AMUSE:
Acquiring and Modeling Urban Simulation Environments

Daniel G. Aliaga
Chris Hoffmann

Spring 2006

Department of Computer Science
Purdue University

Motivation

- Visualization of large urban environments is a great challenge for computer graphics
  - An urban environment is a collection of buildings and roads spanning a large area of land and arranged into neighborhoods, blocks, and parcels

Motivation

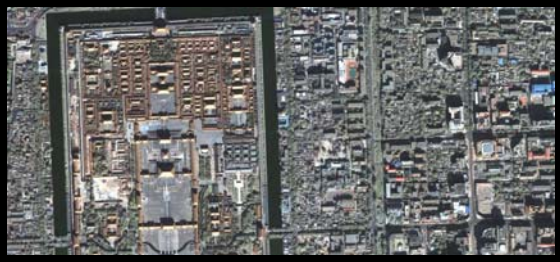
- Lafayette, Indiana

Motivation

- Indianapolis, Indiana

Motivation

- Beijing, China



Motivation

- Rome, Italy

Motivation

- Paris, France

Applications

- Urban Planning
  - What would a proposed neighborhood look like?
  - What would an area look like after population growth?
  - What would happen is we put a road here? (road planning)
  - Architectural designs wish to use common building blocks yet have unique and interesting spaces; can we extrapolate a city given a set of building blocks?

Applications

- Emergency Management
  - Can we create a model of a very large urban space to train emergency response personnel?
  - Can we plan evacuation routes and suggest emergency deployments?
  - Can we prioritize policing and resource deployment?
  - Given an urban model struck by a disaster/attack, we can deploy an emergency-relief communication network?
  - Can we deploy approximate structural information, via a PDA, to rescuers using both the urban model and the emergency-relief communication network?

Applications

- Reconnaissance and Rapid Prototyping
  - Can we rapidly build a prototype of an (enemy) location from aerial views?
  - Can we specify the most beneficial places from where to obtain ground views so as to build an urban model for soldier training and other simulations?
  - If an environment is changing, can we indicate from where we need the most updates?

Challenge

- Modeling large urban spaces typically requires significant manual effort, storage, and computation
- Dense urban areas are particularly difficult because they are both very complex and very widespread
  - Size of the environment makes obtaining detailed structural information prohibitive, leaving us with only sparse information

Observations

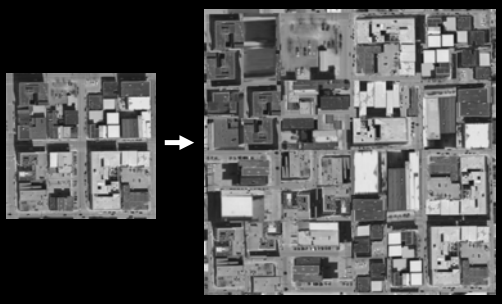
- Large urban environments exhibit significant repetition
  - Similar structures are repeated at the global level; however, they maintain individuality in local detail
- Widespread digital meta-data is available
  - High-resolution aerial views
    - e.g., 6 inch/pixel
  - City/county/parcel boundaries, road networks, basic building information
    - e.g., input to Google Maps

Approach

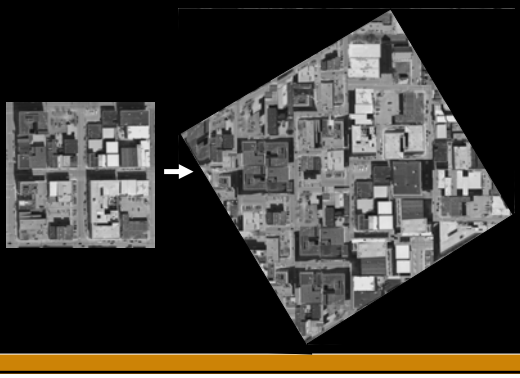
- Perform an inverse urban modeling task by inferring the 2D layout of an existing environment
  - Procedural methods have the advantage of exhibiting a high-degree of detail amplification, e.g. using a small number of parameters yields significant plausible details
- The resulting grammar allows us to
  - create modifications to the existing urban environments, in the style of the original
  - determine the most representative areas and layouts

Example: Changing Urban Spaces

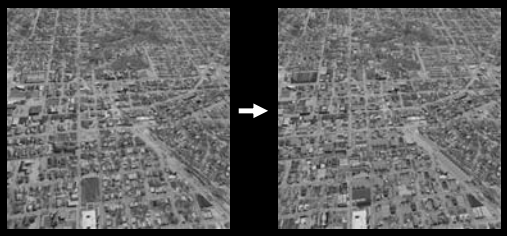
Example: Changing Urban Spaces



Example: Changing Urban Spaces



Example: Simplifying Urban Spaces



Thousands of parcels/buildings

Dozens of parcels/buildings

Related Work

- Forward-generating grammars (L-systems) for creating plants, cities, and buildings
  - Specify a grammar and few initial parameters, then "grow" the structure
  - × Not based on an actual real-word layout or city
- Photogrammetric Reconstruction and IBMR
  - Build a model from photographs (e.g., Façade, Lightfields, robot-based acquisition)
  - × Requires specialized acquisition systems and does not scale
- Inspiration
  - Epitomes and Vector Quantization
  - Build-by-Number
  - One paper: inferring plant L-system parameters from photographs

Terminology and Assumptions

- Parse
  - From aerial views and meta-data, create a set of production rules and a set of terminals
    - e.g., string to grammar
- Derive
  - Using the production rules, terminals, and a starting configuration, create an urban layout
    - e.g. grammar to string

Terminology and Assumptions

- Parcel
  - consists of a piece of bounded land; might contain building structures
- Block
  - Collection of adjacent parcels; interior boundaries are all imaginary; exterior boundary is a road; all parcels have access to road (egress rule)
- Neighborhood
  - Collection of blocks, separated by roads, and mostly of the same classification (e.g. “residential”, “commercial”, “industrial”, “downtown”, etc.)
- Region
  - Collection of neighborhoods, usually separated by major roads

Terminology and Assumptions

- Production
  - Given a region | neighborhood | block, partition by a road | boundary
    - Assumptions (for now): regions are convex polygons, partitions are polylines, production produces two children
- Terminal
  - Is a parcel
  - May or may not contain building contours

Methodology

- 1. Parsing
  - How to parse aerial views and their metadata
- 2. Terminal Simplification
  - Reducing the number of terminals
- 3. Production Simplification
  - Reducing the number of productions
- 4. Novel Derivations
  - Making new layouts

Methodology

➡ • 1. Parsing
  - How to parse aerial views and their metadata
- 2. Terminal Simplification
  - Reducing the number of terminals
- 3. Production Simplification
  - Reducing the number of productions
- 4. Novel Derivations
  - Making new layouts

1. Parsing

- Parse aerial views in a top-down fashion to produce a set of production rules for creating the urban layout
- Existence Question
  - Does such a grammar exist?
- Answer:
  - Yes! It is exactly one production rule for each partition and exactly one terminal for each parcel
- *The interesting work is in simplifying and compacting the grammar so that it can be used in a flexible fashion*

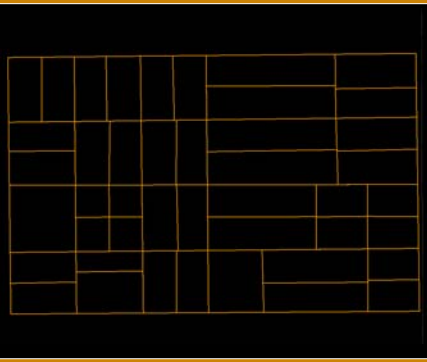
Example Aerial View I

Example Metadata I

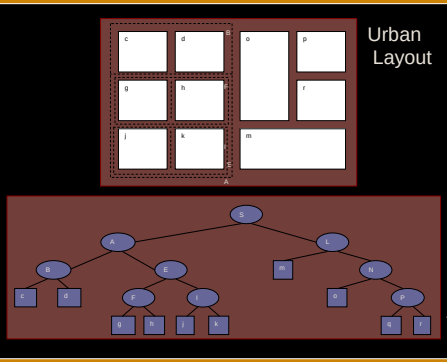
Example Aerial View II

Example Metadata II

Example Metadata II



Parse Tree



Production Rules

S → AL
A → BE
L → mN
B → cd
E → FI
F → gh
I → jk
N → oP
P → qr

~regions
~neighborhoods
~blocks

(lower case = terminals)
(upper case = region|neighborhood|block)

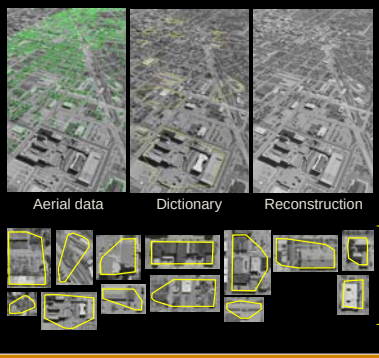
Methodology

- 1. Parsing
  - How to parse aerial views and their metadata
- 2. Terminal Simplification
  - Reducing the number of terminals
- 3. Production Simplification
  - Reducing the number of productions
- 4. Novel Derivations
  - Making new layouts

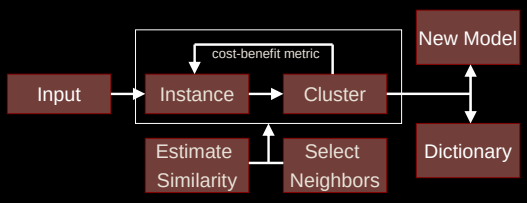
2. Terminal Simplification

- Reduce the number of terminals
  - Find a compact "dictionary" of urban structures
- Serves to
  - make grammar more compact
  - compress the data
    - e.g. image compression
  - prioritize the terminals
    - e.g. guides which parcels should be captured in more detail

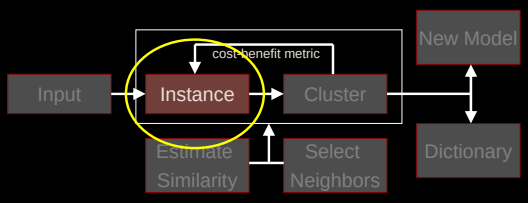
Urban Dictionaries



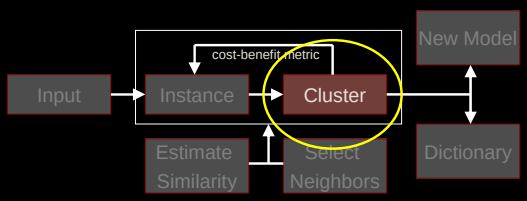
Simplification Pipeline



Simplification Pipeline

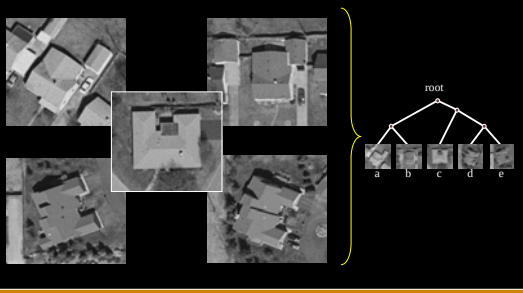


Simplification Pipeline

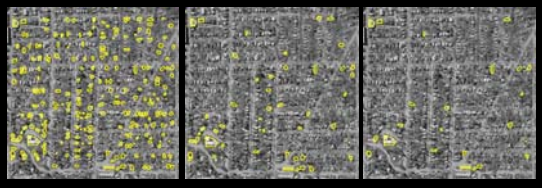


Instancing

- Global redundancy is exploited by growing a bottom-up hierarchy of equivalence classes



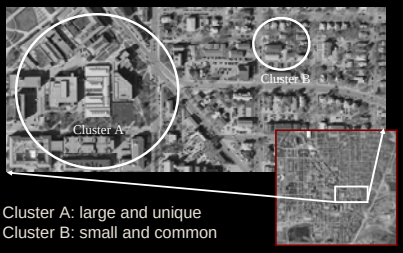
Instancing



Increasing error (and smaller number of equivalence classes)

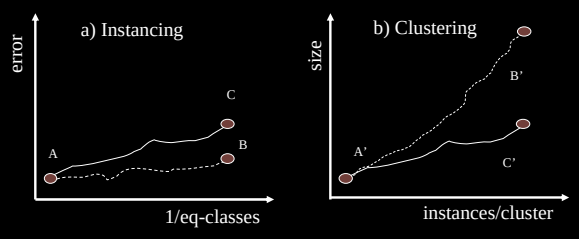
Clustering

- By combining instantiation with spatial clustering, we find spatially compact sets of representative urban areas

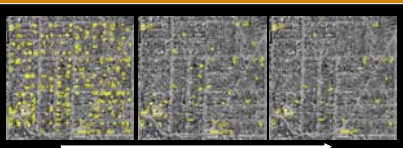


Cluster A: large and unique
Cluster B: small and common

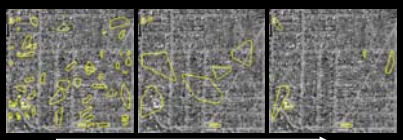
Instancing vs. Clustering



Instancing vs. Instancing-and-Clustering

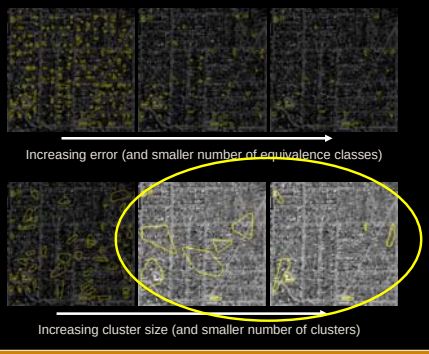


Increasing error (and smaller number of equivalence classes)



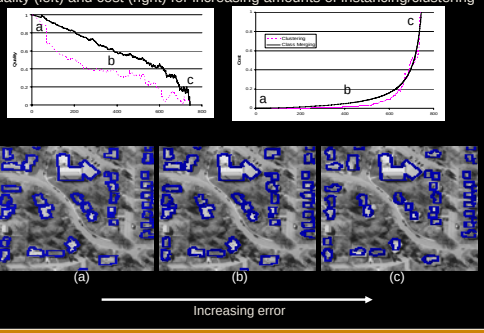
Increasing cluster size (and smaller number of clusters)

Instancing vs. Instancing-and-Clustering

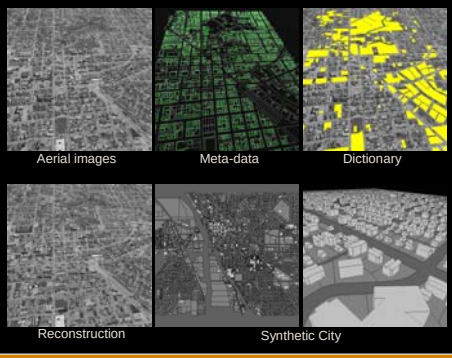


Instancing and Clustering

Quality (left) and cost (right) for increasing amounts of instancing/clustering



Example Terminal Simplifications



Methodology

- 1. Parsing
  - How to parse aerial views and their metadata
- 2. Terminal Simplification
  - Reducing the number of terminals
- ➡ 3. Production Simplification
  - Reducing the number of productions
- 4. Novel Derivations
  - Making new layouts

3. Production Simplification

- Goal
  - Find a set of representative "rules" of the urban environment that can instantiate the same space, new spaces, and similar spaces
- Method
  - Find a dictionary of "rules" to build-up the urban space, for example:
    - Discover the rules by analyzing the layouts
    - Provide core rules that can represent all possible layouts

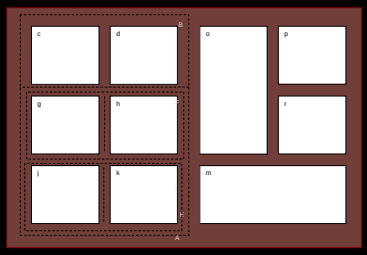
3. Production Simplification

- Rule Clustering
- Rule Canonization

3. Production Simplification

- Rule Clustering
- Rule Canonization

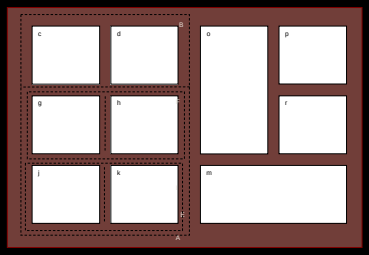
Recall the Original Production Rules…



S → AL
A → BE
L → mN
B → cd
E → FI
F → gh
I → jk
N → oP
P → qr

Full specification of the production rules

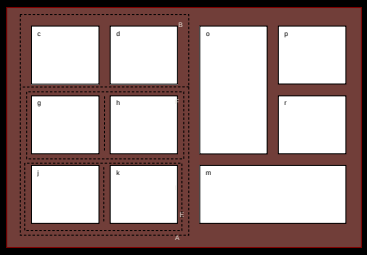
Alternate Rule Notation



$S \rightarrow A_0\ A_1^{180}$
$A_0 \rightarrow B_0E_0$
$B_0 \rightarrow cc$
$E_0 \rightarrow B_0B_0$
$A_1^{180} \rightarrow E_1m$
$E_1 \rightarrow mB_1^{90}$
$B_1^{90} \rightarrow cc$

Shape = letter; position = subscript; rotation = superscript

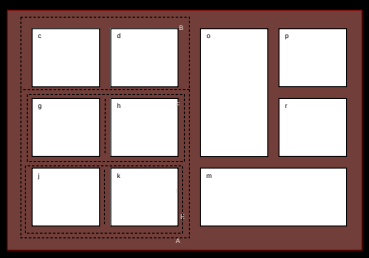
Rule Shape Similarity



S → AA
A → BE | mE
B → cc
E → BB | mB

Rules grouped based on similar shape but *ignoring* differences in location and rotation

Rule Shape Similarity



S → AA
A → BE | mE
B → cc
E → BB | mB

= "Cluster by similar shape"

Rules grouped based on similar shape but *ignoring* differences in location and rotation

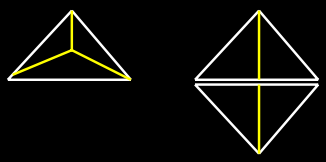
Rule Clustering

- Similarity(A,B) =
  - W_1ShapeSim(A,B) + W_2LocationSim(a,B) + W_3PositionSim(A,B) + W_4PartitionSim(A,B) + W_5TypeSim(A,B)
- Production rule clustering:
  - Define each rule by a n-dimensional vector
  - Perform k-means clustering to obtain k clusters of rules
  - Choose a representative rule from each cluster
- Effectively "infer" the most popular/representative production rule styles

Note: ???

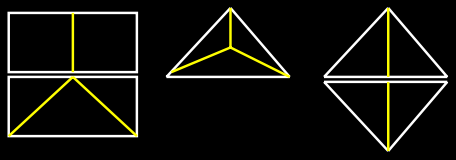
Rule Canonization

- Reduce the full grammar to a more compact one using only a small number of flexible yet powerful canonical production rules
- Consider triangles:



Rule Canonization

- Reduce the full grammar to a more compact one using only a small number of flexible yet powerful canonical production rules
- Consider rectangles:



Rule Canonization

- Reduce the full grammar to a more compact one using only a small number of flexible yet powerful canonical production rules

Rule Canonization

- Reduce the full grammar to a more compact one using only a small number of flexible yet powerful canonical production rules

Rule Canonization

- Reduce the full grammar to a more compact one using only a small number of flexible yet powerful canonical production rules

Rule Canonization

- Reduce the full grammar to a more compact one using only a small number of flexible yet powerful canonical production rules

Rule Canonization

- Reduce the full grammar to a more compact one using only a small number of flexible yet powerful canonical production rules

Rule Canonization

- Reduce the full grammar to a more compact one using only a small number of flexible yet powerful canonical production rules

Rule Canonization

- Reduce the full grammar to a more compact one using only a small number of flexible yet powerful canonical production rules

Rule Canonization

- Reduce the full grammar to a more compact one using only a small number of flexible yet powerful canonical production rules

Rule Canonization

- Reduce the full grammar to a more compact one using only a small number of flexible yet powerful canonical production rules

Peel | Rectangular partition | Triangular partition | Create an Island | Etc…

Note: ???

Methodology

- 1. Parsing
  - How to parse aerial views and their metadata
- 2. Terminal Simplification
  - Reducing the number of terminals
- 3. Production Simplification
  - Reducing the number of productions
- ➡ 4. Novel Derivations
  - Making new layouts

Novel Derivations

- Given production rules and an initial structure, derive an urban layout
- Examples:
  - Change original urban space
  - "Move a road" in the original urban space
  - Fill a new region with an urban space similar to the original
  - Grow an urban area

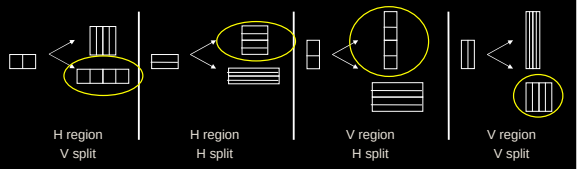
Novel Derivations

- Need to support subset of affine transformations:
  - Translation (e.g., "move a region")
  - Rotation (e.g., "re-orient a neighborhood")
  - Scale (e.g., "stretch/squish a block")

Transformations

- Translation
  - Easy to handle
- Rotation
  - Easy to handle
- Scale
  - ???

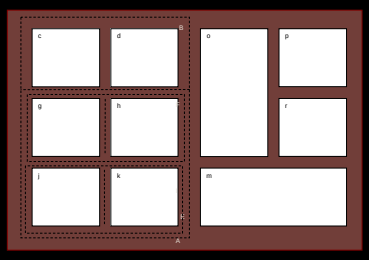
Observation: Scaling/Stretching



Conclusion:

- V stretches handled by H partition
- H stretches handled by V partition

Production Rules Revisited…



$S \rightarrow^{V} AA$
$A \rightarrow^{H} BE \mid mE$
$B \rightarrow^{V} cc$
$E \rightarrow^{H} BB$
$E \rightarrow^{V} mB$

Example Derivations

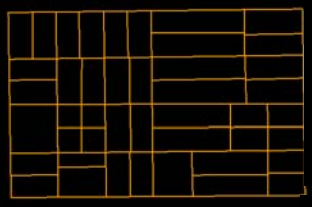
- Original urban space
  - $S \rightarrow AA$
  - $S \rightarrow BEmE$
  - $S \rightarrow ccBBmmB$
  - $S \rightarrow cccccmmcc$
- Stretched urban space
  - $S \rightarrow AA$
  - $S \rightarrow BEmE$
  - $S \rightarrow BEBEmEmE$
  - $S \rightarrow ccBBccBBmmBmmB$
  - $S \rightarrow ccccccccccmmccmmcc$

Note: for simplicity, these derivations are showing 1D strings – data is really 2D

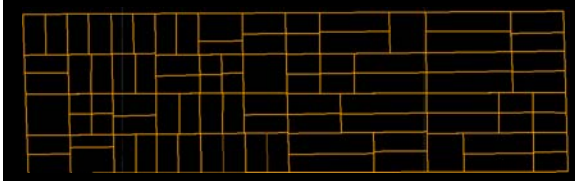
Original Urban Space I

Original Urban Space I

Original Urban Space I



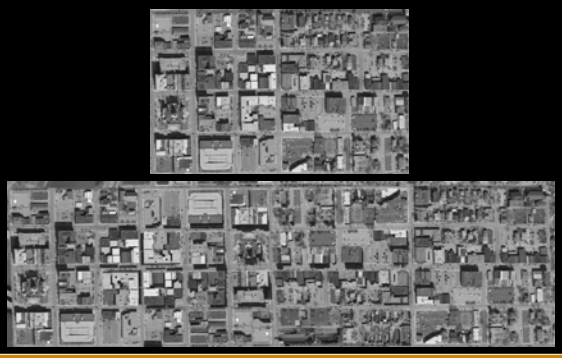
Stretched Urban Space I



Stretched Urban Space I

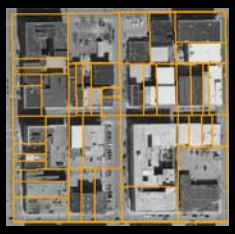
Stretched Urban Space I

Original vs. Stretched I

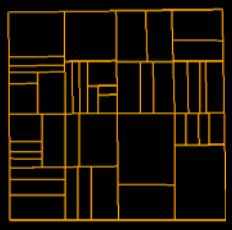


Original Urban Space II

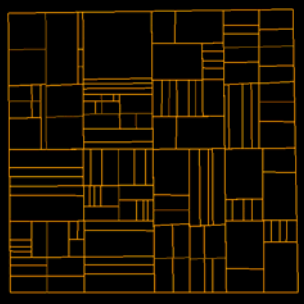
Original Urban Space II



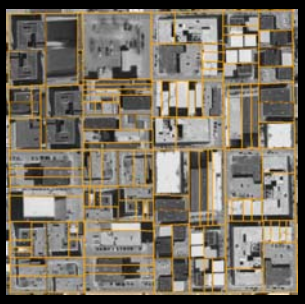
Original Urban Space II



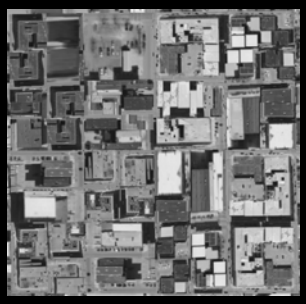
Stretched Urban Space II



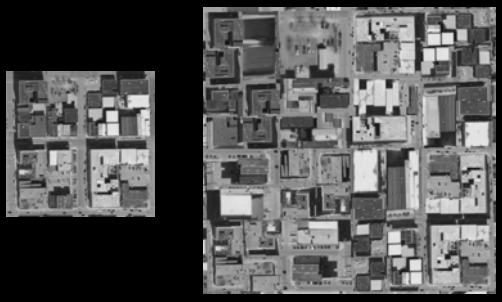
Stretched Urban Space II



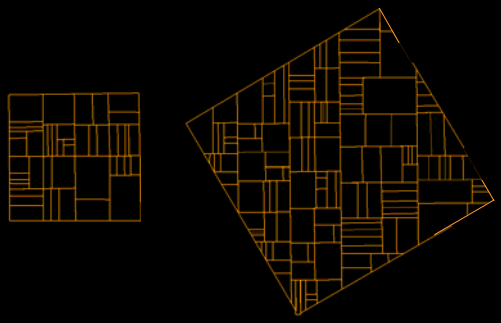
Stretched Urban Space II



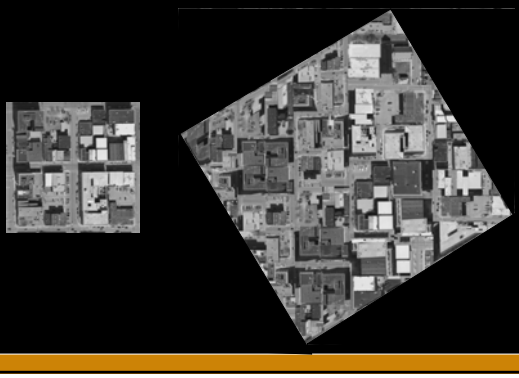
Original vs. Stretched II



Original vs. Stretched III



Original vs. Stretched III



Demo

Conclusions

- Urban Modeling is fun!
- Ability to be able to take views of an entire urban space and make an editable model out of it is very enticing
- Similar to fractal-based compression, the key is to find a good set of generators
  - Fortunately, urban environments offer significant amount of structure (and repetition) which we can exploit

Future Work

- Procedural Simplification
  - Specify canonical procedural rules?
  - Infer canonical procedural rules?
- Obtain data for a larger/more-interesting set of cities
  - Chicago, Rome, Paris, Cusco, etc.
- Full Inverse Modeling
  - Combine with Build-by-Numbers
- Applications
  - Road planning
  - Growth algorithms
  - Rapid prototyping

Thank you!