Gool. 2006. Procedural Modeling of Buildings. In Proceedings of ACM SIGGRAPH 2006 / ACM Transactions on Graphics (TOG), ACM Press, Vol. 25, No. 3, pages 614-623.
- Y. Parish and P. Müller. 2001. Procedural Modeling of Cities. In Proceedings of ACM SIGGRAPH 2001, ACM Press / ACM SIGGRAPH, New York. E. Fiume (ed), COMPUTER GRAPHICS, Annual Conference Series, ACM, pages 301-308.
- http://www.vision.ethz.ch/pmueller/wiki/CityEngine

The CityEngine System

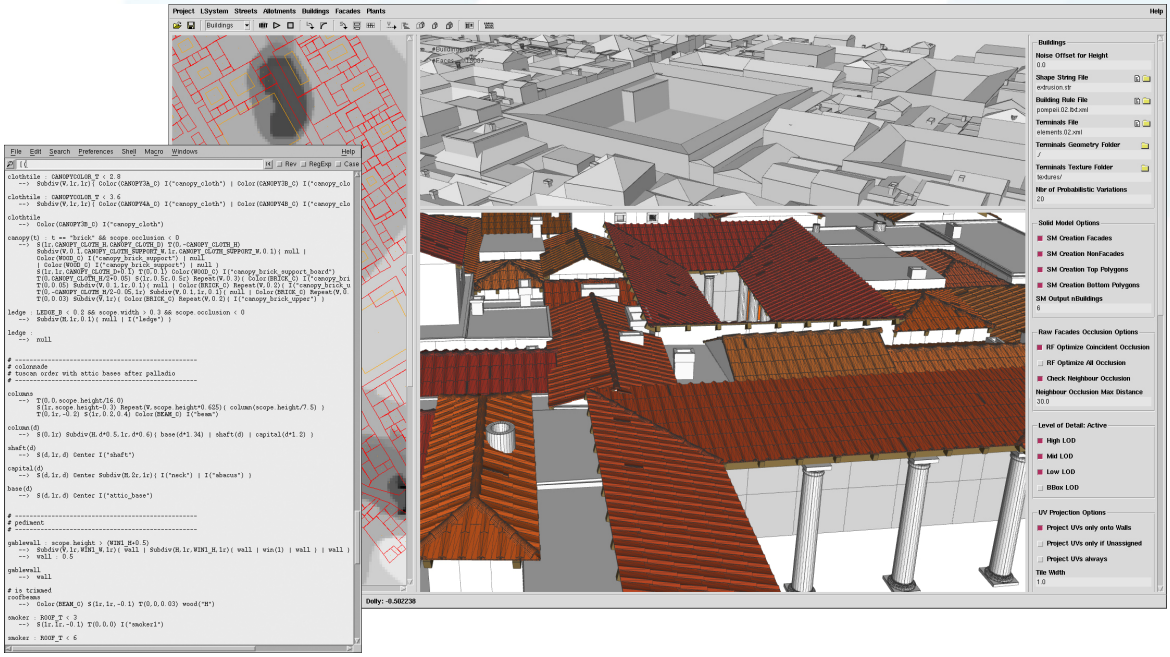
Workflow: Overview



Note: In this course part we concentrate only on procedural buildings i.e. the city layout features of the CityEngine are not treated herein.

The CityEngine System

Workflow: User Interface



URBAN DESIGN AND PROCEDURAL MODELING
PASCAL MUELLER: APPLIED PROCEDURAL MODELING

SIGGRAPH2007

CityEngine User Interface:

- Text editor for editing rules (front window)
- OpenGL preview with many different view modes (similar to Maya e.g. concerning ease of use, camera handling etc). User-friendly displaying of different representations (e.g. derivation steps, different grammar formats, etc) is extremely important for the visual debugging of rules.
- GIS Viewer/Eidtor to display, navigate and edit city maps and layouts (main window, left panel). Important for largescale scenarios.
- Configurable Tcl/Tk user interface (optimized for fast r&d)
  - Customizable Parameters incl Sliders etc
  - Buttons, Help, Status line, Component Editors, …
  - Scripting: Tcl can be used during runtime (like e.g. Mel in Maya)

The CityEngine System

The Elements Repository

- No distinction between objects, textures or shaders i.e. tag-based format incl. cross-linkage (Collada)
- Organized in categories, types and instances

Example: Facades/Windows/SashWindow, type 3, instances 1-7

- Managed in SVN repository i.e. multi-user capable

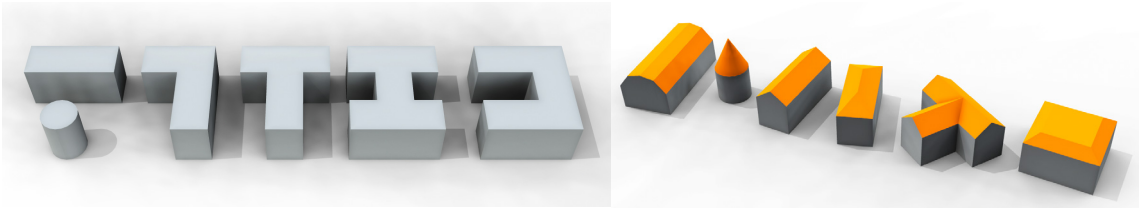
Our current Subversion tree organization (just the first two levels):

- Environment
  - Advertisement (textures of advertising panels)
  - Panoramas (environment maps for window reflections)
  - Pools (textures for swimming pools)
  - …
- Facades
  - Columns (3D models for Doric, Ionic and Corinthian columns - can be used everywhere, not only in classical architecture)
  - Doors (3D models and textures of doors)
  - Ledges (3D models and textures of ledges)
  - Walls (tileable textures for walls i.e. bricks etc)
  - Windows (3D models and textures of windows)
  - …
- General
  - Debug (3D objects for visual debugging)
  - Dirtmaps (generally applicable and tileable dirt textures)
  - Primitives (the basic 3D primitives)
  - Stonework (3D models and textures of stone elements)
  - Woodwork (3D models and textures of wood elements)
  - …
- Interior …
- Misc …
- Roofs …
- Streets …
- Vegetation …

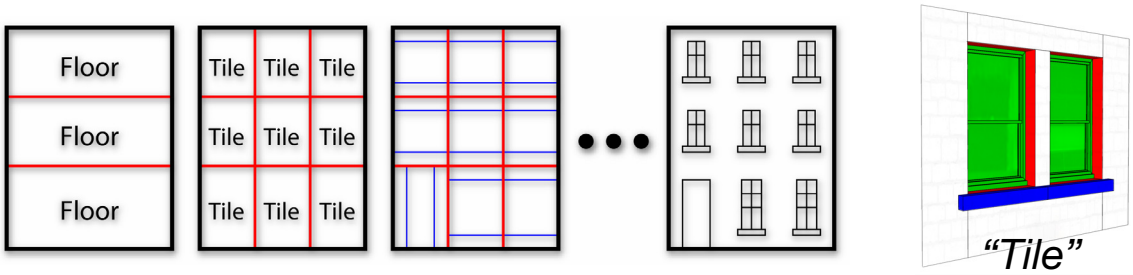
Our library is growing with each project and reusability of the elements is implicitly guaranteed. A good, flexible, open and expandable element library organization is the key to efficient modeling with CGA Shape!

Encoding of Buildings

- *Mass*: rule-driven assembling of primitives



- *Surface*: usual subdivision scheme



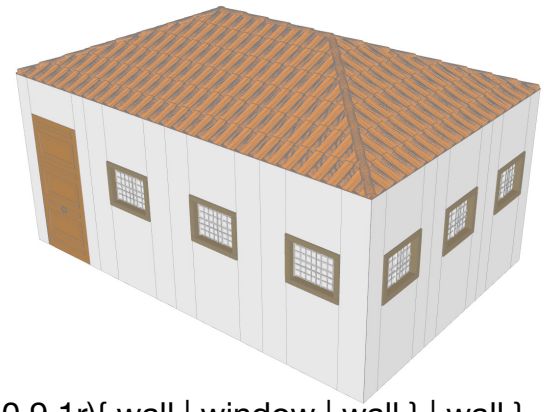
Building mass models are most naturally constructed as an assembling of simple solids to model basic building types: *L*, *H*, *U, T…* The same concept is applied on roofs which are represented as parameterized primitives. Basic roof types are: *Gable, hip, mansard*...

General subdivision scheme for facades: *facade →floor →tile →wall/window*

- Bottom right: A facade tile consisting of wall (White), sill (blue), openbox terminal (red) and window element (green).

Encoding of Buildings

Simple Building Example



1: footprint ⇢ S(1r,4,1r) facades
T(0,4,0) Roof(hip,17){ roof }

2: facades ⇢ Comp(f,sides){ facade }

3: facade : Scope.ry == 0 ⇢
Subdiv(x,3,1r){ entrance | tiles }

4: facade ⇢ tiles

5: tiles ⇢ Repeat(x,2){ tile }

6: tile ⇢
Subdiv(x,1r,1.3,1r){ wall | Subdiv(y,2r,0.9,1r){ wall | window | wall } | wall }

7: window ⇢ S(1r,1r,0.4) I(window)

8: entrance ⇢ Subdiv(x,1r,1.3,1r){ wall | Subdiv(y,3,1r){ door | wall } | wall }

9: door ⇢ S(1r,1r,0.3) I(door)

10: wall ⇢ I(plane)

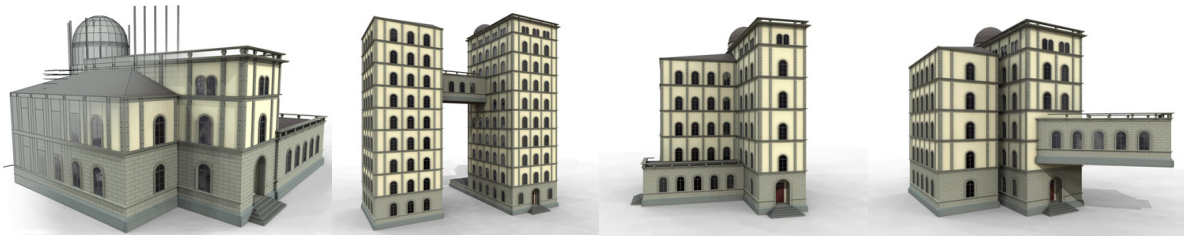
This section details an introductory example of modeling with our shape grammar. The example grammar will generate a simple generic building including its roof from an arbitrary building footprint. This building footprint is the axiom for the rule set.

The first rule extrudes the footprint 4m in y direction, creates a shape *facades* and places a hipped roof shape on top (with roof angle 17°). Rule 2 breaks down the mass model into its side faces resulting in *facade* shapes. Note the plural versus singular notation concept. Rule 3 identifies the front building side to place an entrance, otherwise only windows are generated with rule 4. Rule 6 exemplifies the typical subdivision scheme of a tile.

Encoding of Buildings

Size-Independent Rule Designing

- For reusability and stochastic applications it is important that the rules are as generic as possible
- Usage of repeat splits and relative dimensions whenever appropriate is strongly recommended



For relative dimensions in subdivision splits, the design question is: which are the floating parts? For example, in a tile: have the wall elements fixed sizes and the size of the window adapts or vice-versa? In our experience, most of the time the wall is in relative dimensions and the window size is fixed (since mass-customized glass dimensions are not very common yet), but these are specific design choices which the rule writer has to determine i.e. to our knowledge, no general recipe can be found in architecture to solve this problem. But generally, writing rules with using relative dimensions is very practical and efficient since no rest calculations have to be done.

The Candler building is located in Atlanta and was the former headquarter of Coca-Cola. The seventeen story building was constructed between 1904 and 1906 under the direction of architects George E. Murphy and George Stewart. It is a representative of the Beaux Arts style which is notable for patterns within patterns (hierarchy of spaces), high parapets, projecting facades, pilasters and garland or flower ornamentation. The depicted CGA Shape reconstruction of this office building illustrates how facade designs can vary in appearance, ratios, dimensions and patterns.

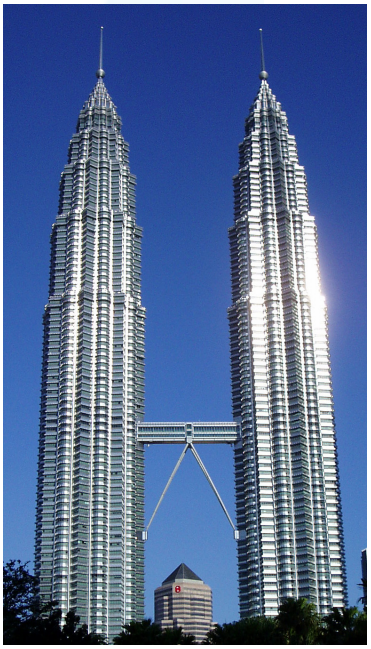
This example uses the SashWindow elements shown in the previous slide. Ruleset excerpt:

```
...
12: windowopening →
    [ S(Scope.sx+0.3,0.2,0.1) T(-0.15,-0.2,-0.1) sill ] [ S(1r,1r,0.4) I(OpenCube) Comp(f,all){ wall } ] T(0,0,0.4) window
13: window → S(1r,1r,0.1) I(SashWindow,3,-1)
...
```

Note the overloaded insert shape operation I(category,type,instance) allowing for intelligent access to the elements repository. The last attribute, the instance number, is set to -1 to indicate that a random instance of the available ones can be taken.

Encoding of Buildings

High-rise Buildings with Setbacks



URBAN DESIGN AND PROCEDURAL MODELING
PASCAL MUELLER: APPLIED PROCEDURAL MODELING

SIGGRAPH2007

Due to their repetitive structure, high-rise buildings and skyscrapers are an ideal target for procedural modeling. The complexity of these buildings stems from the volumetric design, while the façade itself is highly repetitive. In this example, we will illustrate the use of *recursive rules*, *attributed shapes* and transformation operations to simulate a typical setback design and repetitive volumetric structures. We chose to reconstruct of one of the of the two identical Petronas towers located in Kuala Lumpur, Malaysia. The Petronas towers have been designed by Cesar Pelli & Associates Architects and construction work has been completed in 1998. The building is very well suited for a CGA Shape encoding since its design is all over the place highly rule-based. For example, the main building footprint of both towers represents an old Islamic symbol consisting of two squares, where the second one is rotated 45 degrees around its central axis. In addition, cylinders are placed at the intersections of the two squares. Note that we focus in this example on the encoding of the distinctive main form of the towers only i.e. neither the famous sky-bridge nor the sidewing-like structures or the ground floor will be reconstructed.

Encoding of Buildings

High-rise Buildings with Setbacks

1: init ⇢ S(50,452,50) Subdiv(y,5r,4r,50){ element | setback | antenna }

2: setback : Scope.sy > 10 ⇢ S(Scope.dx-6,1r,Scope.sz-6)
Center Subdiv(y,1r,2r){ element | setback }

3: setback ⇢ S(Scope.dx-6,1r,Scope.sz-6) Center element

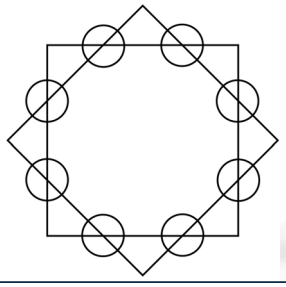
4: element ⇢
main R(0,45,0) Center main T(0.5r,0.5r,0) R(0,22.5,0) rotor(8)

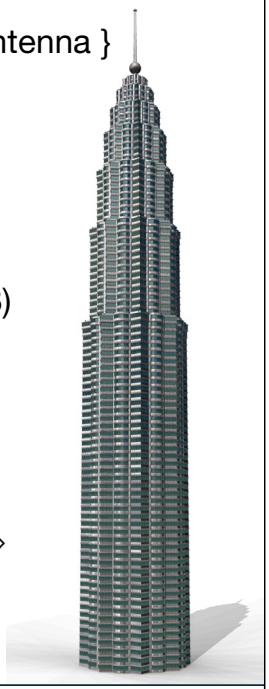
5: rotor(n) : n > 0 ⇢ R(0,45,0) rotor(n-1) T(0.75r,0,-0.1r)
S(0.2r,1r,0.2r) I(VerticalCylinder) oriel

6: rotor(n) ⇢ ε

7: antenna ⇢
S(8,1r,8) Center I(PetronasAntenna)

8: …

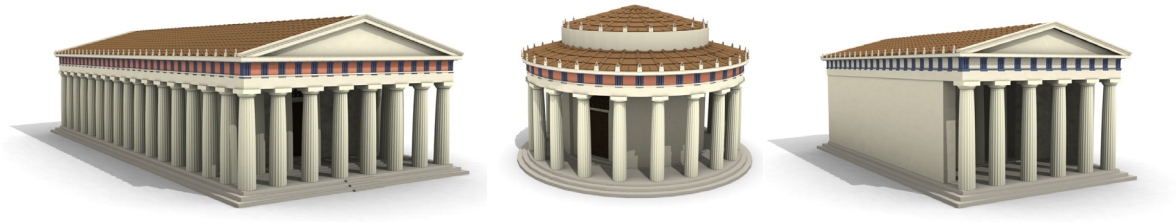




This rule set describes the shape rules for the volumetric design. The first rule scales the init shape (unit cube) to a box which is 50m wide and 452m high. The subdivision split initiates the tapering with the setbacks and the 50m high antenna shape. Rules 1 and 2 model the tapering with setbacks (which is typical for skyscrapers) by scaling down the width and depth by the constant value of 6. As long as the scope's height is bigger then 10m, a subdivision split inserts an element and recursively calls the setback rule. Rule 4 defines the actual shape of an element according to the described footprint: two shapes *main* are created (whereof the second one is rotated 45 degrees) and the generation of the cylindrical oriels is started by the non-terminal *rotor*. The rotor rule computes in recursively manner the placement of the eight cylindrical oriels on a element by inserting a cylinder geometry. The antenna geometry has been modeled in Maya and is placed with rule 7. In the follow up rules, the façade appearance is model via the shapes *main* (cube-shaped) and *oriel* (cylinder-shaped).

Parameterization of Buildings

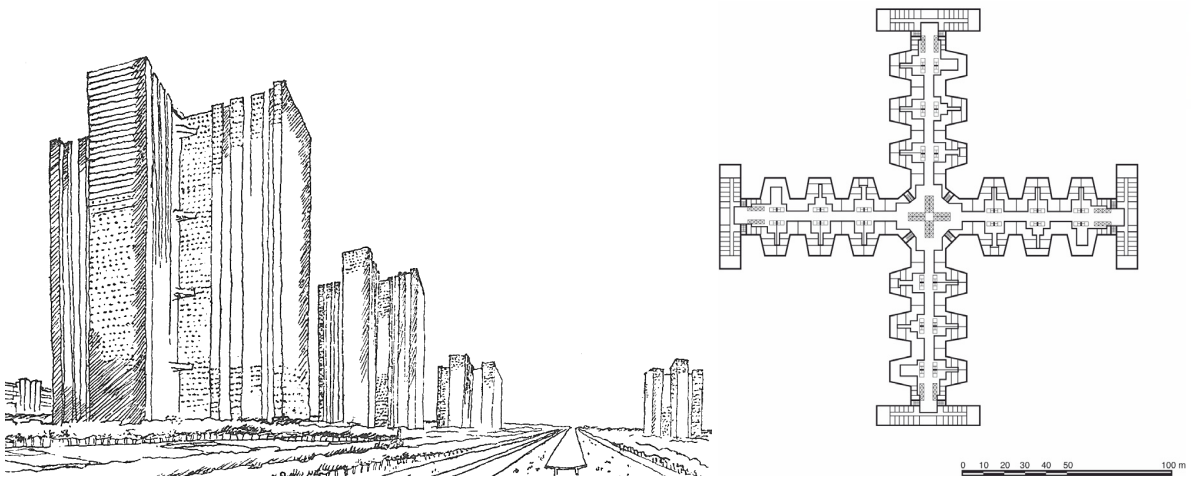
- Control parameters:
  The high-level interface to generate instances of a specific building design e.g. ancient temples



- But what are the parameters of a building design i.e. how can they be defined?

Parameterization of Buildings

Analyzing an Architectural Design



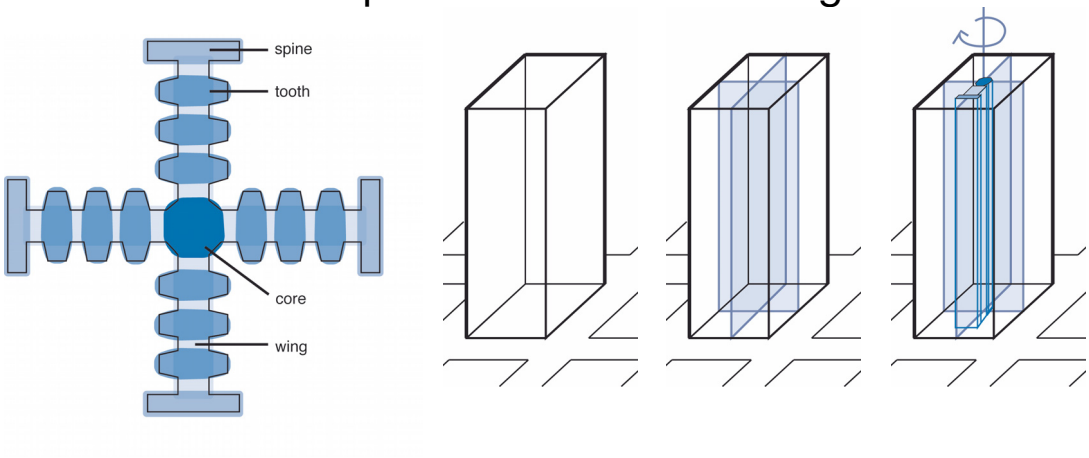
Example design: *"Cruciform Skyscraper" by Le Corbusier*

The Cruciform Skyscraper of Le Corbusier are taken as example. Le Corbusier designed, in great detail, several plans of different variations between 1920 and 1930. Most famous incarnations can be found in the master plans for his Contemporary City (1922) or the Plan Voisin (1925). The (back then) enormous sixty-storey cruciform skyscrapers are built on steel frames and encased in huge curtain walls of glass. They housed both offices and the flats of the most wealthy inhabitants of the cities. These skyscrapers were set within large, rectangular park-like green spaces.

Parameterization of Buildings

Encoding an Architectural Design

- Definition of components and assembling



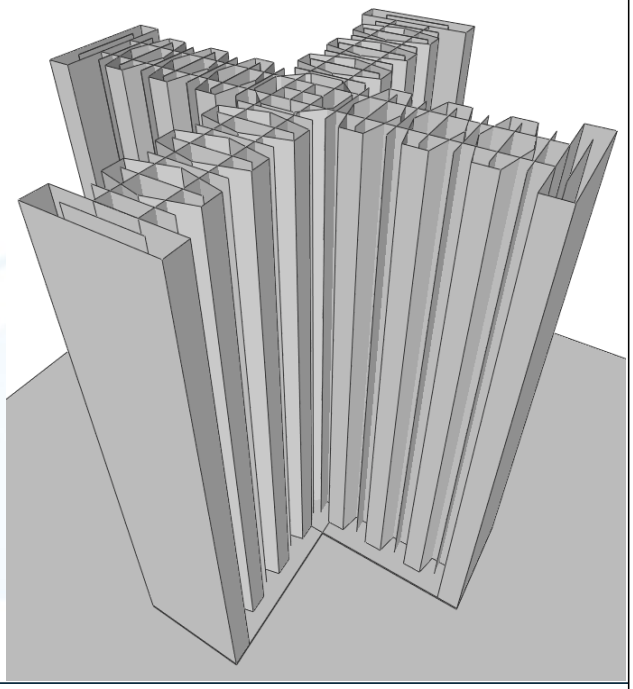
- Definition of parameters and spatial dependencies

Parameterization of Buildings

Control Parameters

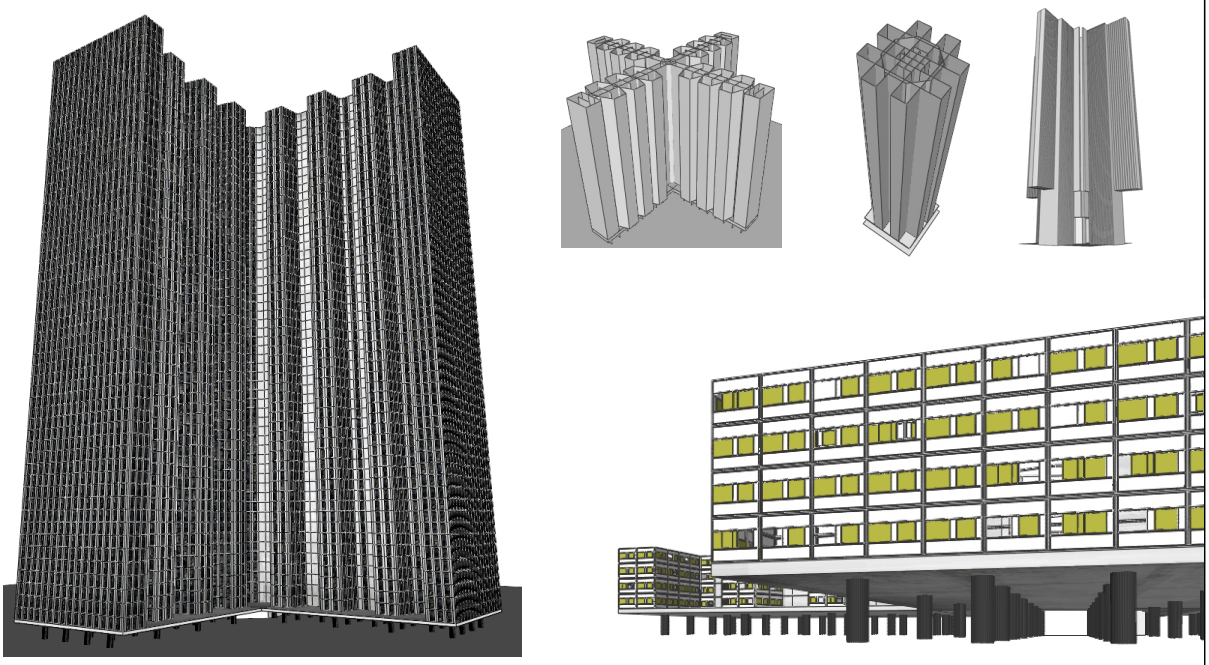
7 main parameters for volume:

define BUILDING_H	220
define GROUNDFLOOR_H	6
define PLATFORM_H	1
define WING_W	50
define WING_SMALL_W	16
define WING_TEETH_W	10
define WING_TEETH_DIST	12



Parameterization of Buildings

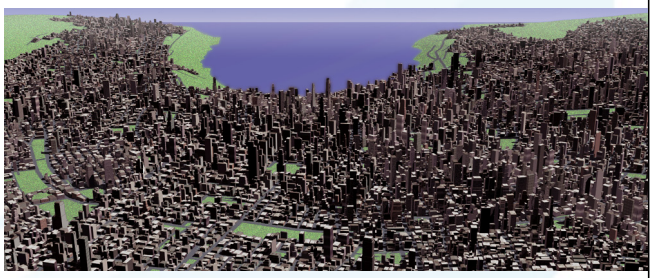
Procedural Corbusier Skyscraper



The appearance is controlled by parameters only.

Stochastic Modeling

- For massive models
- Stochastic rules
- Control grammar
  - random control parameters per building



- Random seed management needed e.g. per building

Problem:
How to control design emergence & retain plausibility?

Example of stochastic rule set without control grammar:

1: lot → S(1r,height,1r) building("DoubleWindow",rand(8)) : 0.8
→ S(1r,height,1r) building("SashWindow",3) : 0.2
2: building(windowcategory,windowtype) → Comp(f,sides){ facade(windowcategory,windowtype) }
3: facade(windowcategory,windowtype) → Repeat(y,floorheight){ floor(windowcategory,windowtype) }
4: floor(windowcategory,windowtype) → …
…
8: windowopening(windowcategory,windowtype) → [S(1r,1r,0.4) I(OpenCube) Comp(f,all){ wall }] T(0,0,0.4) window(windowcategory,windowtype)
9: window(windowcategory,windowtype) → S(1r,1r,0.1) I(windowcategory,windowtype,-1)
…

The problem is that such attributes have to be transferred top-down throughout the derivation in an inconvenient way. The same ruleset with a control grammar:

define windowcategory "DoubleWindow":0.8 or "SashWindow":0.2
define windowtype (windowcategory == "DoubleWindow") ? rand(8) : 3
1: lot → S(1r,height,1r) building
2: building → Comp(f,sides){ facade }
3: facade → Repeat(y,floorheight){ floor }
4: floor → …
…
8: windowopening → [S(1r,1r,0.4) I(OpenCube) Comp(f,all){ wall }] T(0,0,0.4) window
9: window → S(1r,1r,0.1) I(windowcategory,windowtype,-1)
…

Stochastic Modeling

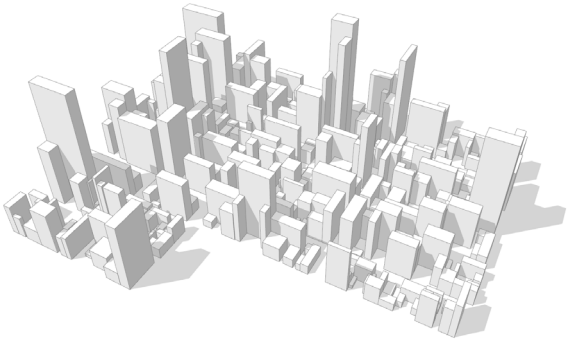
Encoding City-wide Variation

- Input data influences initial shape and its attributes
  - Shape of parcel / footprint ⇢ shape / size of a building
  - Population density ⇢ size / function of a building
  - Land use map ⇢ function of a building
  - Streets ⇢ orientation / function of a building
- Generally applicable rules
  - Size-independent rules with „boundary conditions“
  - Context-sensitive rules i.e. check scope dimensions and oclusion via rule condition
- User-guided rule set selection

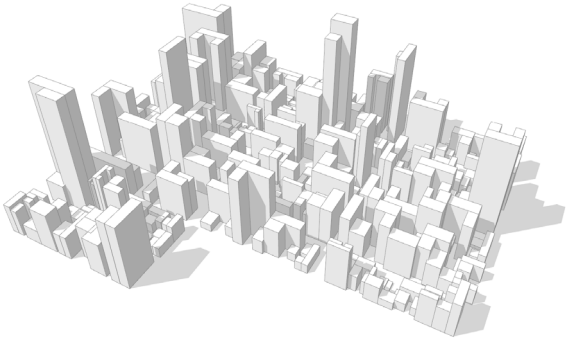
Stochastic Modeling

Design Control and Emergence

- Conditional rules to check context are crucial
- Too much randomness does not look convincing



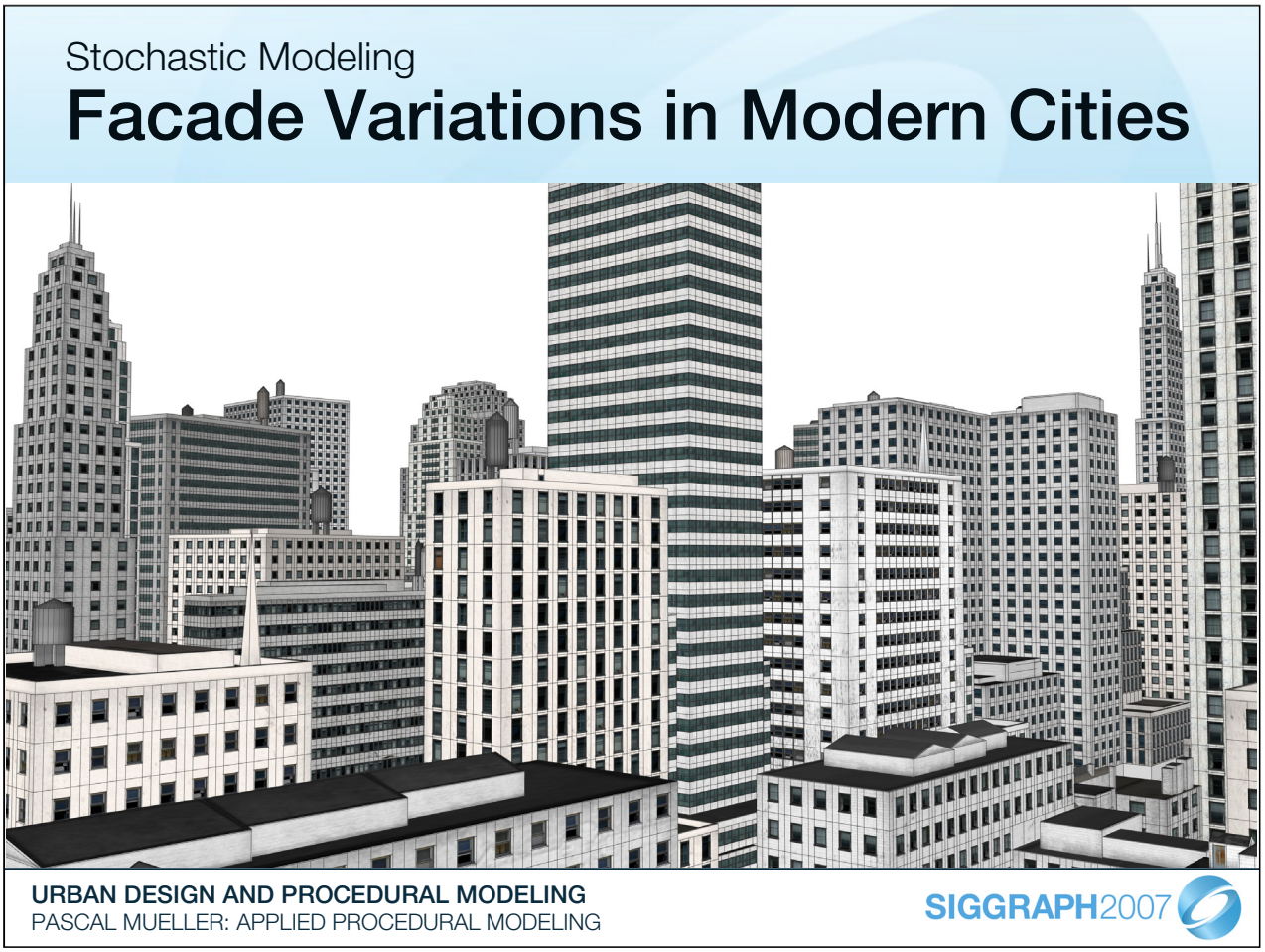
Fully random U-shapes



Controlled random U-shapes

The images show the result of a stochastic shape grammar ruleset:

- Starting with the building lot as initial shape
- Stochastic variations of mass models ('U' form as basis)
- In the left picture, heights and ratios of the sidewings are computed totally random
- In the right picture, the randomness has been slightly constrained (ruleset given in [Müller et. al 2006])



The figure shows facade variations created by a stochastic rule set consisting of:

- 6 different floor rules
- 6 different tile rules (each one with random ratios)
- 10 different windows

Stochastic Modeling

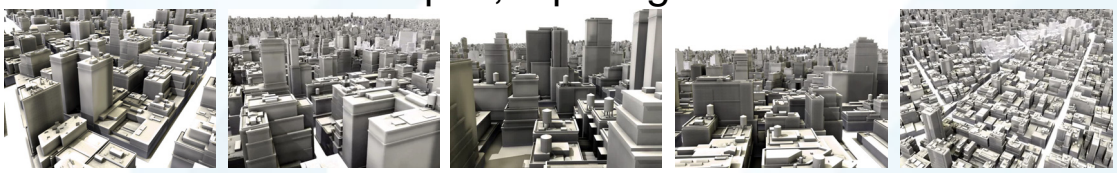
Garden Variations in Suburbia

A stochastic shape grammar rule set for landscape architecture has been developed to procedurally model the rule-based distribution and placement of vegetation and landscape objects in given urban environments, which are described through landscape patterns. Thus, the user can vegetate landscape and city parks, alleys, gardens, patios and even single buildings by applying context sensitive rules. Furthermore, arbitrary interactions between distinct instances of the vegetation and the urban environment can be encoded.

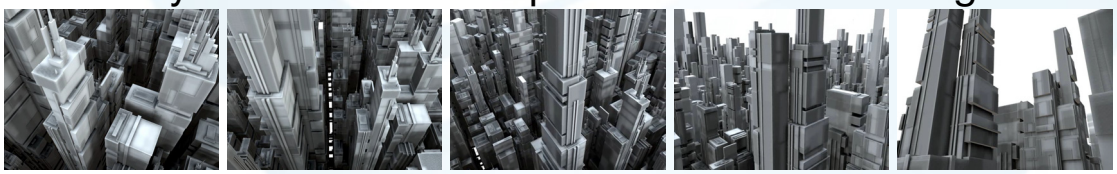
Stochastic Modeling

Miscellaneous City Examples

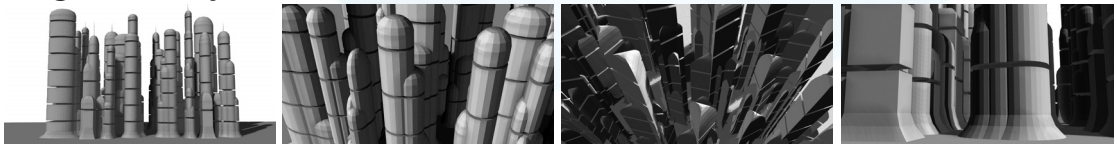
Variations in basic shapes, tapering and roofs



Sci-Fi city with variations in protrusions and bracing



Organic city with variations in valence and side elements



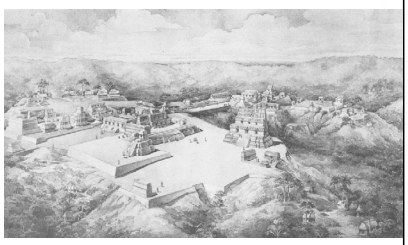
Procedural Reconstruction of Archaeological Sites

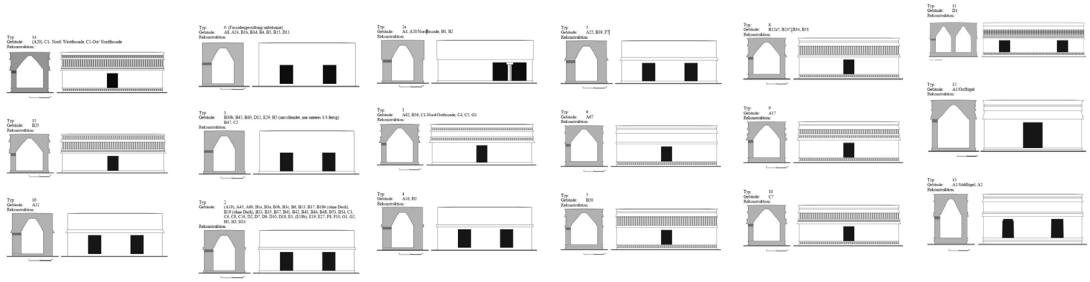
- Huge demand for 3D in cultural heritage
- Main focus still on the 3D reconstruction of *major* monuments, but *complete* site models are needed
- Archaeologist have accurate architectural knowledge no expert 3D knowledge (outsourcing too expensive)
- Procedural modeling serves as high-level 3D authoring tool and perfectly suited to reconstruct the ruined or unknown parts

Procedural Reconstruction of Archaeological Sites

Rebuilding an Ancient Mayan City

- Close collaboration with archaeologists
- Detailed GIS data available
- Site perfectly suited for CityEngine
  - One architectural style
  - Parameterizable building types





URBAN DESIGN AND PROCEDURAL MODELING
PASCAL MUELLER: APPLIED PROCEDURAL MODELING

SIGGRAPH2007

Close collaboration with archaeologists from Bonn to reconstruct the ancient city ‘Xkipché’ in Mexico:

- Detailed GIS data available
- Site perfectly suited for CityEngine
- One architectural style
- Parameterizable building types

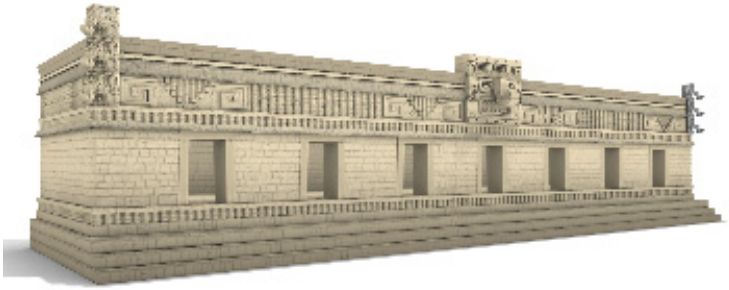
Further reading:

- P. Müller, T. Vereenooghe, P. Wonka, I. Paap and L. Van Gool. 2006. Procedural 3D Reconstruction of Puuc Buildings in Xkipché. Eurographics Symposium on Virtual Reality, Archaeology and Cultural Heritage (VAST), pages 139-146.

Procedural Reconstruction of Archaeological Sites

Encoding Mayan Architecture

- No formal design pattern exists for Puuc architecture
- We created one rule set for all buildings (with 32 control parameters per building)
- Also heavily ornamented buildings can be encoded



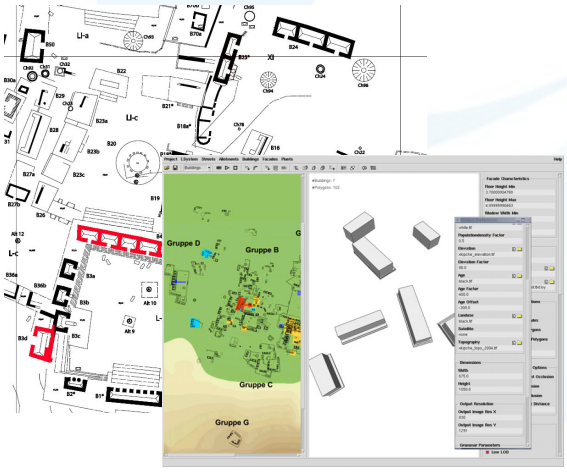
Problem: Almost no formal design pattern exists for Mayan architecture. After studying literature, we had to create one (based on similarities to classical architecture). We created one rule set for all buildings. Note that also heavily ornamented buildings can be encoded.

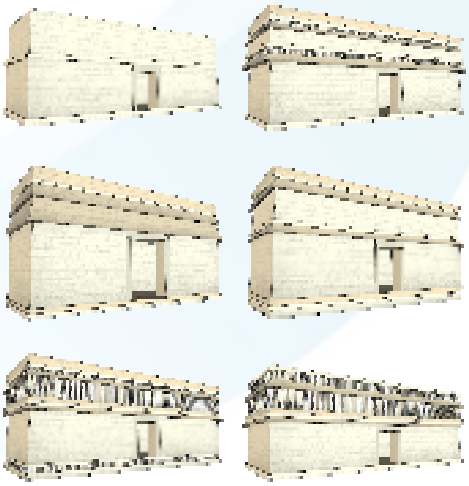
- we simplified some proportions and ended up with 32 control parameters per building
- Detailed plans and data available for some buildings

Procedural Reconstruction of Archaeological Sites

Accurate Mayan House Modeling

- Integration of building reconstruction parameters in the GIS data

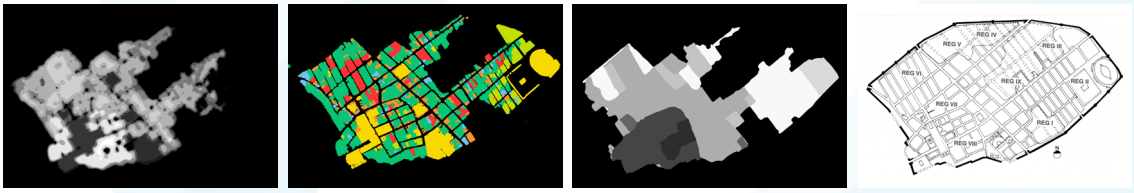




- Different to pompeii: here we want to be really accurate: the archaeologists look at any stone...
- high-level interface for archaeologists
- GIS data controls rule selection via control parameters
- Reconstruction without any CAD knowledge: just 32 parameters...

Procedural Reconstruction of Archaeological Sites

Procedural Pompeii



- Shapes: extrusions with different roof types / setbacks / peristyliums
- One or two-storey buildings (height between 5m and 9m)
- Lower parts of facades often painted in a redish color
- Remarkably large doors (~4 meter)
- Small and barred windows
- ...

URBAN DESIGN AND PROCEDURAL MODELING
PASCAL MUELLER: APPLIED PROCEDURAL MODELING

SIGGRAPH2007

Sociostatistical maps are used as input for the system (shown in the pictures on the top):

- Left: population density.
- Middle left: functional map (one tone for each zone, e.g. domestic zones are marked green).
- Middle right: age of urban area (darker areas mark older parts of the city).
- Right: overview map of ancient Pompeii for comparison.

Due to the good preservation of the city, the domestic architecture of Pompeii has been studied extensively. The Pompeian architecture spans several centuries and the buildings exemplify the evolution of domestic architecture: from the Italic model of the 4th to 3rd centuries BC to that of 1st century AD Imperial Rome. The form of the Pompeian townhouse is derived from Greek and Hellenistic designs and varies greatly in size and elaboration. The houses were built in a wide variety of shapes and sizes, but they usually displayed the preference for symmetry around an axis that characterizes most of Roman public architecture as well.

Based on such archaeological knowledge and real building footprints (=initial shape) and GIS data, ancient Pompeii was reconstructed with 190 manually written CGA shape rules. The whole model is a rulebased composition of 36 terminal objects (plus 4 tree types and the environment).

Procedural Reconstruction of Archaeological Sites

Procedural Pompeii: Renderings

Various views of the procedural Pompeii model

- Based on real building footprints, the city was generated with 190 manually written CGA shape rules.
- The whole model is a rule-based composition of 36 terminal objects (plus 4 tree types and the environment).

Applications in Architecture

- Master planning
  - For fast developing zones (mainly Middle East and China)

- Instant design variations on client-side
- Simulation & optimization (“form follows flow”)
  - Sustainable architecture, zero-energy cities, green spaces, …

The main goal of our research is to produce an architectural content modeling system for computer graphics. While our shape grammar tries to imitate the architectural construction process as well as possible, there are significant differences as we are more concerned with the visual appearance of the building rather then real-world construction issues, such as statics, zoning laws, functional floor layout, interior design and details like heating, electric lighting, or escape routes. Hence, to make the CityEngine more amenable for architectural applications, we will have to integrate these real-world issues by developing the corresponding grammar interfaces.

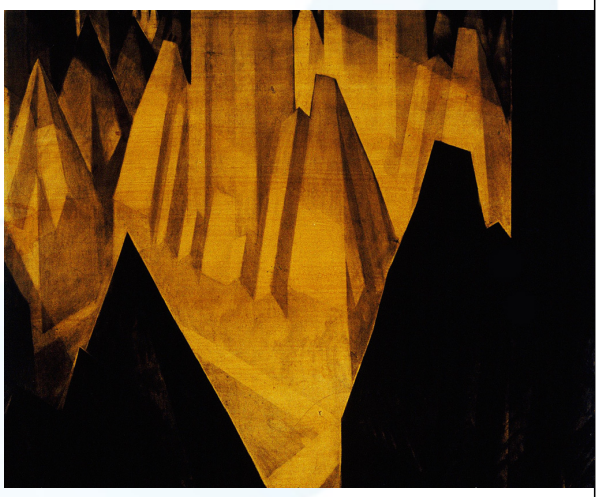
The figures show manually modeled visualizations of master plan projects:

- Left: Raha Beach, United Arab Emirates. http://alrahabeach.ae/eng/home/. Rendering by Last Pixel. http://www.lastpixel.com.au/showreel_stream_raha.html
- Right: BHAA, China. Rendering by Liu Yuetao. http://www.cgarchitect.com/3dawards/nominees/bhaa.htm

Applications in Architecture

Zoning and Building Envelopes

- Main constraints for the design of novel buildings nowadays.
- Specified by the legal authorities and can differentiate between cities, states or countries



Drawing by Hugh Ferriss

⇢ Encoding of zoning laws

The encoding of zoning laws is a very interesting application field for CGA Shape. In architecture, zoning laws specified by the legal authorities are, besides the financial directives of the realty developers, the main constraints for the design of novel buildings nowadays.

These laws can include the following specifications for example:

- Envelope: The envelope defines the volume wherein a building can be designed. It can be either directly given by the legal authorities or can be determined or modified by applying other zoning laws. Usually the envelope corresponds to the extruded ground area, but also further detailed envelope with setbacks at certain heights or setback angles are possible (see e.g. [Ferriss 1998] for spectacular illustrations of the envelope regulations in New York City).
- Floor area ratio (FAR): The main density indicator calculated as the ratio between the total floor area and the lot area.
- Mime: Allowable built area on an envelope surface in percent per face e.g. to force consistent front façade alignments of different buildings on a street.
- Minimum distance: A building has to keep a particular distance to others depending on its height and use (because of e.g. fire issues).
- Day lighting: Rules to limit the shadowing e.g. a high-rise >40m is not allowed to cast a shadow on adjacent housing for more than 2 hours.
- Etc..

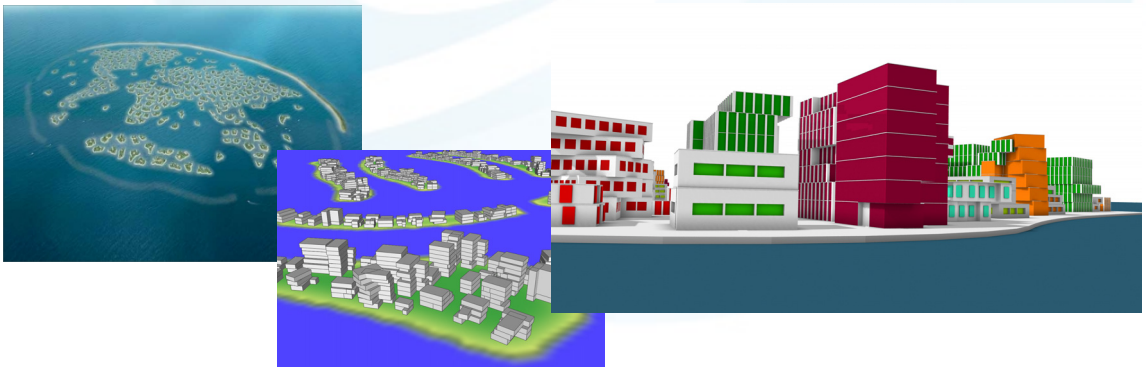
Further reading:

- Ferriss, H. 1998. The Metropolis of Tomorrow. Princeton Architectural Press.
- The Hugh Ferriss Architectural Drawings and Papers Collection. 2007. Columbia University. http://www.columbia.edu/cu/lweb/indiv/avery/da/ferriss.html

Applications in Architecture

Dubai World Islands

- Test-case is the „Australian continent“ of the upcoming Dubai World Islands
- Abstraction and management of architectural content
- Instant visualization of Urban Planning results for clients



The slide illustrates an example: Urban Planners have created building envelopes for the Dubai World Islands and the CityEngine has been used to draft buildings within these envelopes.

Link:

- http://www.theworld.ae/

Procedural Techniques in the Entertainment Industry

Reasons to use procedural techniques:

- Faster, better, cheaper, reusable, etc…
- Custom-designed implementation strategies possible (code, grammar, scripting, visual programming, etc…)

Arguing against procedural techniques:

- Often an over-blown solution that overshoots the mark
- *Artistic directability* not guaranteed anymore … really?

Artistic Directability

- Artists fear to lose control over design
  - Non-traditional design approach
  - Design granularity unclear
- *But procedural modeling does not affect the artistic directability. Far from it!*
  - Models can be edited in the traditional way afterwards
  - Visual tools to modify intermediate representations (shape-tree) can be easily realized and will be highly practical
- Limitation: converting visual edits back to rules
  - Open problem in design theory

Procedural Techniques in the Entertainment Industry

Quo Vadis Procedural Modeling?

- New user interaction techniques are emerging
  - Sketch-based, touch screens, gesture capture, …
  - Easily available and accessible for non-expert users
- *New design paradigms*
  - Computer more than an improved pen (CAD)
  - Computer more than calculating simulations & optimizations
- Procedural modeling will become more important
  - Encode design knowledge in/with the computer
  - But: massive technology evangelism needed!

Acknowledgments

- Simon Haegler, Andreas Ulmer, Peter Wonka
- Luc Van Gool, Markus Gross
- Jan Halatsch, Antje Kunze
- Tijl Vereenooghe, Iken Paap
- Pixar Animation Studios
- EC IST NoE EPOCH
- EC IST Project CyberWalk

Urban Modeling in Games

Andy Fuller, Electronic Arts

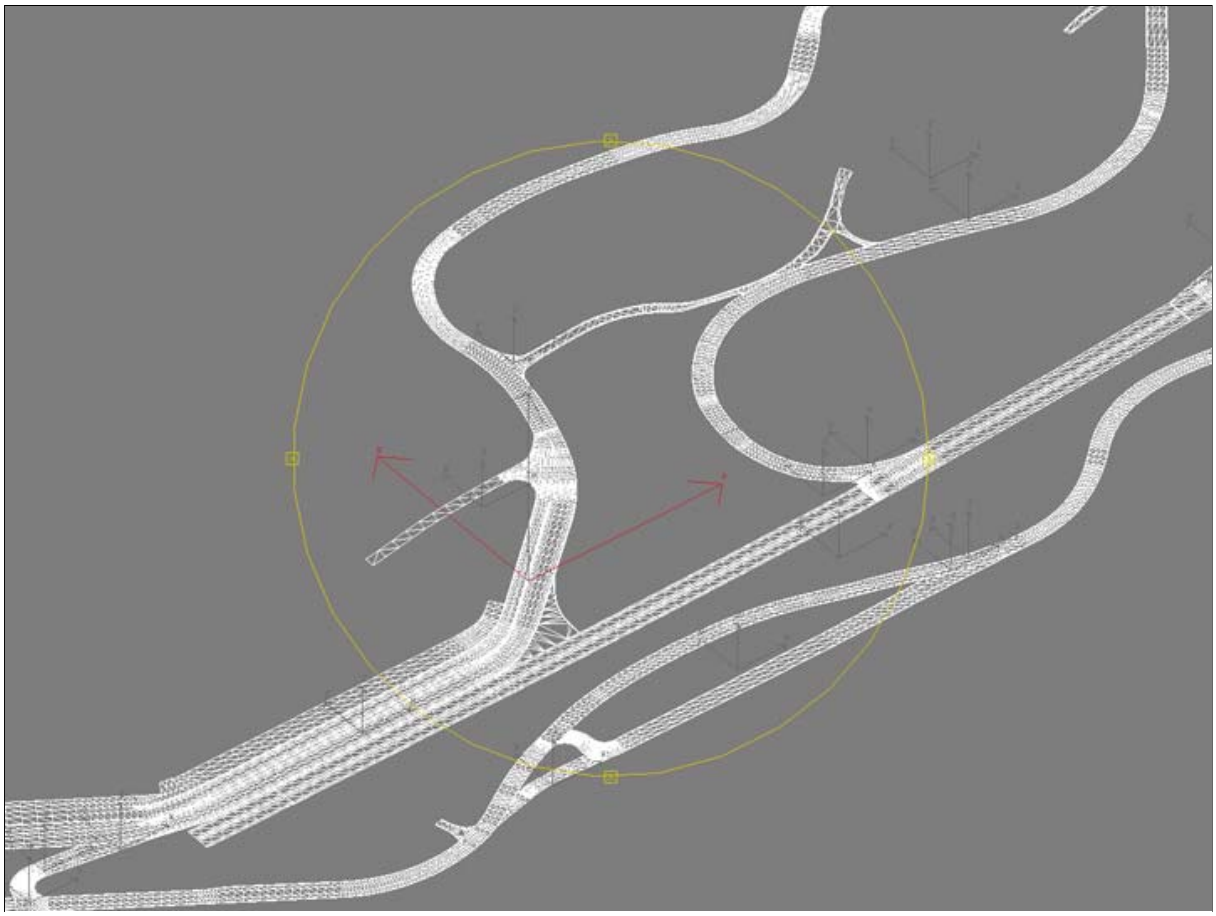
Opening movie - footage of real in game visuals from NFS Carbon.

Modeling Process Overview

- Step 1:
  Base Geometry
- Step 2:
  Color, Normal and
  Specular Mapping
- Step 3:
  Offset Normal Mapping
  and Visual Filters

Detail Level

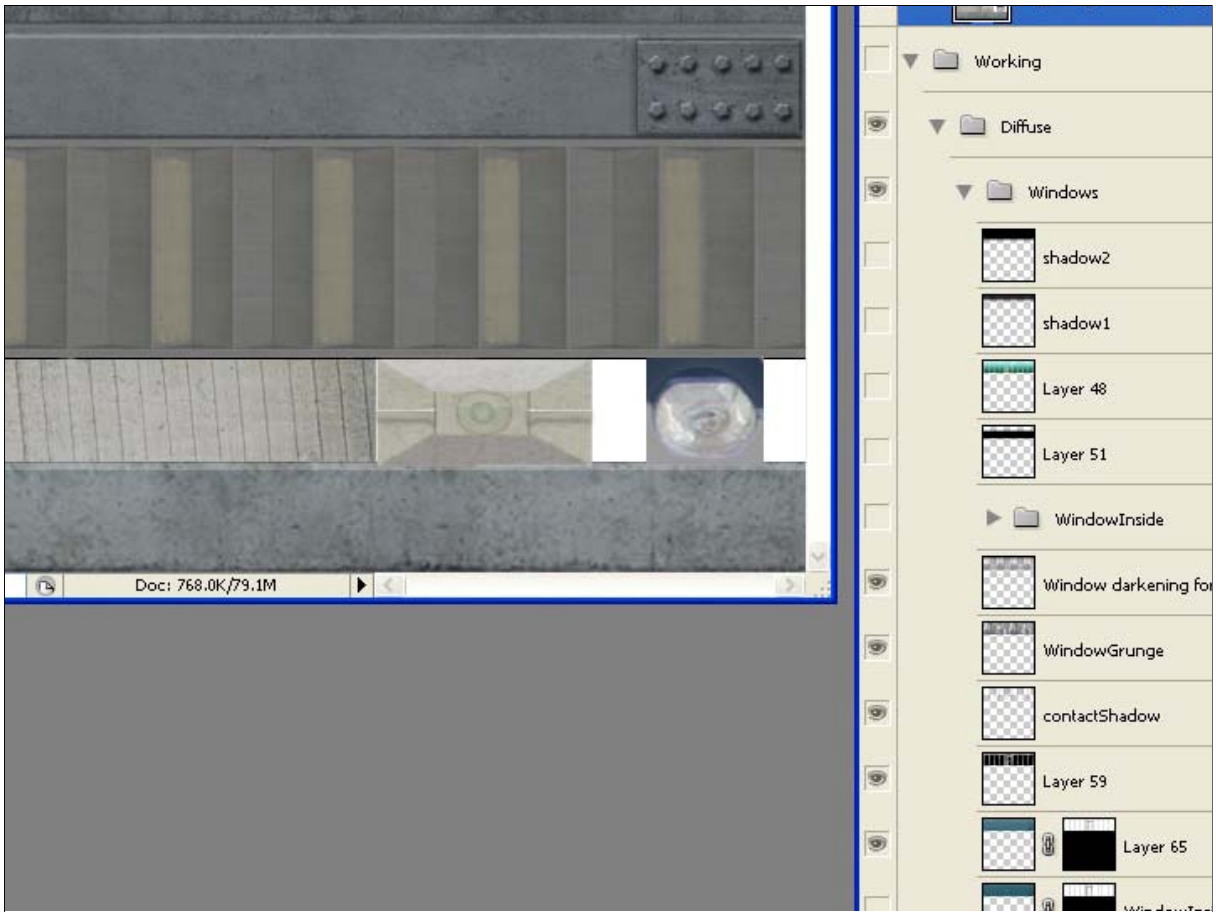
- City environments often get blurred-out to create a faster sense of speed
- When the car is actually stopped, you can see that the world has very little detail, and low-res textures.



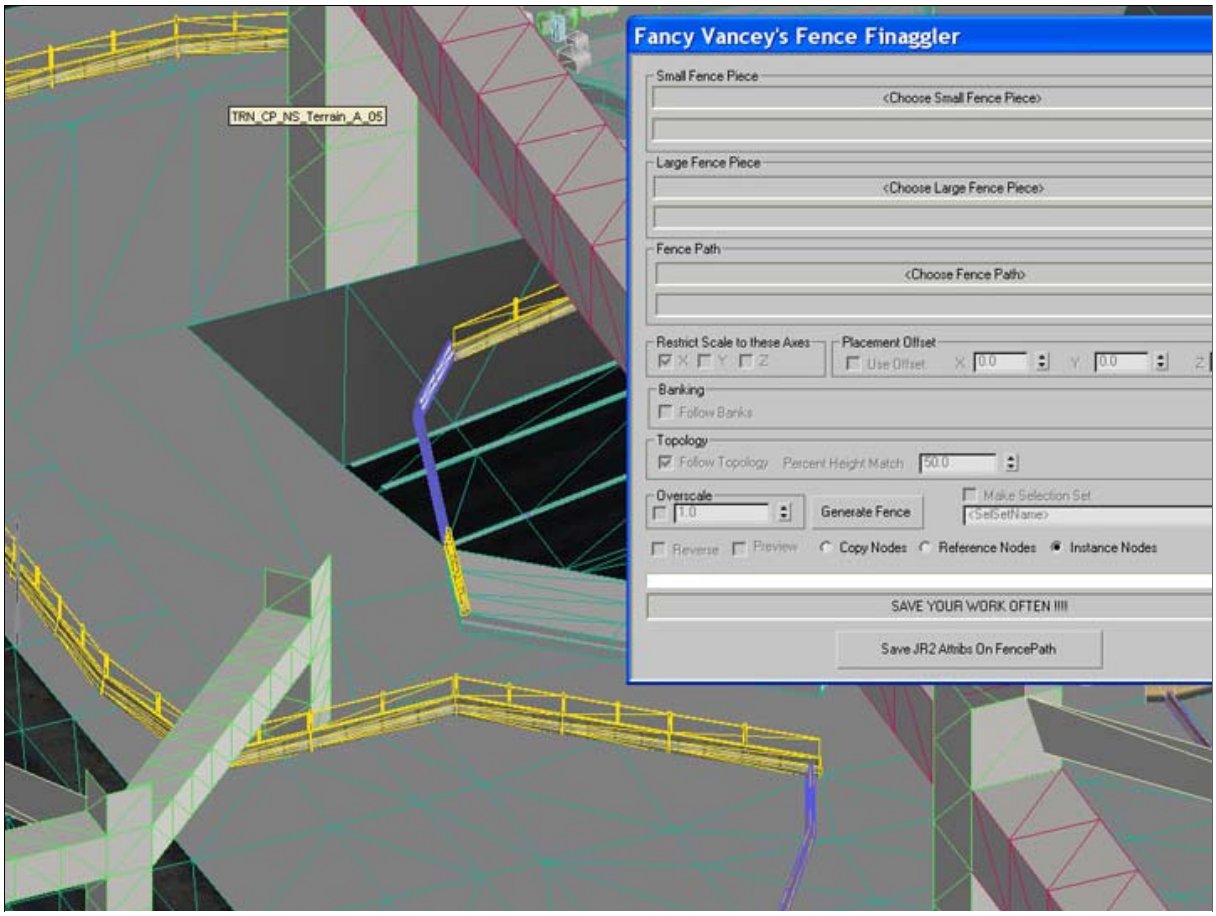
Illustrates the EA in-house Road Tool that generates mesh from a spline, applies UV mapping, adds density to curves and height elevation, banking to corners, and allows for different lane widths, etc.

Represents topography generated by EA's in-house Terrain Tool by applying it to sets of splines in 3D Studio Max. This tool is designed to use CAD data height information.

Procedural methods are also used to generate organic terrain.



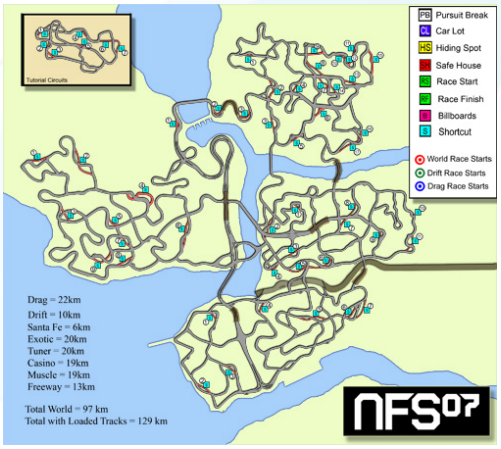
Photoshop scripts are used to create and organize layers that contain normal maps, spec maps, alpha, etc. The game engine then reads the individual layers and applies them to a regular diffuse map or shaders, as needed.



Fence Fenagler is an EA in-house tool used to lay down fencing, guard rails, barriers, etc. along a spline.

Putting the World Together

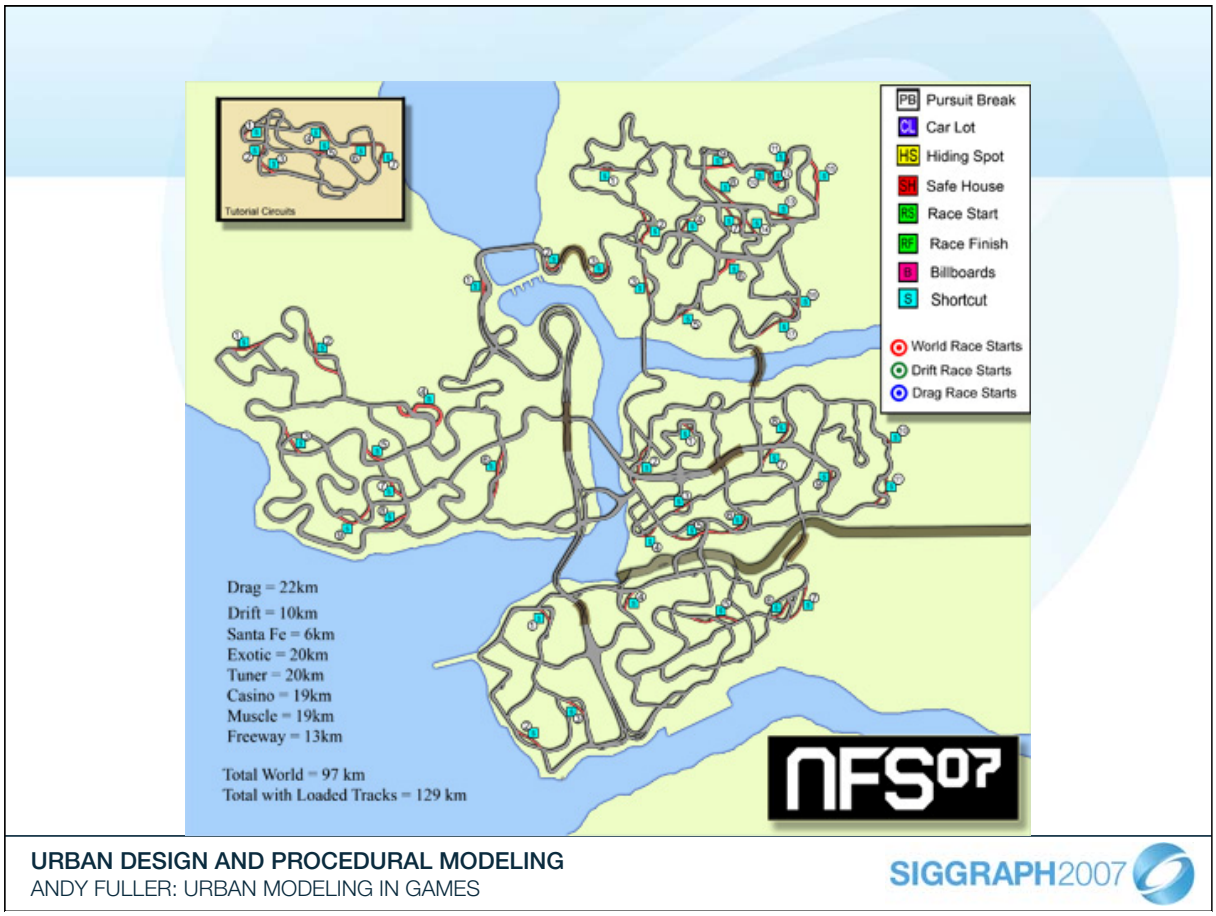
- Game-play is most important
- Editing of map regions to design the entire city



There was an immense amount of research and concept work required for constructing the city and the suburban streets that made up 'Need For Speed: Carbon'. There's a bit of Santa Fe, San Diego, Los Angeles, and San Francisco in the game and kept it within a sizable, but regional city. Most important was that the design of the city supports the gameplay.

After a bit of work, we had a good deal of concept work for each region to make them all feel a bit different and support the idea of the different car classes. The muscle car region was to be more working class. With that in mind, the team used an industrial feel with dock yards, factories, and train yards with a flatter terrain. The exotic region was more up-scale, Beverly Hills, Malibu Beach with the steep terrain and tight turns. The tuner region was more city center with the taller buildings and historic neighborhoods combining a blend of the rolling terrain and flat long roads. Once the player worked through those areas you move to the Casino region which favors no particular car class and the art direction was a blend of all regions to parallel that idea; glitzy, with some residential and industry.

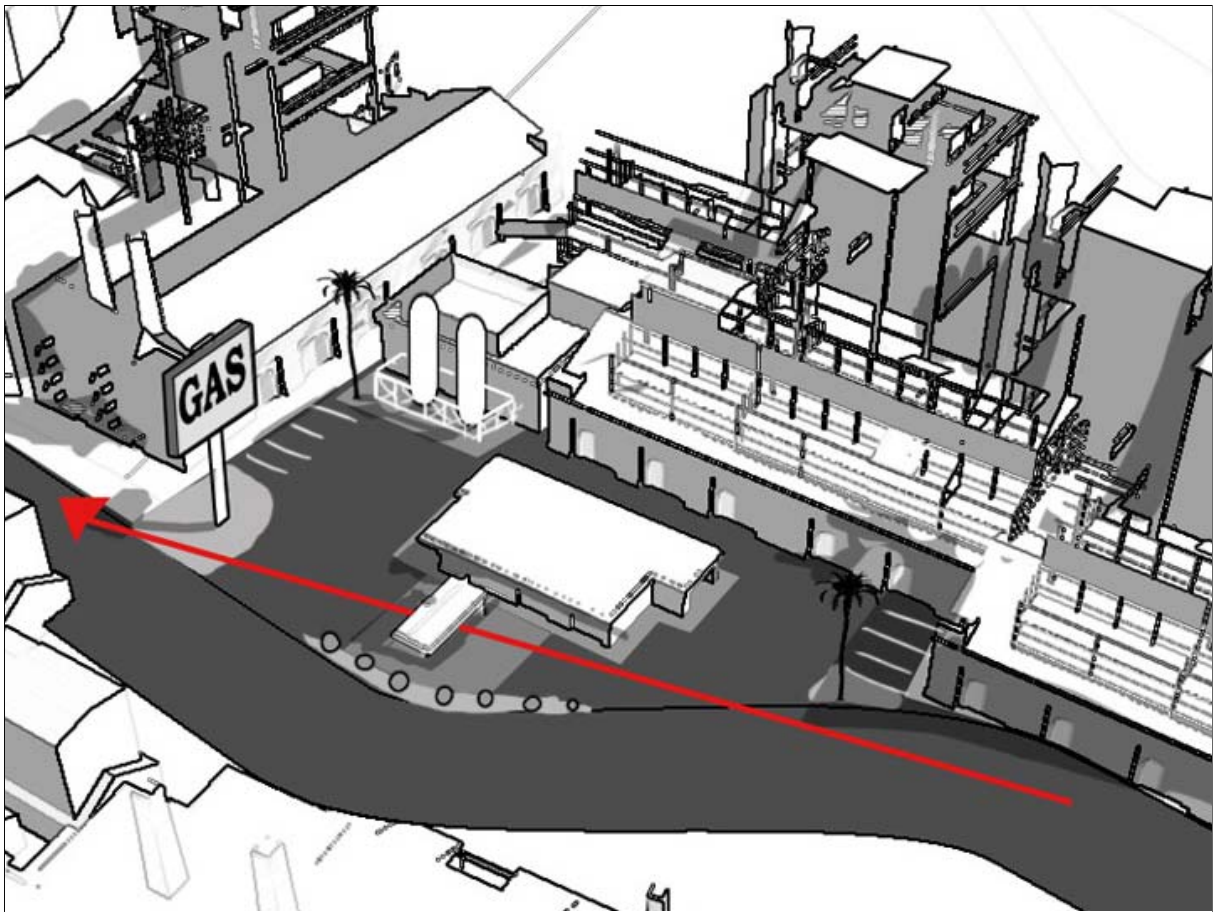
With all the platforms on which the game was going to be produced for, it was decided to keep the time of day at night to simplify multiple lighting scenarios. The art direction was to keep things as dark as possible and use lighting to help guide the player's eye down the tracks. Color was mainly used to highlight the buildings and environment to contrast the darker areas.



Most important when designing NFS world environments is the emphasis on great drives. We will actually lay down roads and tune them for the best driving experience, then build our cities and world environments around the great drives.

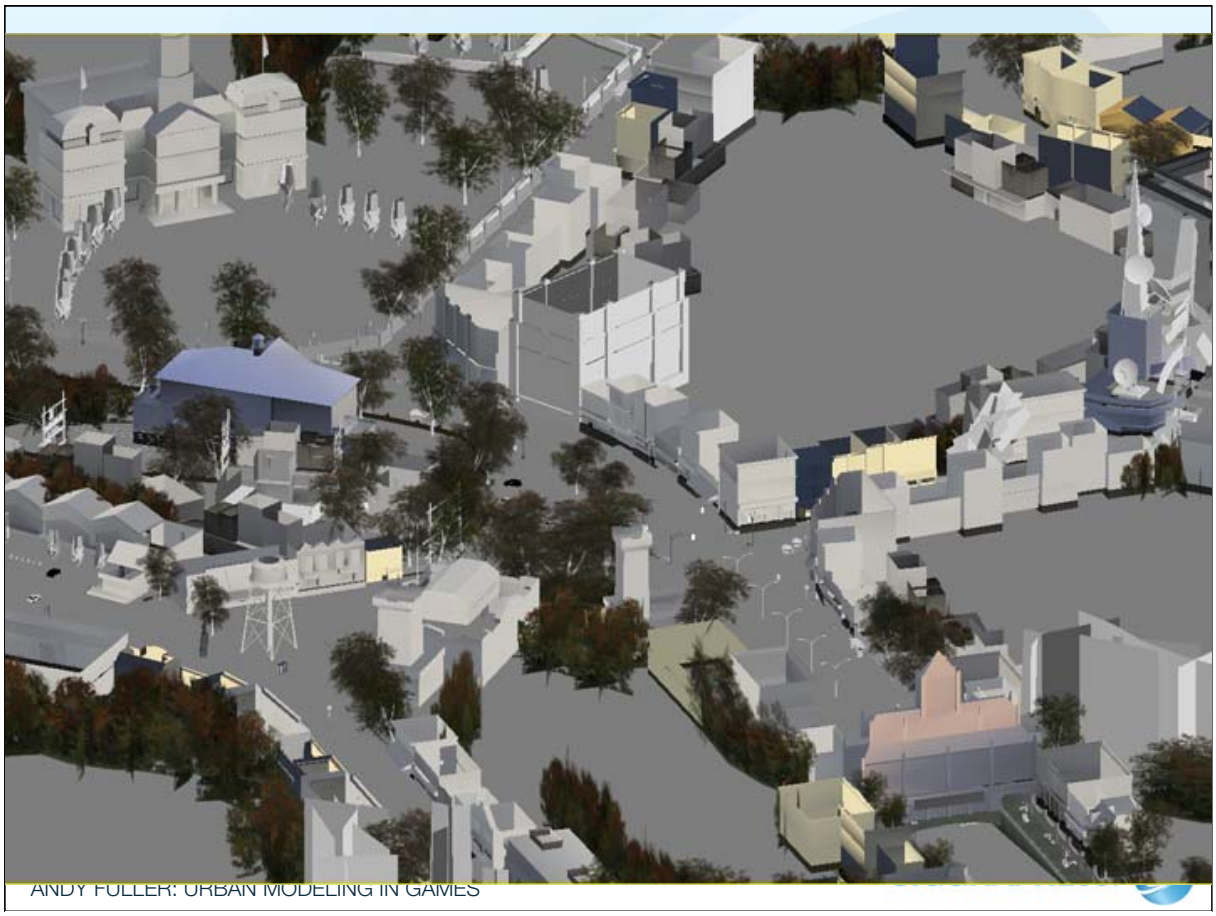
Sense of Speed is extremely important for game play and must be given top priority focus when creating a procedural city. After the drives are designed, NFS cities and world environments are created to effect an exaggerated sense of speed.

When NFS cities are designed, areas cater to a game-play moment, such as an escape route used to outrun a police car.

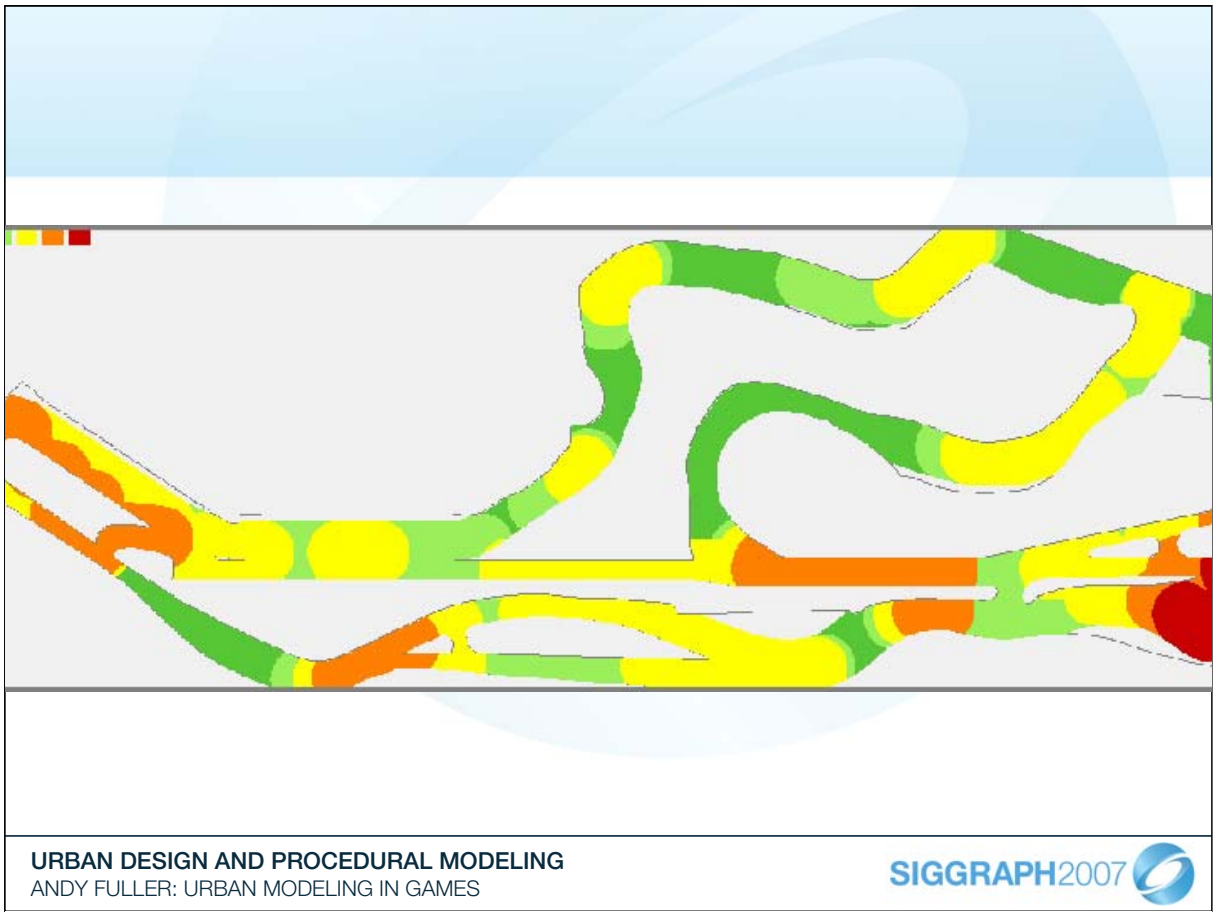


Another example of concept art produced in the design process. This piece illustrates a possible shortcut.

All NFS architecture and scenery objects are hand modeled from photo reference and x-reffed into a scenery file from libraries.

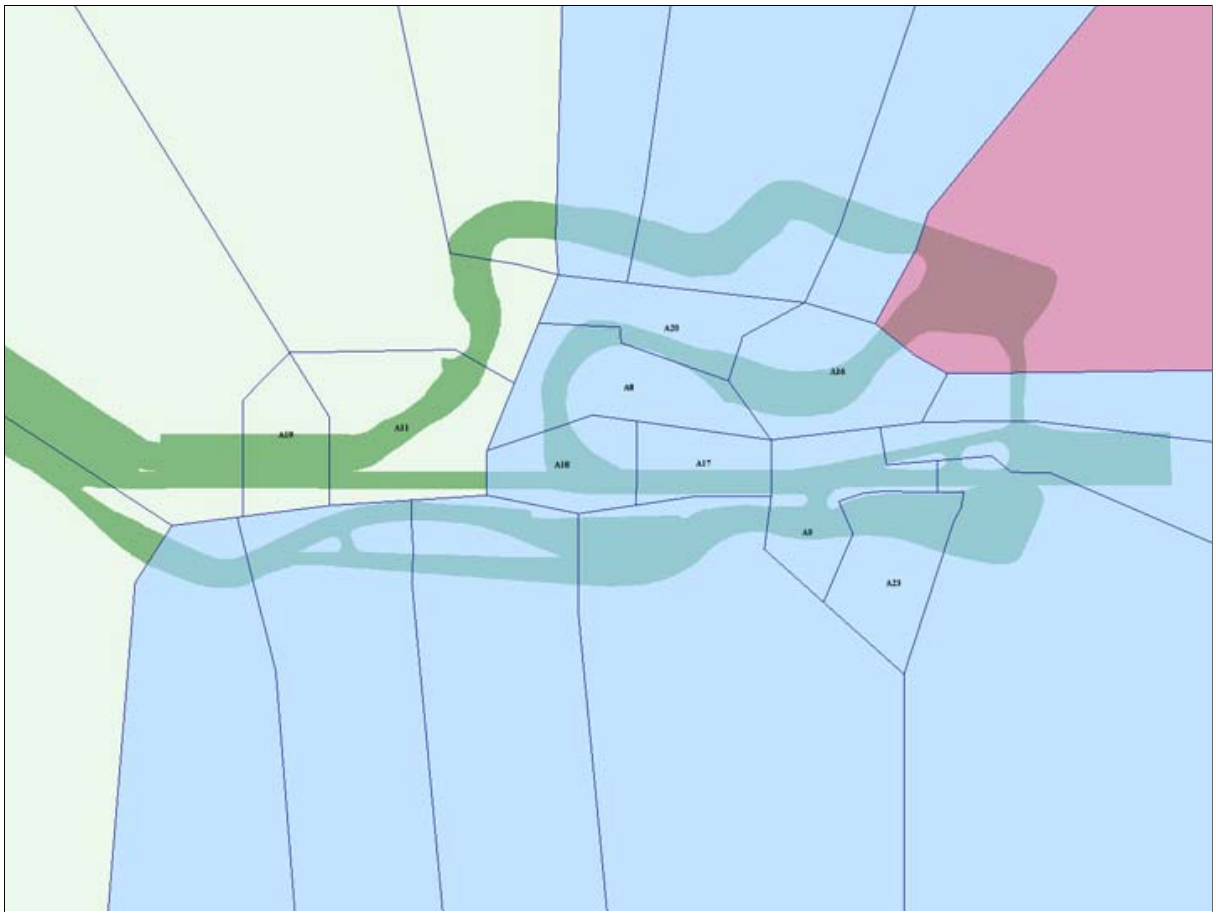


As you can see the NFS cities only outline the drives to create fake façade to optimize and meet memory budgets. Cities exist only to the reach of the car camera - an area limited to roads and some drivable terrain.



Frame Rate Mapper - Difficulties and Limitations.

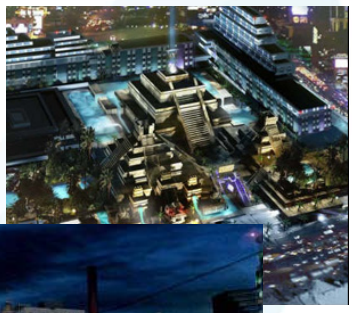
When designing a best drive, there will always be spots along the drive that have a long view-distance that creates difficulty when trying to render all the models and textures in the distance - often bringing frame rates to their knees. As illustrated above, the Frame Rate Mapper in-house tool helps indicate where the problem areas are (in red).

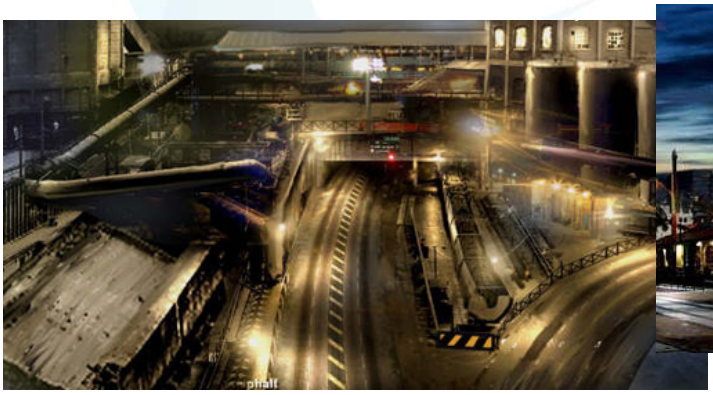


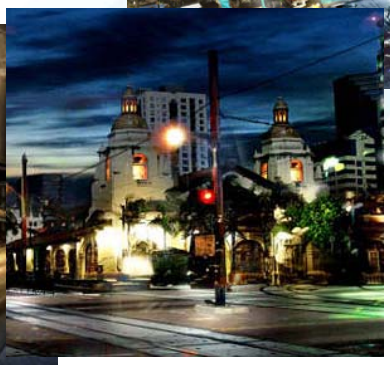
In order to render NFS city and world environments, it is necessary to stream in the assets. An in-house EA tool, SeeULater, generates section boundaries that determine which assets to load at a certain place on the track. Other assets are hidden until needed, and memory budgets met.

Lighting Concept

- Night shots only
- Lighting was used to support the game-play e.g. guiding the player

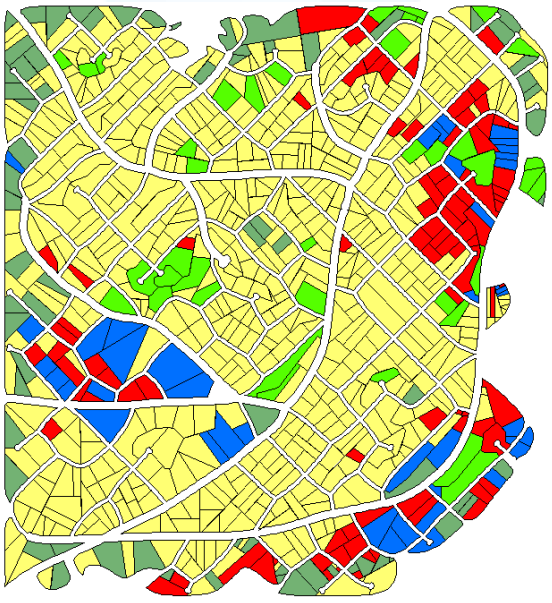




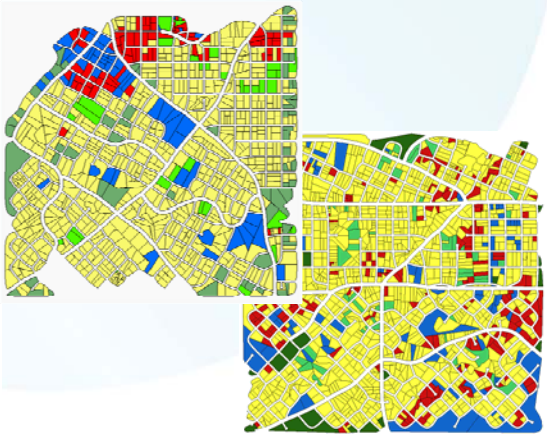


With all the platforms on which the game was going to be produced for, it was decided to keep the time of day at night to simplify multiple lighting scenarios. The art direction was to keep things as dark as possible and use lighting to help guide the player's eye down the tracks. Color was mainly used to highlight the buildings and environment to contrast the darker areas.

Real and Virtual Urban Design



Benjamin Watson
North Carolina State Univ.
http://cde.ncsu.edu/watson/



Designing virtual cities

- *Should look and feel real*, so need to understand
  - Physical patterns in real cities
  - Developmental regulation of real cities
  - Design of real cities
- *Should fit applied needs*, so tool
  - Need not model any particular city
  - Need only look “good enough:” shortcuts allowed
  - Need only model appearances, not processes
  - Be largely automated, yet editable

S. Eisner, A. Gallion & S. Eisner. 1993. *The Urban Pattern.* New York: Simon & Schuster.

Urban patterns: the core

- Higher density development in core
  - More commercial, government than elsewhere
- Concentration in the core
  - More in older cities
  - More in older nations
- Organization around core
  - Hub for transport – typical spoked pattern
  - Usually more zoning constraints

S. Eisner, A. Gallion & S. Eisner. 1993. *The Urban Pattern.* New York: Simon & Schuster.

Urban patterns: the periphery

- Outside the core
  - Less dense
  - Lower cost/acre
- Suburbs
  - Residences
  - Retail and restaurants
- Industry
  - Away from residential & high density

Urban patterns: transport hierarchy

- Roads organized into hierarchy
  - Lower levels: slower speeds, local access
  - Upper levels: higher speeds, distant access
- Some typical levels
  - Secondary: residential and workplace access
  - Primary: moving between neighborhoods
  - Highway: moving between cities
- Other components: rail, air, waterways

Urban challenges: smaller scale

- Tenements
  → Regulations for light, air, lot coverage
- Structural failure & fires
  → Construction codes
- Incompatible uses (e.g. homes & slaughterhouses)
  → Regulating proximity of use

Urban challenges: larger scale

- Speculation
  → Coupling of subdivision to development
- Congestion
  → Coupling of development to road construction
- Slums and blight
  → Eminent domain to assemble developable plots
  → Tax increment financing

Urban challenges: sprawl

- Today, development often focused outside core
  - In 70s, metro population up 10.2%, but only 0.2% in core
  - Vitality & residential density in core falls
- Multicore “edge” cities
  - Periphery contains alternate urban cores
- Trend stronger in younger nations & cities

→ Solutions very complex

Pctg stats: from US Statistical Abstracts

Urban planning

- "An effort to bring orderly development into urban communities and reduce social and economic conflicts that would endanger life and property" – *Eisner et al., The Urban Pattern*
- Implements solutions to above challenges
- Comprehensive ("master") plan
  - Describes developmental goals
  - Considers land use, transport, infrastructure…

AIA: The American Institute of Architects

AICP: American Institute of Certified Planners

APA: American Planners Association

www.planning .org: a valuable resource site run by the APA

Zoning

- "The regulation… of the height, bulk and use of buildings, the use of land, and the density of population". – *Edward Bassett, 1916.*
- "The legal regulation of the use of land…. for the protection of the public health, welfare, and safety." ". – *Eisner, Gallion & Eisner.*
- Distinct from planning
  - It is the instrument of planning, not the plan itself
  - Zoning map only shows the state of the plan

S. Eisner, A. Gallion & S. Eisner. 1993. *The Urban Pattern.* New York: Simon & Schuster.

Zoning history

- Regulations have existed since ancient times
  - Babylon, Greece, Rome, middle ages
- First U.S. zoning law: New York City, 1916
  - Edward Basset
  - Supported by Supreme Court in *Euclid v. Amber*:

"With the increase and concentration of population, problems have developed… which require additional restrictions in respect to use and occupation of private lands in communities."

Zoning law

- Zoning ordinance
  - Defines zoning regulations
- Zoning map
  - Defines districts where regulations apply
- Common district types
  - *Open:* undeveloped, agricultural…
  - *Residential:* one family, multiple family, high density…
  - *Commercial, industrial*…

Subdividing land

- Process
  - Developers must employ licensed surveyor/engineer
  - Detailed map prepared
  - Plan reviewed by planning director, planning commissioner, and perhaps city council
- Constraints
  - Lot sizes and dimensions set by zoning district
  - Developers must install streets, services, facilities
  - Approval considers city plan, demand on services

Urban design

- "The process of making better places for people than would otherwise be produced." – *Carmona et al., 2003*
- "The attempt to give form, in terms of both beauty and function, to selected urban areas or to whole cities. Urban design is concerned with the location, mass and design of various urban components and combines elements of urban planning, architecture and landscape architecture" – *Naphthali Knox, AICP, 1990*

M. Carmona, T. Heath, T. Oc & S Tiesdell. 2003. *Public Places – Urban Spaces: The Dimensions of Urban Design*. Oxford: Architectural Press.

Urban design elements: *Lynch*

- *Paths* for movement: roads, paths, rail…
- *Edges* between regions: shores, railroad cuts, walls…
- *Districts* or neighborhoods
- *Nodes* of travel: transit intersections, district cores
- *Landmarks:* mountains, spires, tall buildings…

K. Lynch. 1960. *The Image of the City.* Cambridge, MA: MIT Press.

Urban design elements: *Eisner et al.*

- *Movement, circulation and time:* getting around efficiently
- *Sound:* avoiding aural pollution
- *Unity:* respecting character of nearby areas
- *Light:* well-lit communities are safer
- *Color:* pleasing visual harmony vs. jarring contrast
- *Taste & smell:* good air quality

Urban design goals: *Lynch*

- *Vitality:* supports human functions and needs
- *Sense:* easily understood and navigated by users
- *Fit:* matches human behavior
- *Access:* easy to reach other places, services, persons
- *Control:* easy to create and manage access
- Should be *efficiently* achieved and *fairly* distributed

K. Lynch. 1981. *A Theory of Good City Form.* Cambridge, MA: MIT Press

Urban design goals: *Jacobs & Appleyard*

- *Liveability:* comfortable for all residents
- *Control:* residents feel a sense of ownership
- *Access:* to opportunities, new experiences, fun
- *Meaning:* layout, function & opportunities understood
- *Community:* easy to participate in public life
- *Self-reliance & Fairness:* urban efficiency and equity

A. Jacobs & D. Appleyard. 1987. Towards an urban design manifesto: a prologue. *J. American Planning Association*, *53*, 112-120.

Design movement: New Urbanism

- A proactive reaction to blight and sprawl
- Principles and advocacy
  - Neighborhoods should be diverse in use and application
  - Communities should be designed for foot traffic and transit, as well as the car
  - Cities should be shaped by their accessible public institutions
  - Frame urban places with architecture that celebrates local character

Congress for The New Urbanism

www.cnu.org

Design movement: Smart Growth

- Another proactive reaction, primarily to sprawl
- Some principles and advocacy
  - Boundaries that limit sprawling growth
  - Mixed-use development
  - Fiscal sharing among nearby localities
  - Preserving open space
  - Redeveloping inner core areas

www.smartgrowth.org

A. Downs. 2001. What does “smart growth” really mean? *Planning (APA)*, April.

Prior efforts in simulation

- Urban planning and design
  - *Problem:* predicting impact of developmental plans
  - *Characteristics:* input heavy, unsteerable, large scale
  - *Example:* UrbanSim
- Urban geography
  - *Problem:* studying urban processes
  - *Characteristics:* input heavier, unsteerable, larger scale
  - *Example:* SprawlSim

Planning:

Brail, R. and Klosterman, R. (eds.). 2001. *Planning Support Systems.* Redlands: ESRI Press.

Paul Waddell and Gudmundur F. Ulfarsson, Introduction to Urban Simulation: Design and Development of Operational Models. In Handbook in Transport, Volume 5: Transport Geography and Spatial Systems, Stopher, Button, Kingsley, Hensher eds. Pergamon Press, 2004, pages 203-236.

Paul Waddell, Alan Borning, Michael Noth, Nathan Freier, Michael Becke, and Gudmundur
Ulfarsson. Microsimulation of urban development and location choices: Design and implementation of UrbanSim. Networks and Spatial Economics, 3(1):43–67, 2003.

Dowling, R., R. Ireson, A. Skabardonis, D. Gillen, P. Stopher,.A. Horowitz, J. Bowman, E. Deakin,

and R. Dulla. (2000) Predicting Short-Term and Long-Term Air Quality Effects of Traffic-

Flow Improvement Projects. NCHRP 25-21. Transportation Research Board.

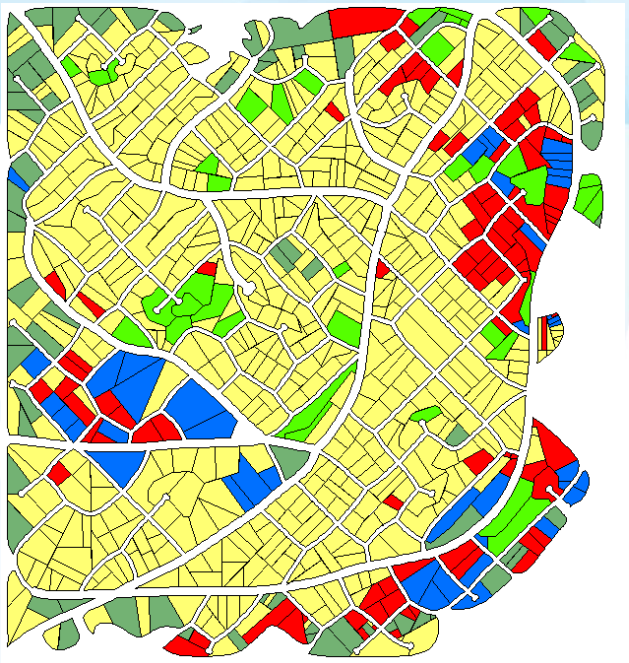
Our approach

- Agent-based simulation
  - Similar to flocking, particles
  - Robust to user intervention and interaction
- Model urban development
  - Environment: terrain and structures
  - Agents: structure developers

We are using agent-based simulation, the same technology used for flocks, and particles.

This approach has only very recently been taken up by the urban planning & geography communities.

Results:
land use

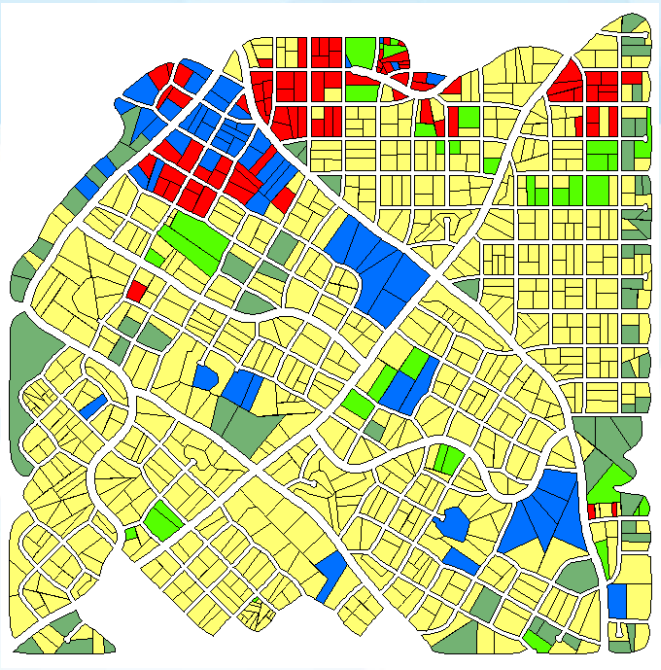


A 1.5x1.5 mile city:
residential, commercial, industrial, park, roads

Here is a 2D vectorized map of our output. Colors indicate land use, with dark green indicating undeveloped land.

Thick white lines are primary roads, thinner white lines secondary roads, and the thinnest gray lines are property boundaries.

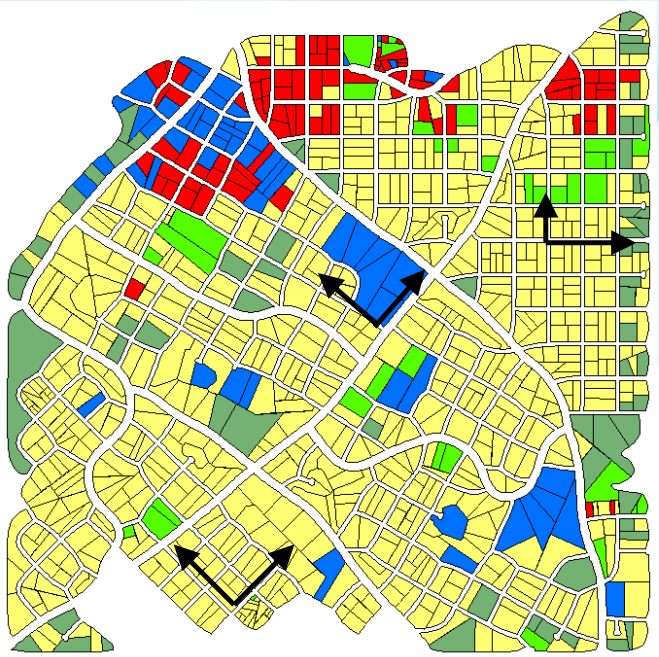
More results: *land use*



A 1.5x1.5 mile city:
residential, commercial, industrial, park, roads

A different section of town

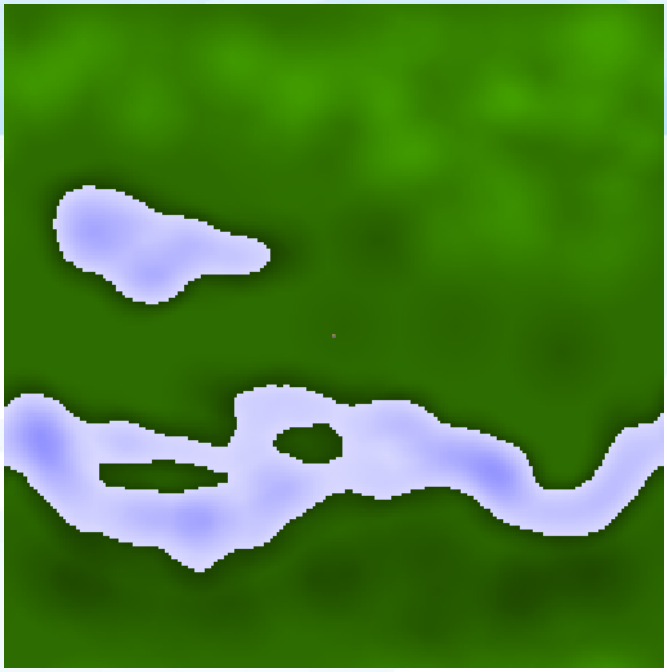
Control: *road gridding*



Artist paints a commercial zone, and various road layouts

Note the many road layouts that the artist has painted onto the terrain as input constraints. These can very in orientation, “tightness”, and spacing.

Input needed:

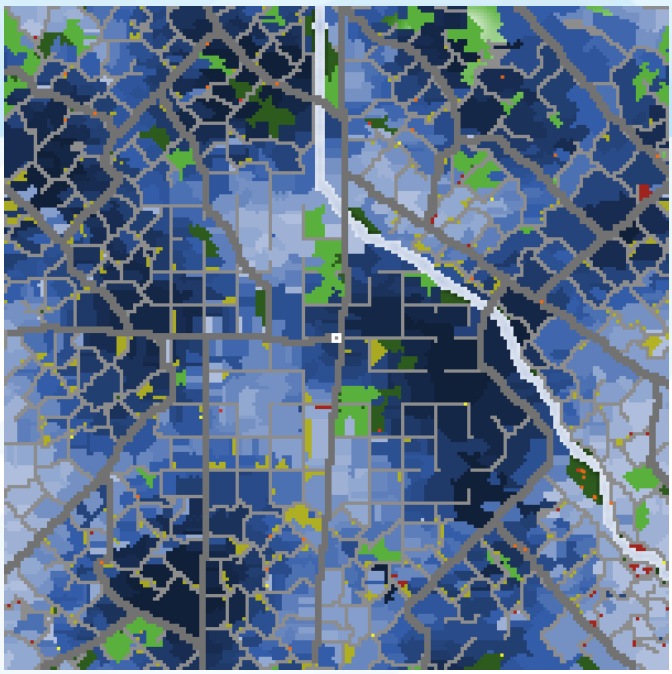


None.
Optional: terrain/roads/use, global or local constraints

Sims can run without *any* input, or can be guided by artist input constraints and parameters.

Here, the artist has painted a simple geography, with high regions a brighter green/lighter blue.

Control: *centrality*

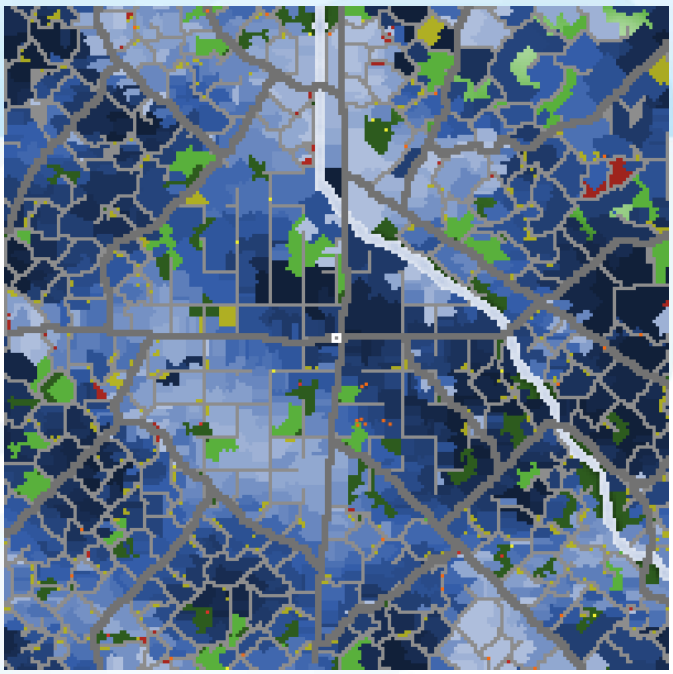


Weak core:
density smoothness constraint weak

Now we'll demonstrate artist control.

Artists can easily manipulate the centrality of urban use by changing one parameter. Here loose clustering...

Control: *centrality*

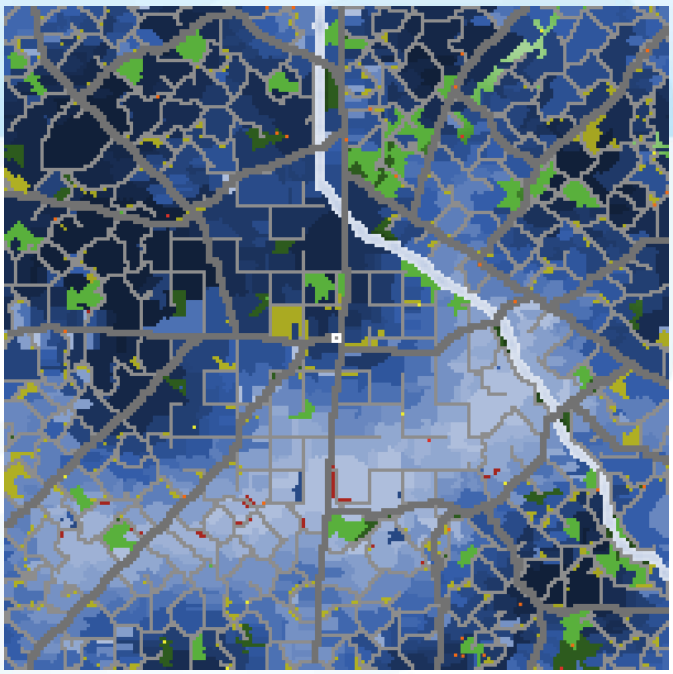


Stronger core:
stronger smoothness constraint

Tighter clustering…

Note that these sequences are generated with the same input terrain, including a few roads. So you can see the subtle variations and randomness introduced by repeated runs of the sum on the same input (with one changed parameter).

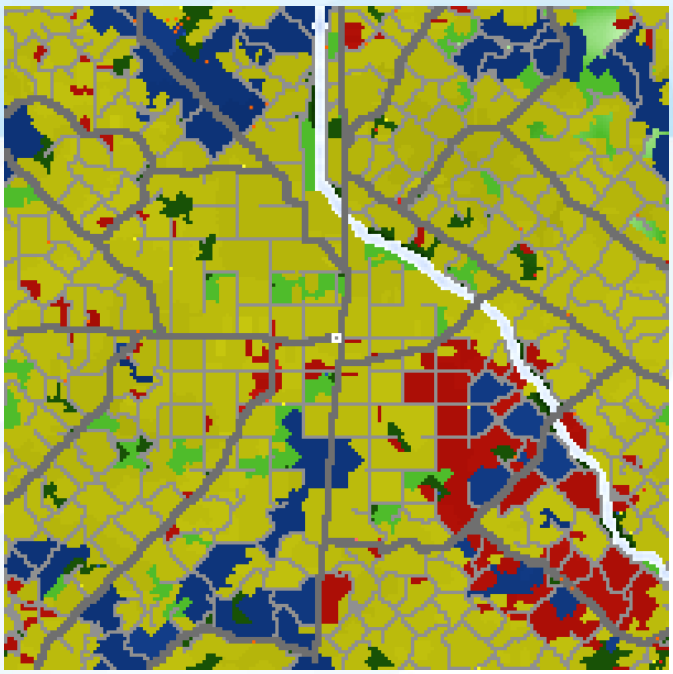
Control: *centrality*



Strong core:
smoothness constraint strong

Tightest clustering.

Control: *prevalence of each land use*

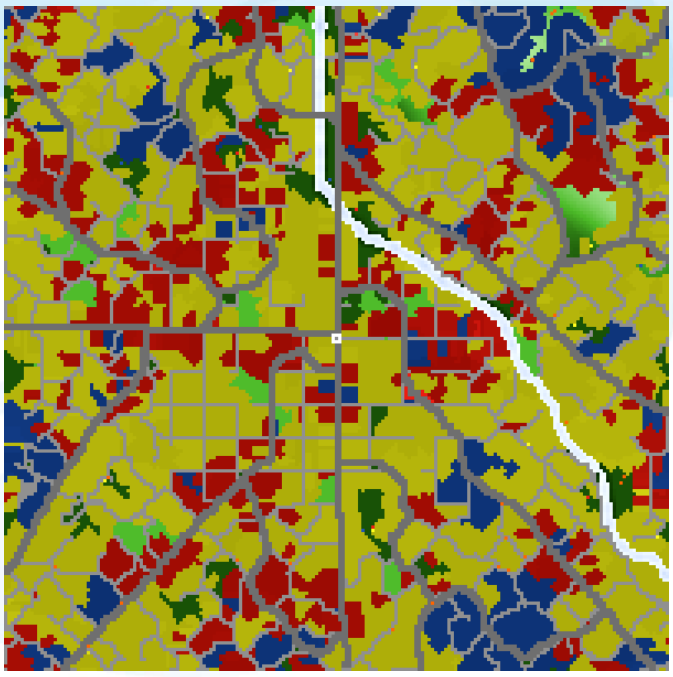


Little commercial use:
land cover parameter smaller

Similarly, artists can control the amount of density and land coverage dedicated to each use by manipulating two parameters.

Here, very little commercial use.

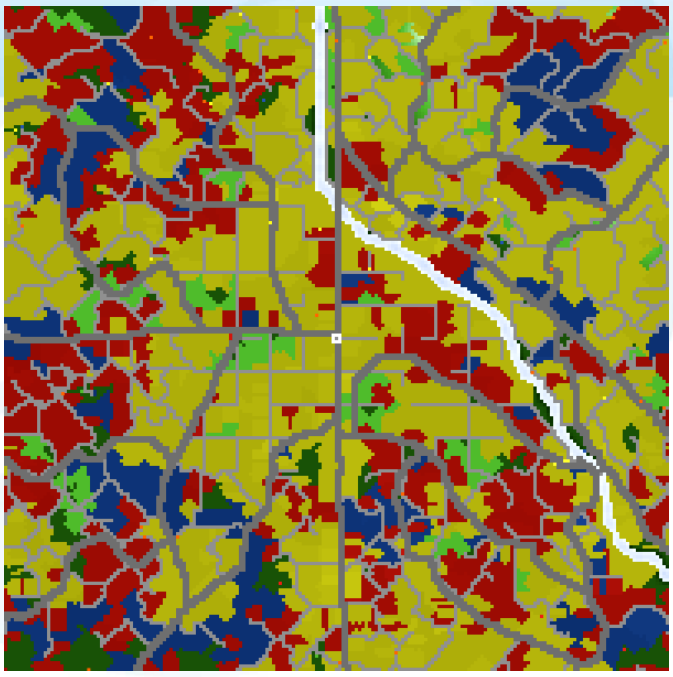
Control: *prevalence of each land use*



More commercial use

More commercial use…

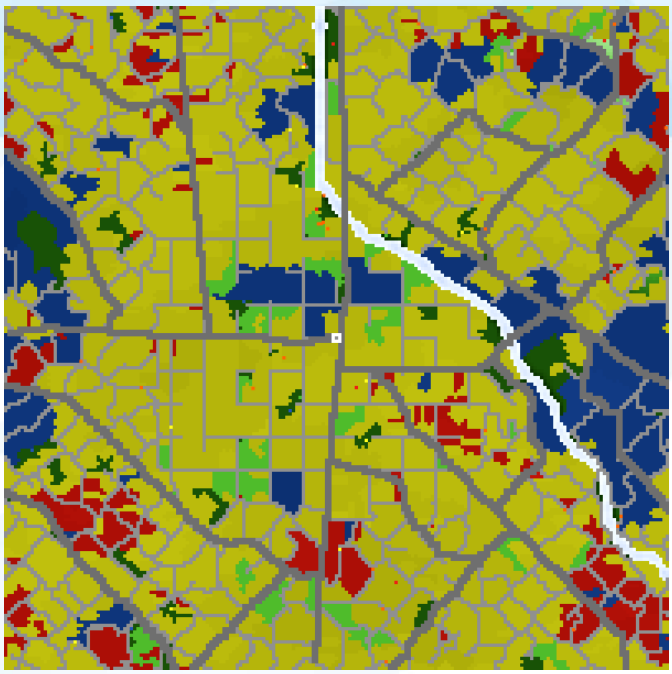
Control: *prevalence of each land use*



Lots of commercial use

Still more.

Control: *clustering*



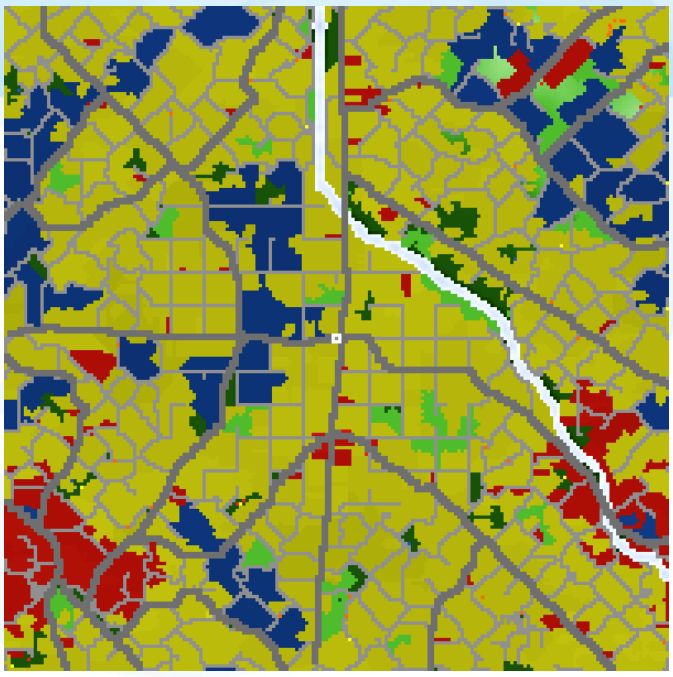
Thinly distributed commercial use:
importance of proximity weak

Land uses cluster together, with varying degrees of tightness.

Artists can control tightness of clustering with a per usage type parameter.

Here commercial clustering is weak.

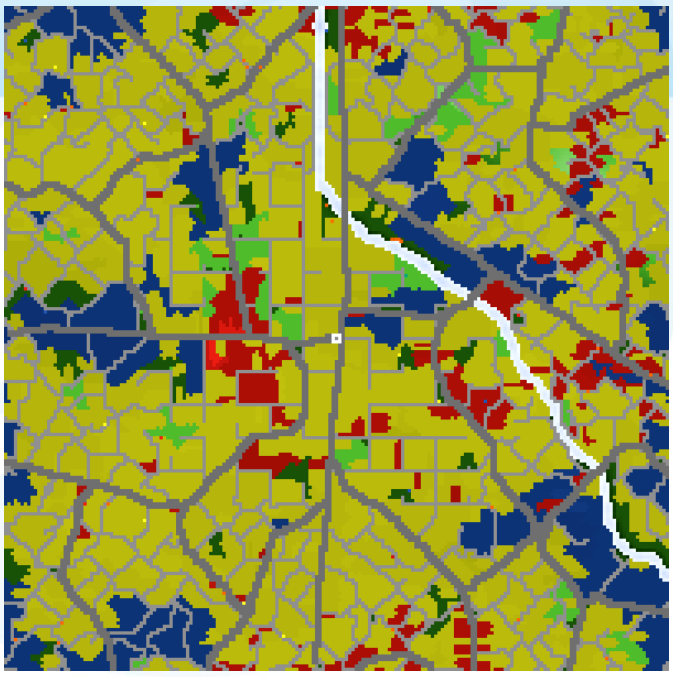
Control: *clustering*



More commercial clustering

Now it’s stronger.

Control: *clustering*

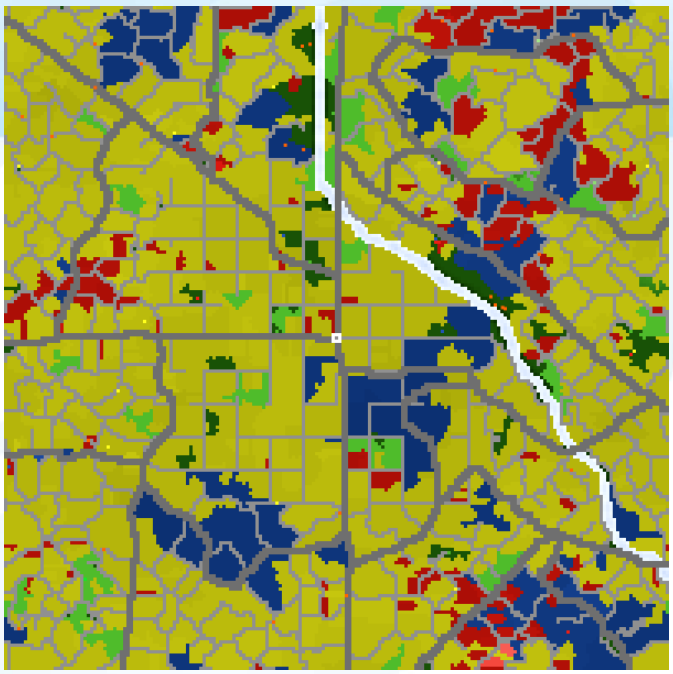


Lots of commercial clustering

And still stronger.

Note the especially dense (bright red) development in the center, and that uses have largely abandoned the left side of the city.

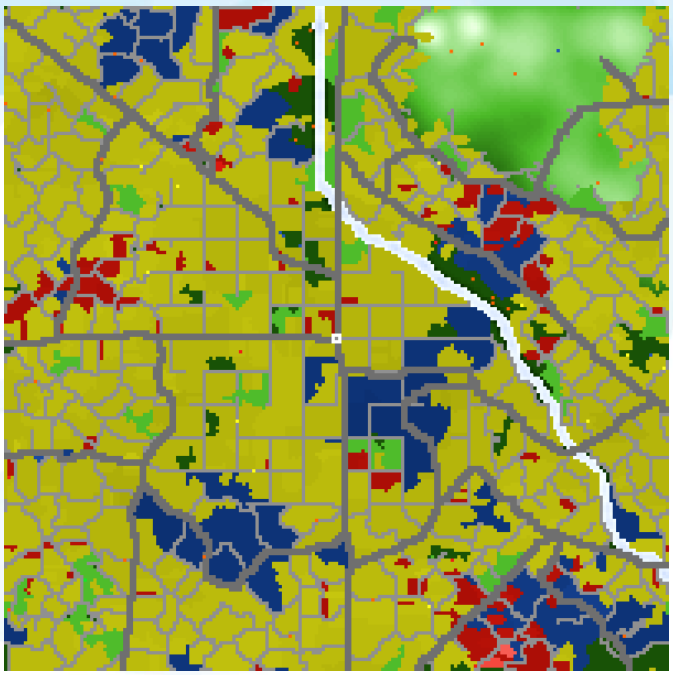
Control: *interruptability*



State of city:
artist unhappy with commercial cluster top right

We mentioned interruptiblity. Artists can change and wipe out portions of the city as they like, then watch how the city adapts.

Control: *interruptability*

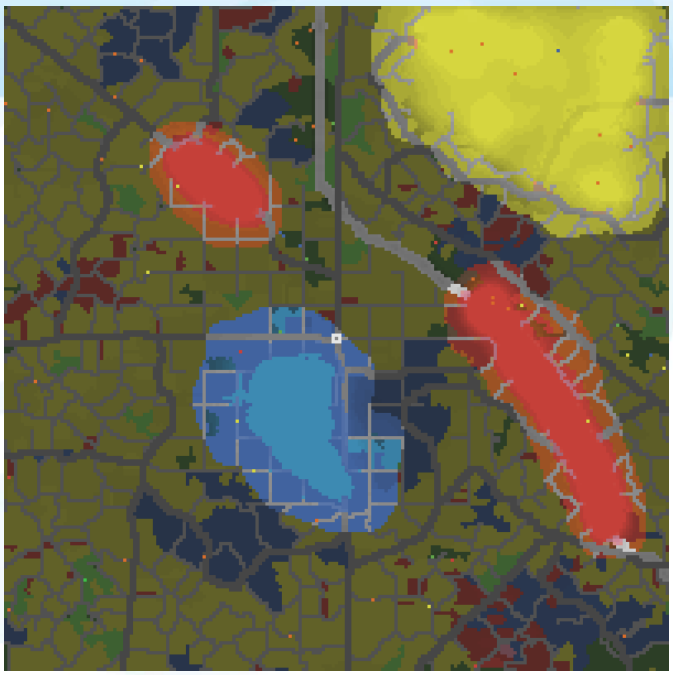


Erase the cluster…

Here the artist levels the commercial concentration top right.

The artist could then simply press “go”, and watch what ensues.

Control: *interruptability*



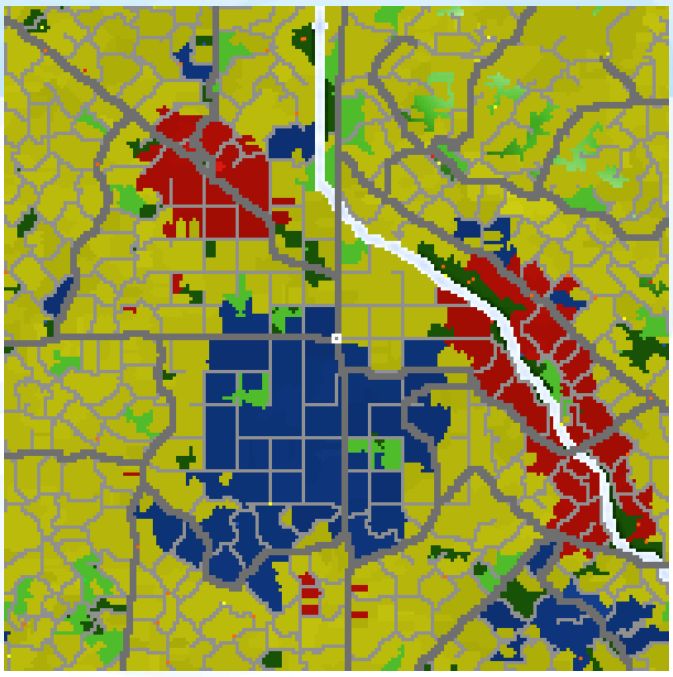
Add “honey”, incentives for development.

But the artist can provide incentives for different types of development, which we call “honey”. Here the artist paints some honey onto the city.

Color indicates the sorts of development attracted by the honey.

We’ve made the honey particularly strong here, for demonstration. But it can easily be made less sweet.

Control: *interruptability*



Result: honey “sweetness” controllable

And appropriate development ensues. The artist does not have to carve out new roads and property boundaries by herself.

Developer agents

- Build structures in environment
- Property
  - res, ind, com and park types
- Roads
  - Primary, access extenders,
  - access connectors

Agents build things.

There are seven types of developers: residential, industrial, commercial and parks, as well as primary road, access road extenders, and access road connectors.

Property developers

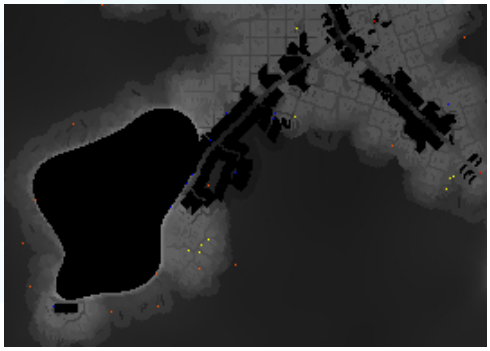
- During each sim tick:
- Move toward valuable land
  - Hill climbing w/ random drops
- Build on most valuable land seen recently
  - Create parcel or increase density
  - Keep development if profitable

This is the basic behavior of the four types of property developers.

As noted earlier, property developers can convert uses if it's profitable. We also don't permit industrial to be converted directly to residential, and vice versa.

Residential value

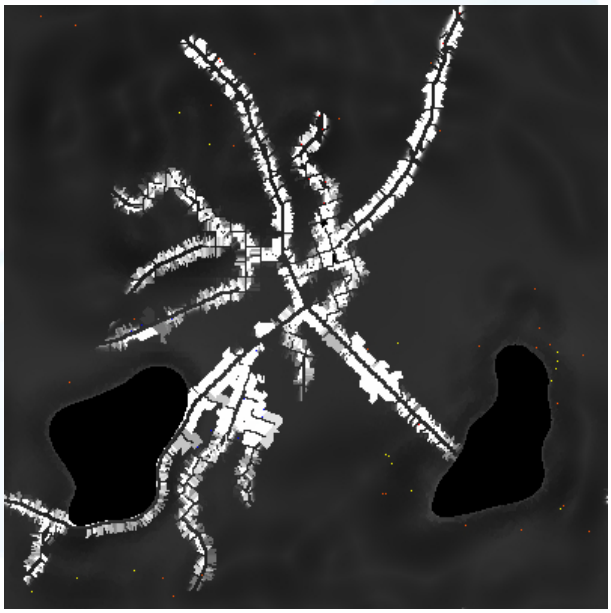
- Near water
- Near residential
- Slightly above average elevation
- Far from industrial uses



Here are the major factors that residential developers use to value land, with a corresponding value field.

Commercial value

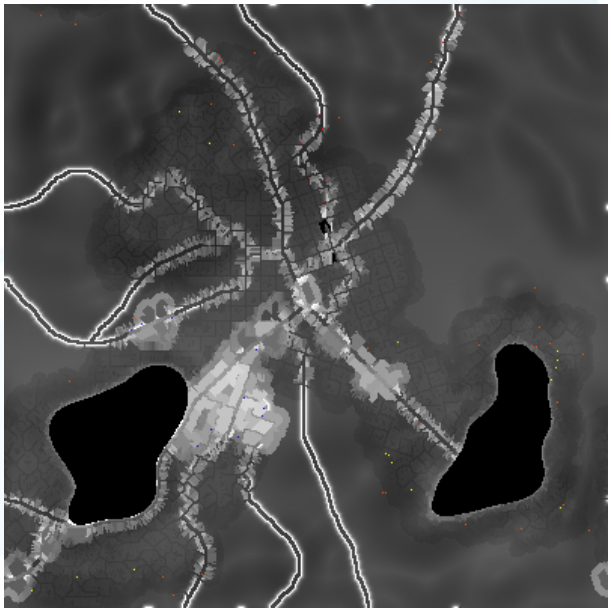
- Near market
  - Res & com
  - Near primary roads
- Near water
- Flat land
- (Away from parks)



And the major factors for commercial value, with a corresponding field.

Industrial value

- Flat land
- Near water
- Near industrial
- Near primary roads
- Far from residential
- Far from parks



Factors and field for industrial value.

Park value

- Near other parks
- Far from industrial, commercial
- Not valued by other uses
- Hilly terrain
- Near water
- Near residential

Factors for park value.

Calculating value

- *Value = Constraints × (Importance•Terrain) + Honey*

- *Constraints:* powerful limits on value
- *Importance:* relative value of terrain characteristics
- *Terrain:* attributes of local environment
- *Honey:* artist-provided development incentives

This is the high level math of calculating property value.

Importance is a vector depending on developer type. It weighs terrain attributes.

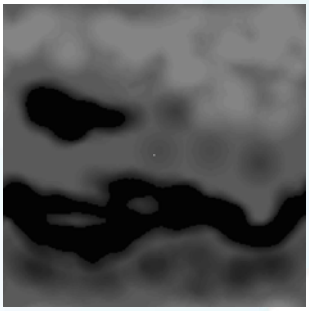
Terrain is a vector depending on location. It weighs elements such as proximity to roads, water, and usage types.

Terrain

- A vector with locally varying elements, e.g.:

$$d_w = \frac{1}{\sqrt{1 + d_w}}$$

- *Proximity to water*



$$e_h = e^{\frac{\left(E_z - \overline{E_z} - E_{offset}\right)^2}{-128}}$$

- *Elevation*

Here are two example terrain attributes: proximity to water and elevation.

Value to water proximity falls with inverse square of distance.

Residential folks like to have views and be slightly above average elevation.

Importance

- A vector of weights applied to each terrain element

e_h = *elevation*
e_v = *elevation flatness*
e_{pv} = *elevation variance*
e_{fp} = *elevation at flood plain*

d_w, d_r = *proximity to water, res*
d_i, d_{pk} = *proximity to industry, park*
d_{pr} = *proximity to primary roads*
d_m = *proximity to market*

	Importance									
Usage	e_h	e_v	e_{pv}	e_{fp}	d_w	d_r	d_i	d_{pk}	d_{pr}	d_m
Residential	0.3				0.3	0.4				
Commercial		0.2			0.15	0.15				0.4
Industrial		0.5			0.3		.1		.1	
Park			0.1	0.1	0.1	0.1		0.4		

Here is a table with most of the terrain attributes in columns, and uses in rows. A few have been omitted for clarity.

Importance weights are in rows.

Weights sum to one, once off-screen attributes are included (just a few).

For example, residential developers value above average elevation, proximity to water, and proximity to other residential development.

Constraints

- Proportions of land cover per use
- Proportions of population per use
- Density smoothness
- Per use proximity constraints

Changing the smoothness constraint

Constraints are multiplicative factors that strongly limit value.

We demonstrated the constraints earlier, here's a reminder.

Road agents

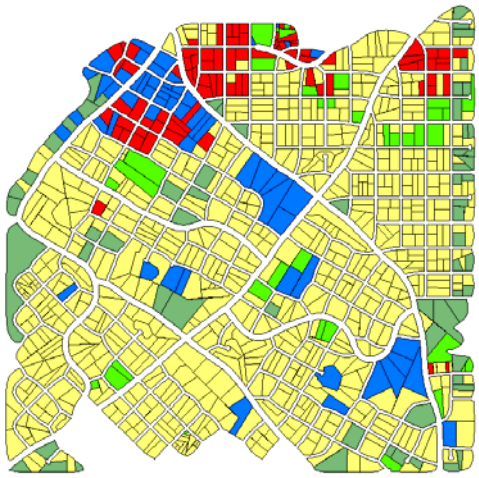
- Access road extenders
  - Ensure that developed land is reachable
- Access road connectors
  - Ensure that access network is connected
- Primary roads
  - Extend, connect primary network

There are three types of road agents.

Currently we implement only two levels in a road hierarchy; more would increase realism, and be fairly simple.

Road constraints

- All paintable:
  - Horizontal, vertical grid spacing
  - Grid orientation
  - Grid tightness
  - Overall road density



Artists can control road layouts at a high level by painting them onto the input map.

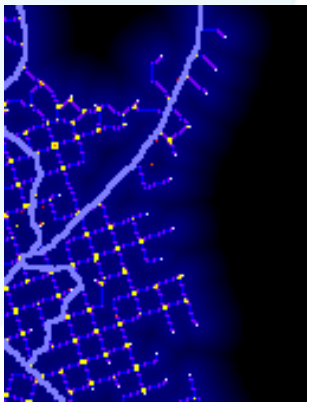
Remember that map with different griddings? Here it is again.

Orientation, tightness, spacing and overall road density can be painted (pavement/sq m).

Also roads can be directly painted by the artist.

Extenders

- Prospecting
  - Look for sites away from network
  - Check that grid constraints okay
  - Try to build
- Attempting to build
  - Shortest path back to road
  - Avoid property lines
  - Avoid elevation change
  - Check constraints

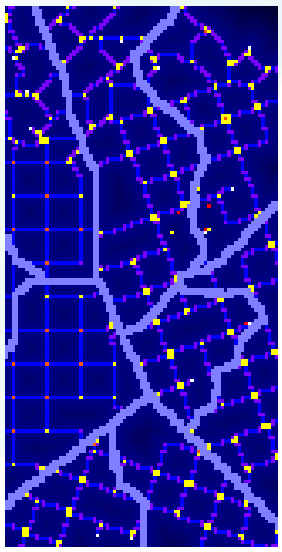


Extenders navigate through a distance-from-road landscape.

Road building does not always succeed: a completed segment may violate the road constraints.

Connectors

- Prospecting
  - Random walk of roads
  - Choose random destination
  - If shortest path too long, try building
- Attempting to build
  - Trace path to destination
  - Avoid roads, elevation change
  - Check grid constraints, road length

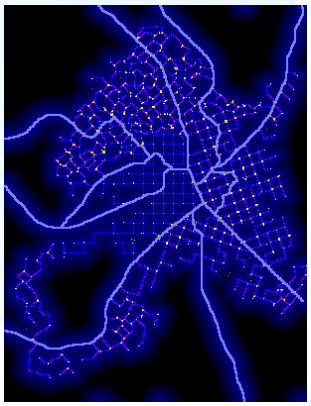


Connectors navigate on the existing road network.

If a random destination is too long on existing network, build a connecting segment.

Primary developers

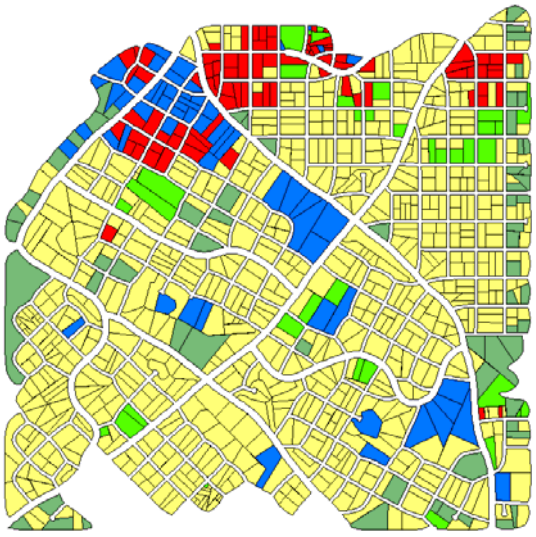
- Prospecting
  - Look for sites away from primaries
  - Avoid other primary developers
  - Build
- Building
  - Toward nearest primary
  - And toward or away from center
  - Avoid water, prefer access roads



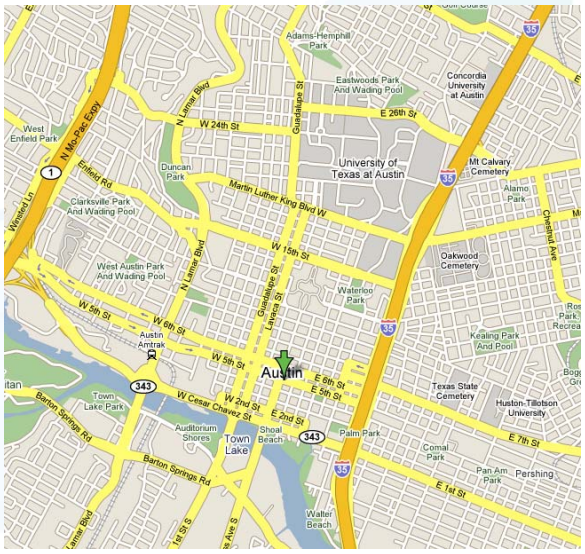
Primary road buildings navigate in a distance-from-primary road landscape.

They prefer to upgrade existing access roads to primary roads.

Simulated vs. real world



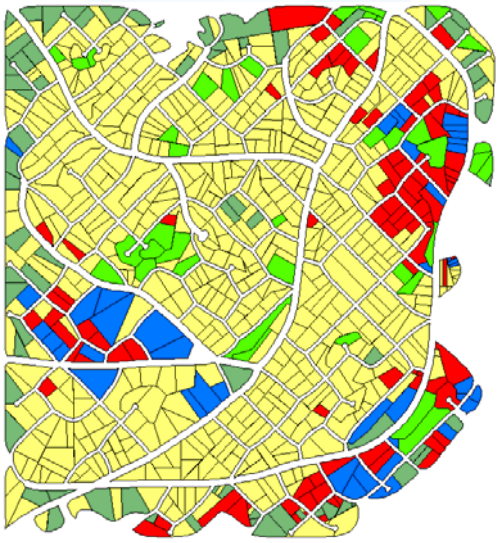
Simulated



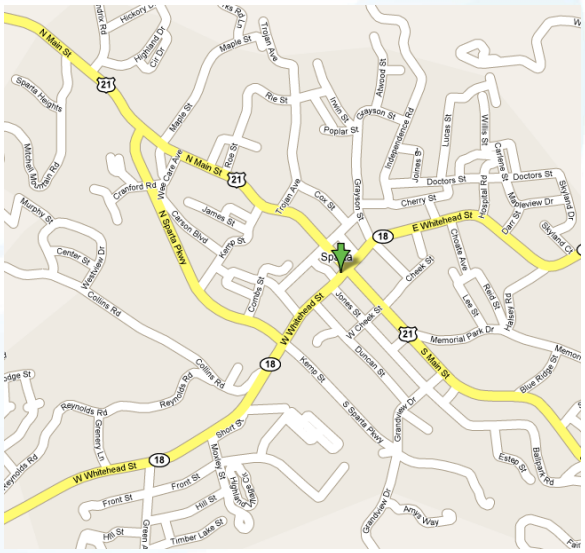
Austin

It's easy to get road layouts and satellite imagery. Less easy to get parcel lines. Hard to get trustworthy land use information!

Simulated vs. real world



Simulated



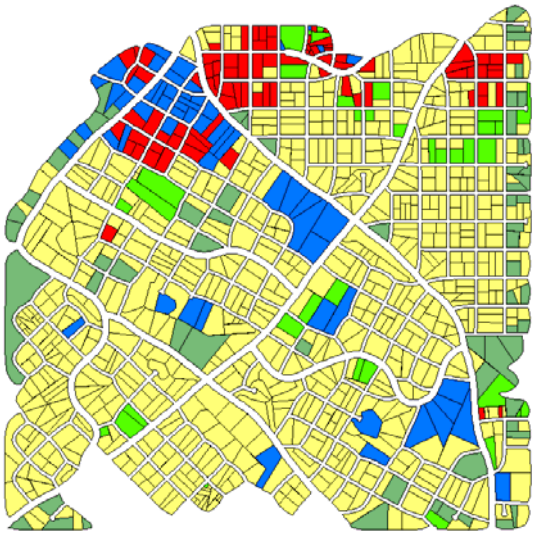
Sparta, NC

URBAN DESIGN AND PROCEDURAL MODELING
BEN WATSON: REAL AND VIRTUAL URBAN DESIGN

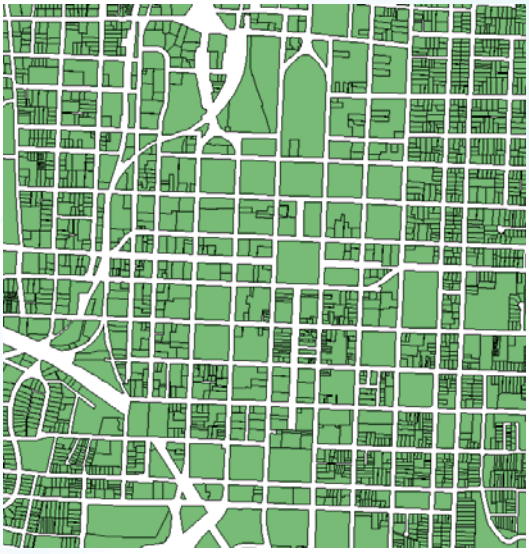
SIGGRAPH2007

Ascutney is a small New England town. It has already been bypassed by a new highway.

Simulated vs. real world



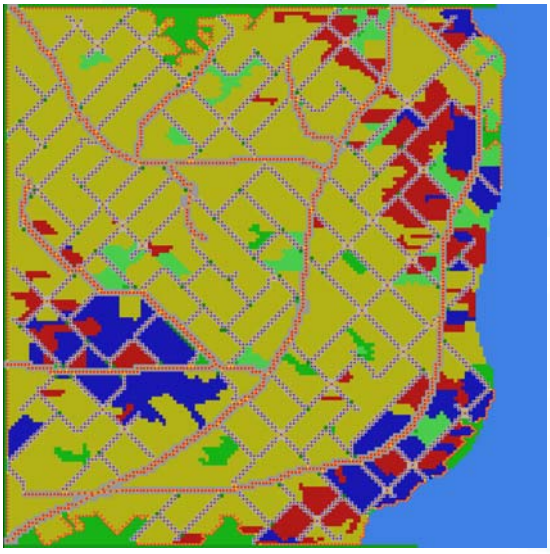
Simulated



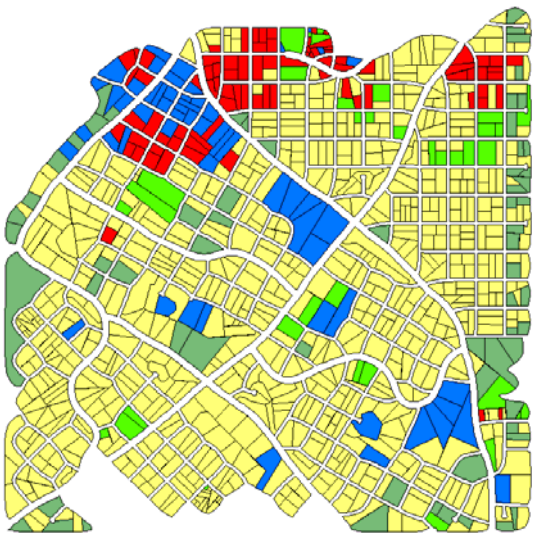
Raleigh

It's easy to get road layouts and satellite imagery. Less easy to get parcel lines. Hard to get trustworthy land use information!

Vectorization



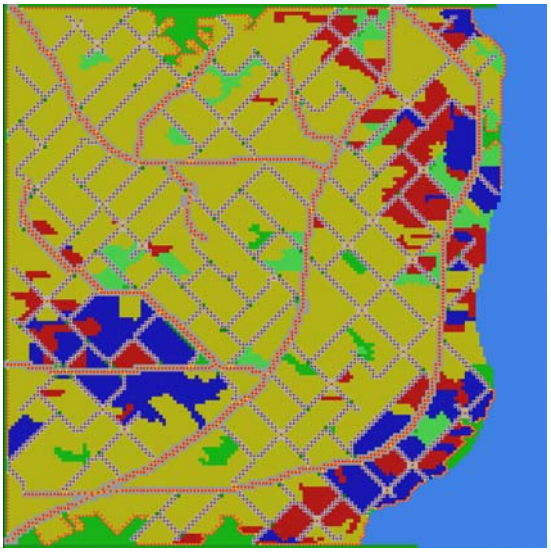
before



after

Transforming the simulation grid into something more continuous.

Vectorization process

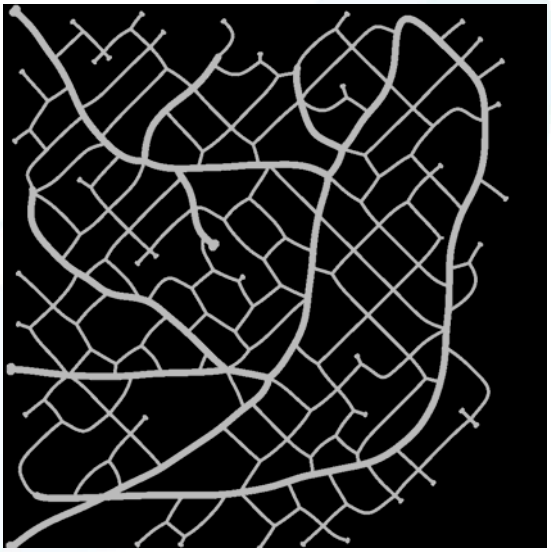


Simulated

Extract road topology

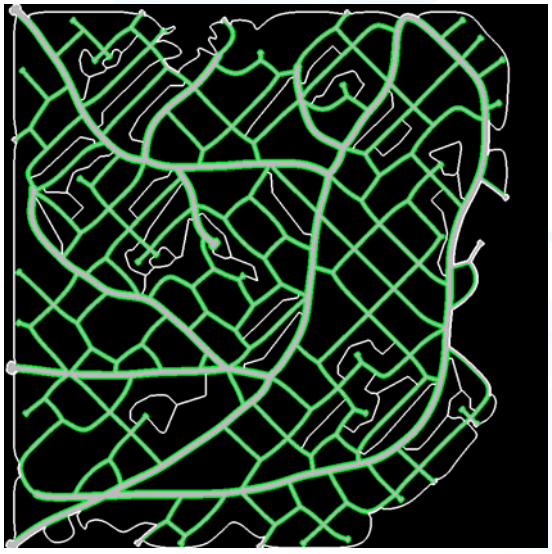
Vectorization process

Smoothing, boundaries



Add road geometry

Vectorization process



Add block spines

Subdivide blocks

Future work

- Near term
  - *Speed*
  - *Mixed use*
  - *Deeper road hierarchy*
- We are working on
  - *Higher level control*
  - *"character"*

Speed: we have focused on correctness rather than speed to date. We used NetLogo, a language and development envt implemented on top of Java. A 3x3 mile city can take several hours to develop. We believe a port to C++ and simple optimizations would increase speed several fold. This is important for realizing interactive use.

Mixed use: uses currently cannot share a parcel. This parcel sharing is quite common in urban centers.

Better parcels. As we mentioned earlier, we must improve and "smooth" parcel boundaries in a post process. We will do this using 2D polygonal optimizations.

Higher level control: We would like to describe high level character using sets of the parameters we have currently implemented, and permit artists to interact at that level, rather than just at the lower levels. For example, "Chinese" development, 18th century development, etc.

URBAN DESIGN AND PROCEDURAL MODELING

Conclusion

Discussion

Q & A