Graphics Lab: Basic C++

Final project:

Procedural Building Generator

Team 2:

- *Naurin Jamil*
- *Ruotong Li*
- *Nicola Krombach*
- *Jonas Abert*

Task:

- *C++ implementation of the paper*

Procedural Modeling of Buildings

Pascal Müller* ETH Zürich

Peter Wonka† Arizona State University

Simon Haegler* ETH Zürich

Andreas Ulmer* ETH Zürich

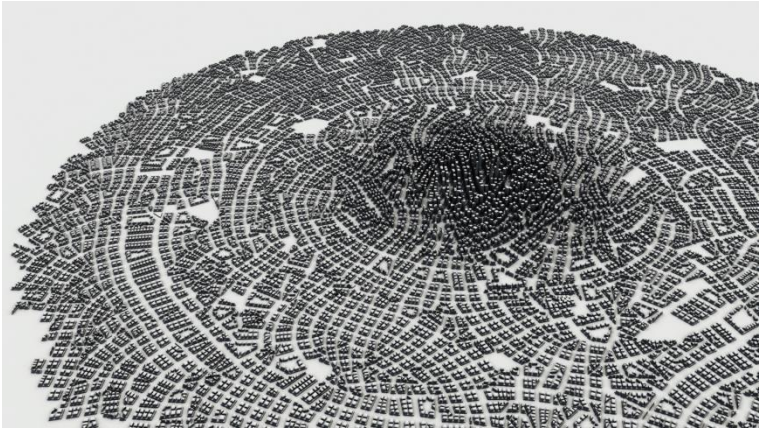
Luc Van Gool* ETH Zürich / K.U. Leuven

Procedural modeling

“[…] produces extensive [3D] models for computer games and movies, at low cost. Context sensitive shape rules allow the user to specify interactions between the entities of the hierarchical shape descriptions.”

Müller et al. 2006

Application:

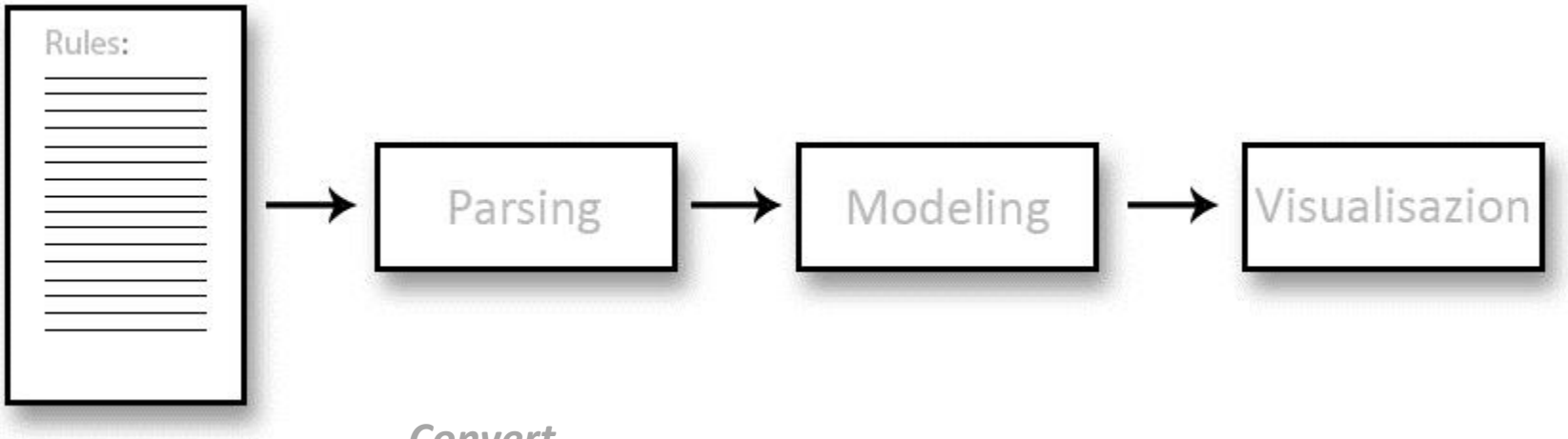


[1]

[2]

[1] http://lesterbanks.com/
[2] http://adobe.com

Basic Workflow:

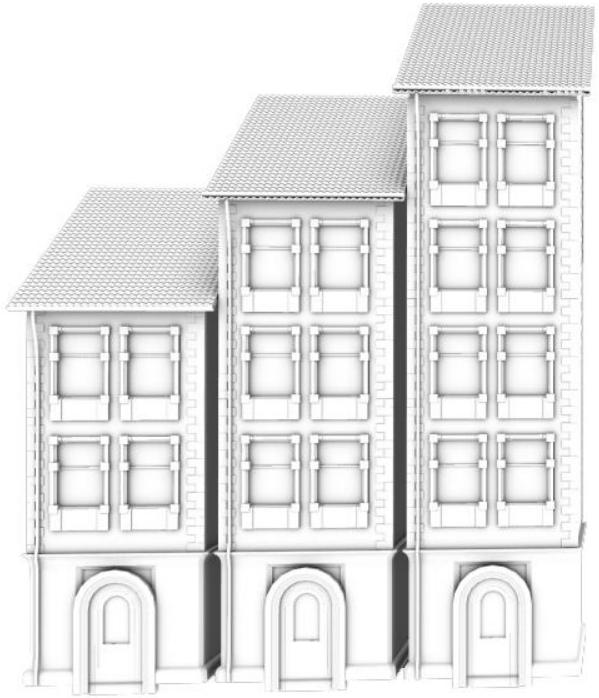


Text File (Grammar)

Convert Grammar into Rules

Apply Rules

Show result



Integration:

Jonas Abert

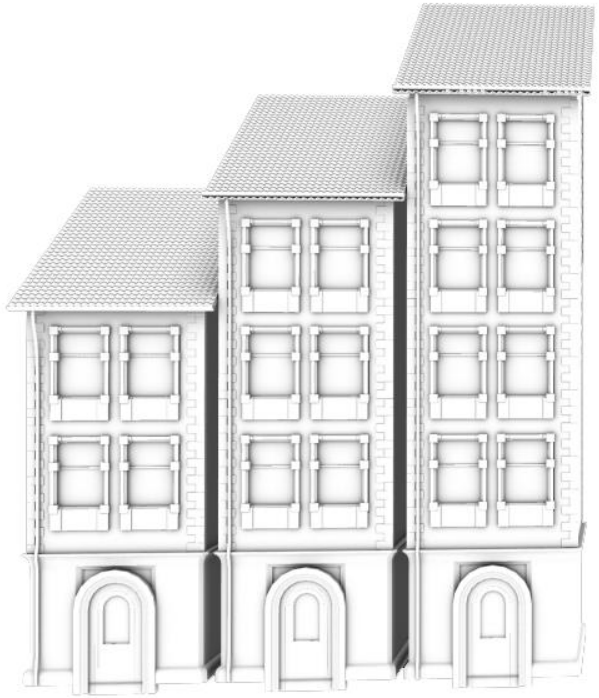
Tasks

- ***Development***
  - *Design of the general program logic*
  - *Implementation of a framework that supports our needs*
- ***Management***
  - *Project coordination*
  - *Keeping an overview over the project at any given time*
- ***Support***
  - *Helping the others at developing and implementing their parts*

Goals

- ***Flexibilty***
  - *wide varriaty of procedural models*
  - *Definition through Grammar file only*

- ***Simplicity***



Parsing:

Naurin Jamil

Tasks

- ***Development***
  - *Affix the format for the grammar file to model different structures*
  - *Implement Split string to parse the rules*
  - *Design initial version of the Final Grammar*
  - *Affix the Grammar Data Structure to be implemented for modelling steps*

- ***Testing***
  - *Use a Test grammar to test the code for correct parsing*

Why do we need to parse?

- *Parsing is the process of analysing a string of symbols in the language, according to the rules of a formal **grammar**.*

- *It helps to perform **formal analysis** of a line or sentence(in our case, a line of rule) into its constituents, resulting in a parse tree, showing a syntactic relation to each other, which may contain semantic and other information.*

- *Our Parser is a software component that takes input data as a Rules.txt file and builds a **dataStructure** from it, giving a clear representation of the input data and checking correct syntax in the process.*

How do we parse?

- ***Parsing is implemented in the following steps :***
  - *A grammar file format is specified, as in the referance paper, with a few alterations to simplify the lexical analysis. It is written as a* ***Rules.txt*** *file, that serves as the input for parsing*
  - *The individual rules are read from Rules.txt file, rule constituents are identified and filled in the object/dataStructure as its different parameters, that are used in the subsequent implementation steps.*

Implementation

sidewing->S(1r,1r,Scope.sz*rand[0.4-1.0]) {facades}:0.5->S(1r,Scope.sy*rand[0.2-0.9]) {facades}: 0.3-> () {epsilon}: 0.2

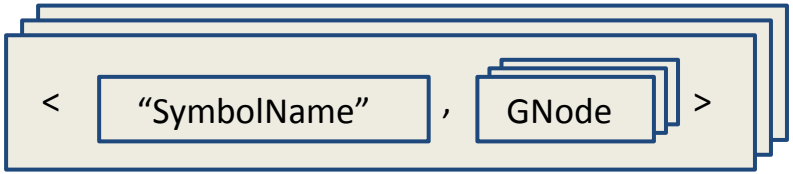
- *sidewing – **SymbolID***
- *S – **Function***
- *(1r,1r,Scope.sz*rand[0.4-1.0])- **parameters***
- *facades – **NewIDs/ SymbolIDs***
- *0.5 - **probability***
- *.. And so on are the segregated consituents of the rule obtained by parsing, that are filled as the object parameters*

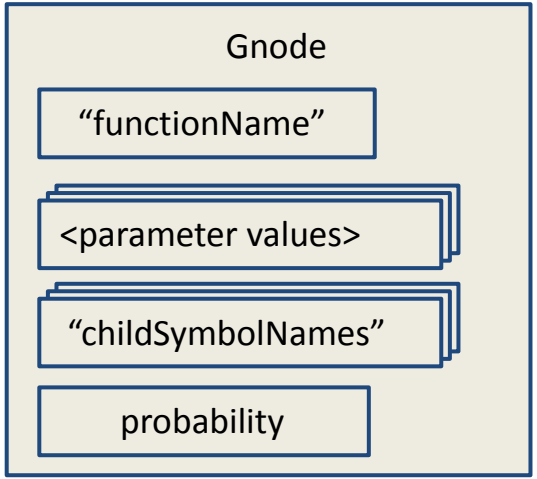
Parsing code Logic

What is the code logic behind Parsing?

- *Two text files are passed as input, one for reading the model configuration as* ***config.txt*** *and other for the defined rules as* ***Rules.txt***
- *Read the configuration in the defined function and save the parameters in a* ***map<ID, value>config***
- *Split the read rule lines by calling the split function and populate the vectors of type string with the rule constituents like,* ***function, SymbolIDs, parameters, probability***
- ***GNode*** *is our defined dataStructure that stores the above constituents*
- *Finally, we define* ***vector<pair < SymbolID, vector<Gnode>> ruleSet.***

Visual Representation





Results

- ***Output of parsing***

  Parsing results in populating the vector<pair< SymbolID, vector<Gnode>> ruleSet.

  This map is the dataStructure used by the subsequent Modelling step.

  - *The keyword SymbolID is used to query the map, so it is essenially unique*
  - *All the functions applied on this symbol are stored in the vector of Gnode*
  - *So, one Gnode only stores the information of one function, also its probability*

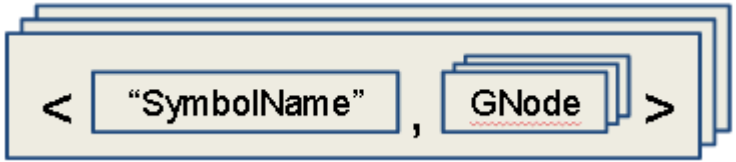
Modeling:

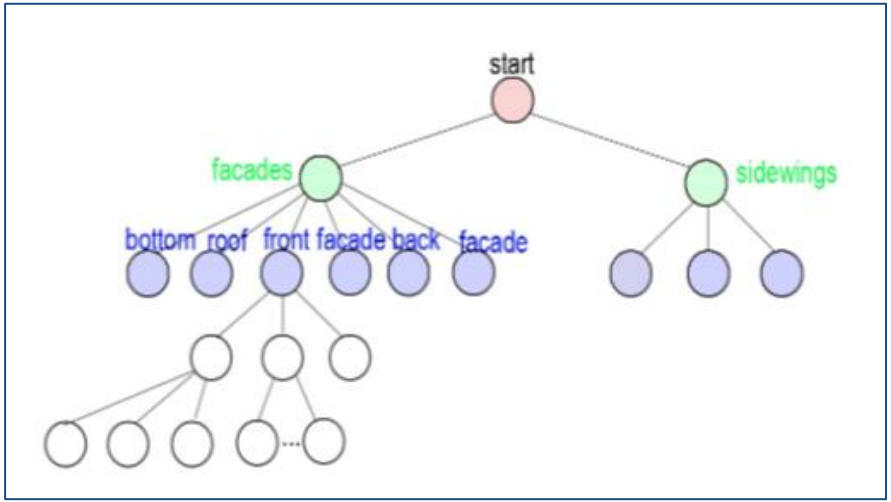
Ruotong Li

Tasks

- *Read rule from rule set I got from Parsing*
- *apply GNode on Symbols*
- *store new created Symbols into the deviation tree*

How to do it?





Task:

using the ruleset to create a **deviation tree**

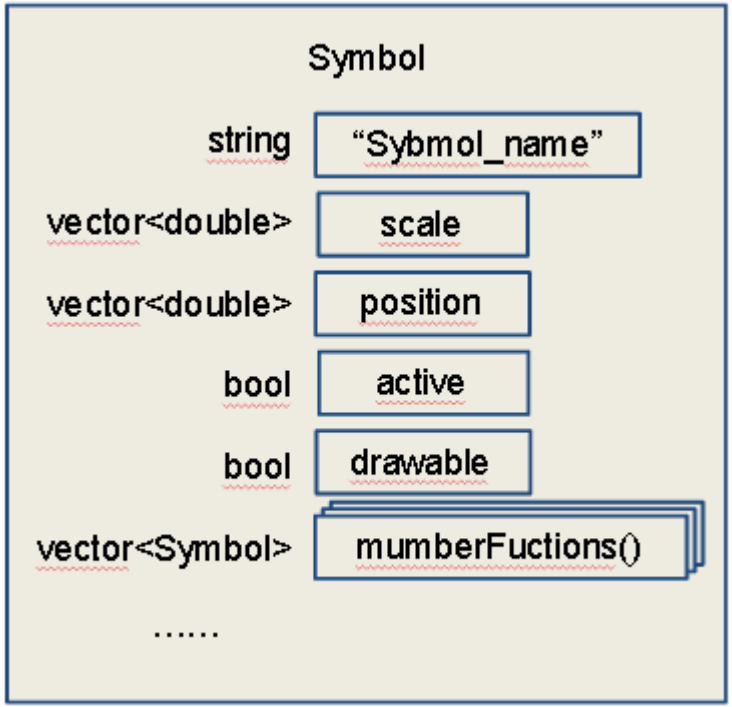
Idea:

take the pair one by one to GNode on the **tree_elements**

What is the tree_element?

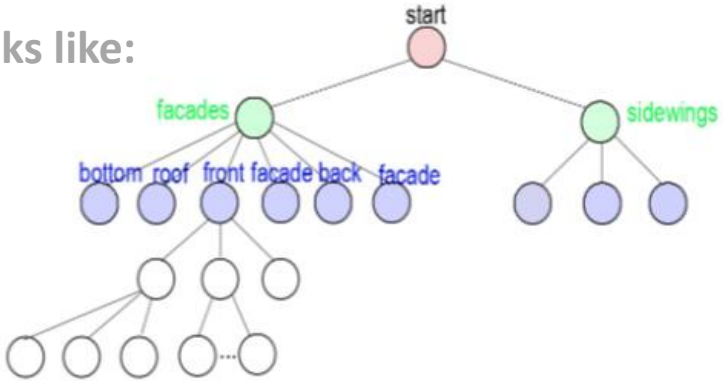
- **Symbol**

- it is the basic element of the deviation tree
- it represents the simple 1D, 2D or 3D elements
  - lines, plane and boxes
- it contains element atributes
  - position, scale, name
- it can apply rules to create new Symbols

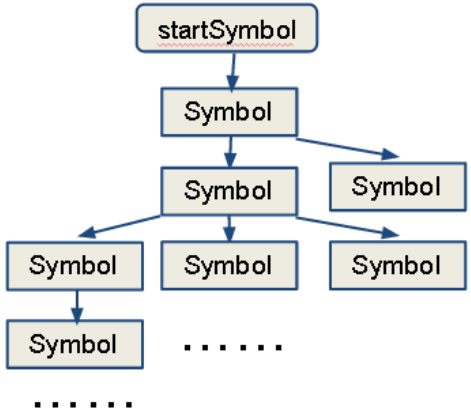


What is the deviation tree?

what it looks like:



what it really is:

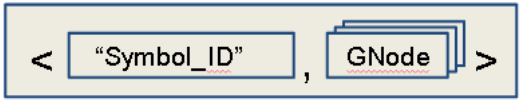


The tree contains all Symbols created in the process

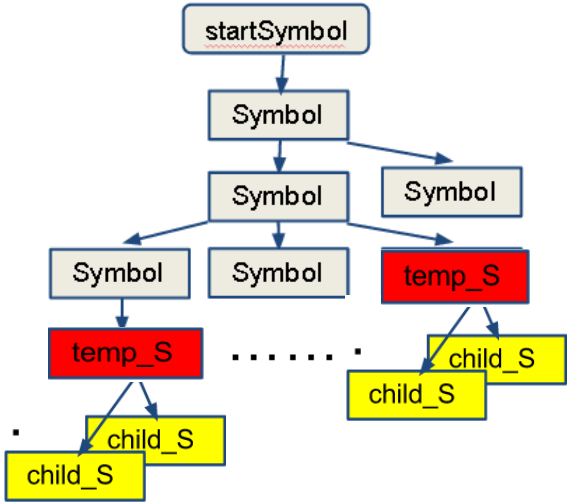
- Nodes in the tree are Symbols
- Get child_node by applying Gnode on parent_node

How is the tree created?

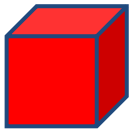
- take a rule from the rule set



- search “Sybmol_ID” through the tree, mark them
- apply GNode(s) on marked Sybmols
- append childSymbol with the function



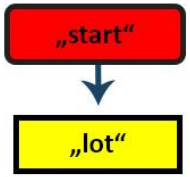
Let’s start it with a box:



position	(0, 0, 0)
scale	(1, 1, 1)
name	“start”

start.sale(s1, s2, s3, “lot”)

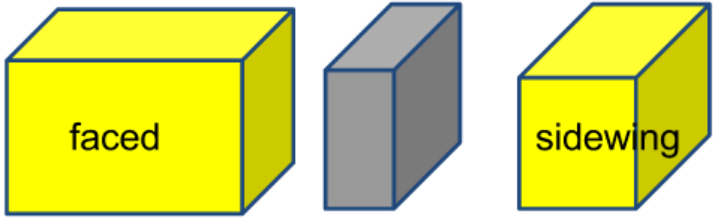
position	(0, 0, 0)
scale	(s1, s2, s3)
name	“lot”



then

position (0, 0, 0)
scale (s1, s2, s3)
name “lot”

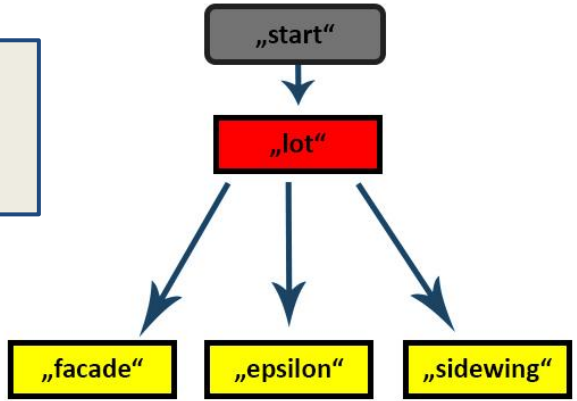
subDiv (“X”, d1, d2, d3 , “faceds”, “epsilon”, “sidewing”)



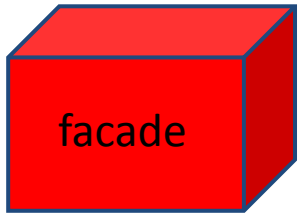
(0, 0, 0)
(d1, s2, s3)
“faced”

(d1, 0, 0)
(d2, s2, s3)
“epsilon”

(d1+d2, 0, 0)
(d3, s2, s3)
“sidewing”

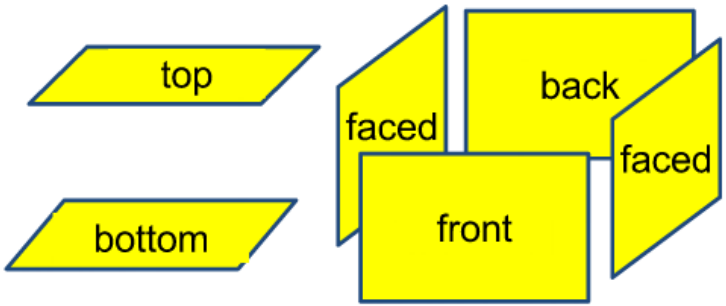


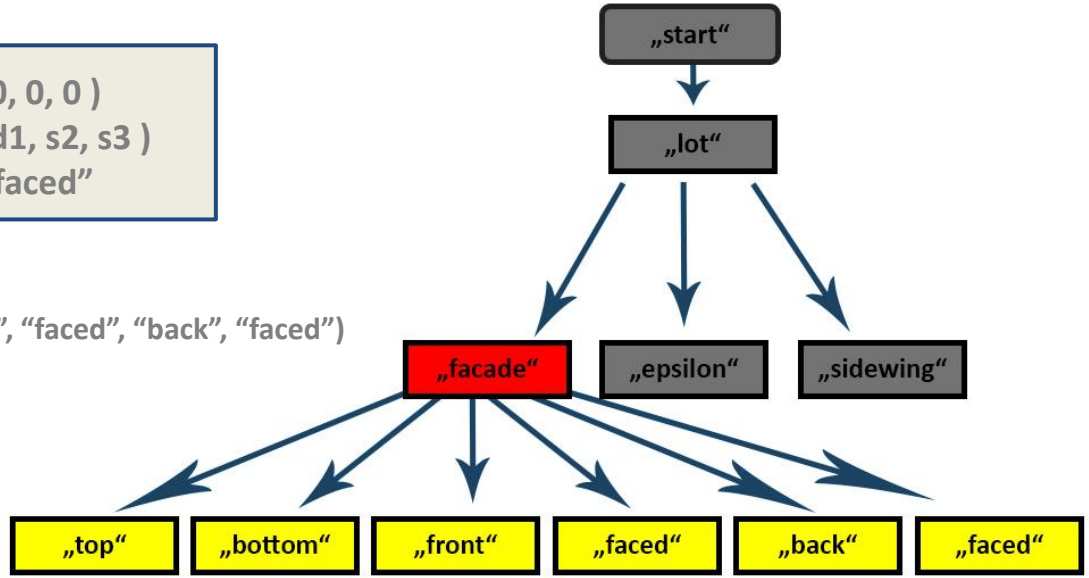
Now we can apped the tree



position	(0, 0, 0)
scale	(d1, s2, s3)
name	“faced”

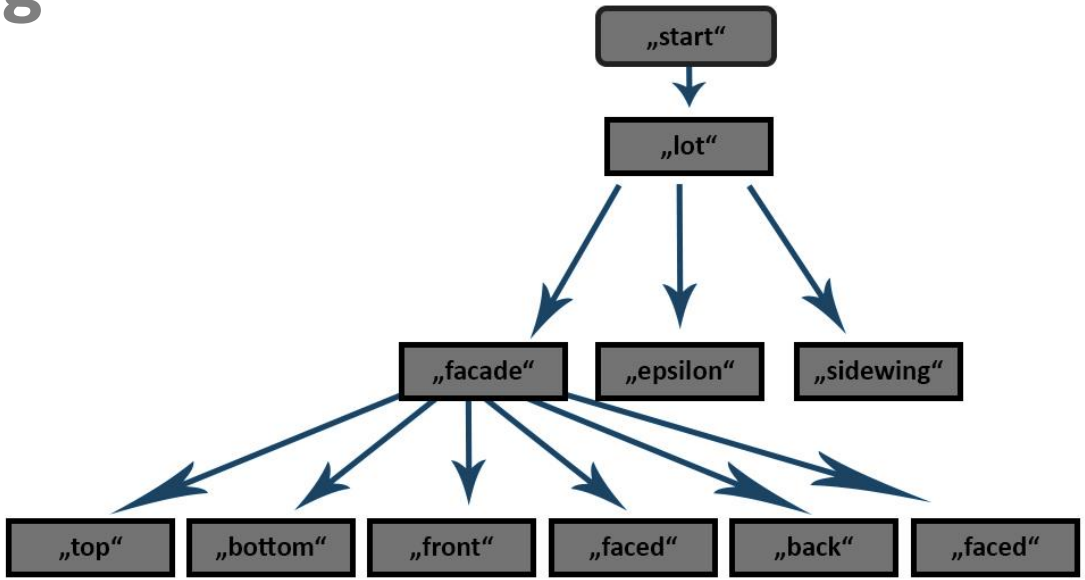
comp (“top”, “bottom”, “front”, “faced”, “back”, “faced”)





Functions & Naming

- **Scope rules:**
  - Translation
  - Scaling
- **Production rules:**
  - Subdivision
  - Component split
  - Renaming



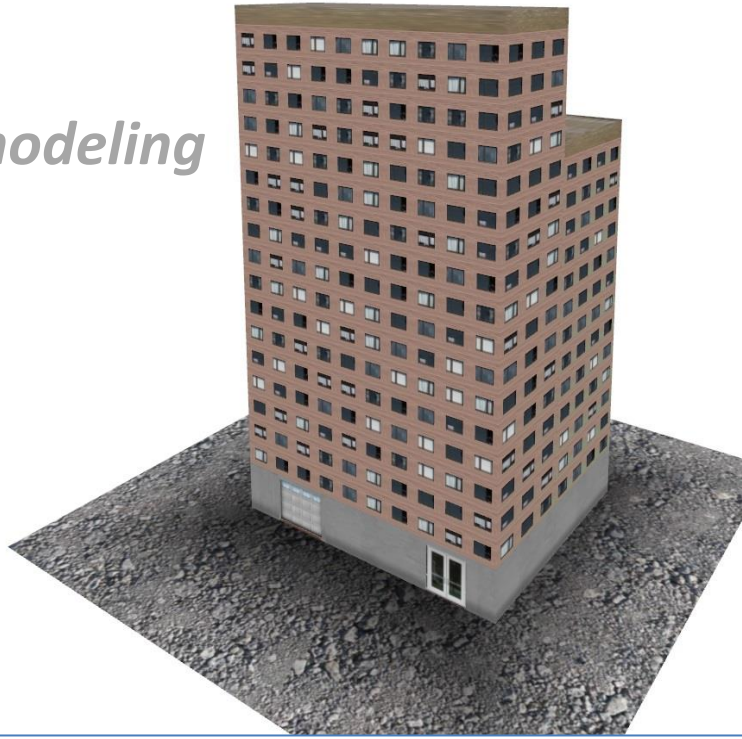
- **Naming Convention:**
  - When a Symbol is called “epsilon” it is set to inactive

Visualization :

Nicola Krombach

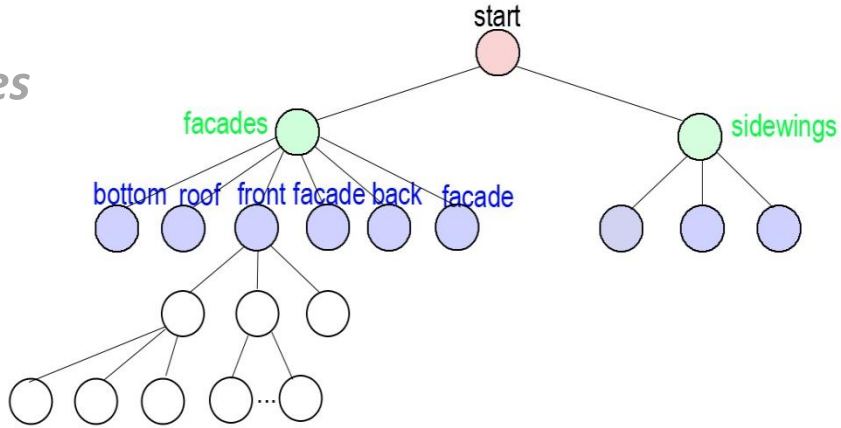
Task

- *draw derivation tree generated from modeling*
- *interactive movement in scene*
- *navigate tree history*



Derivation tree

- *Symbols as nodes*
  -> attributes like position, scale, type
- *Traverse over leaf nodes and draw them in scene at a given position and scale*
- *Apply different textures for symbol types*
  -> e.g. for windows, doors, roof, wall
- *Possibility to switch between different texture sets*

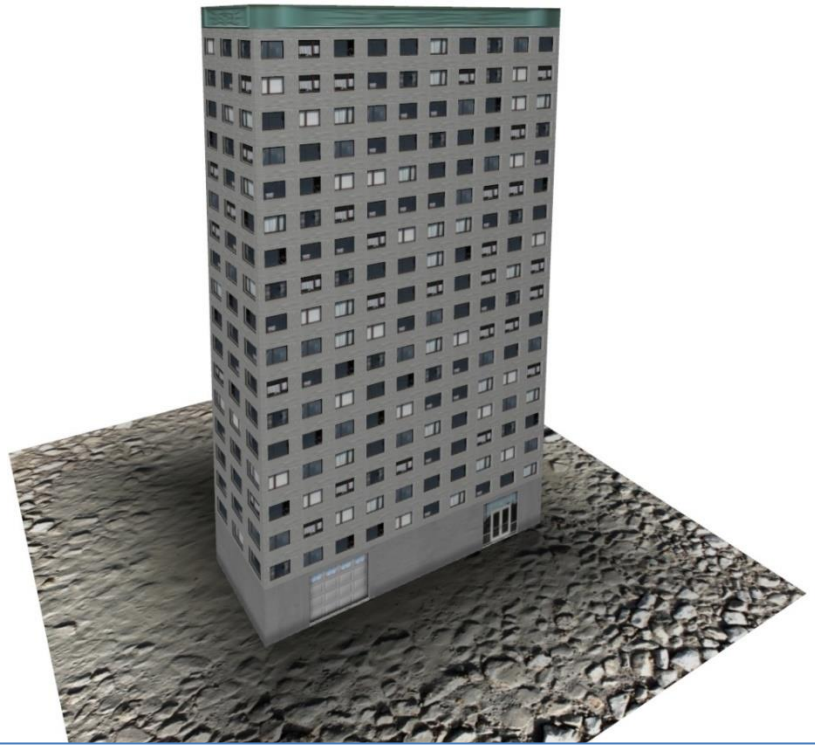


Office Building

- # Lines in the Grammar: 92
- # Nodes in the tree: 6496*
- # Leaf Nodes: 4235
- Depth of the tree: 18

*rand[a,b] function in grammar causes height and width variation

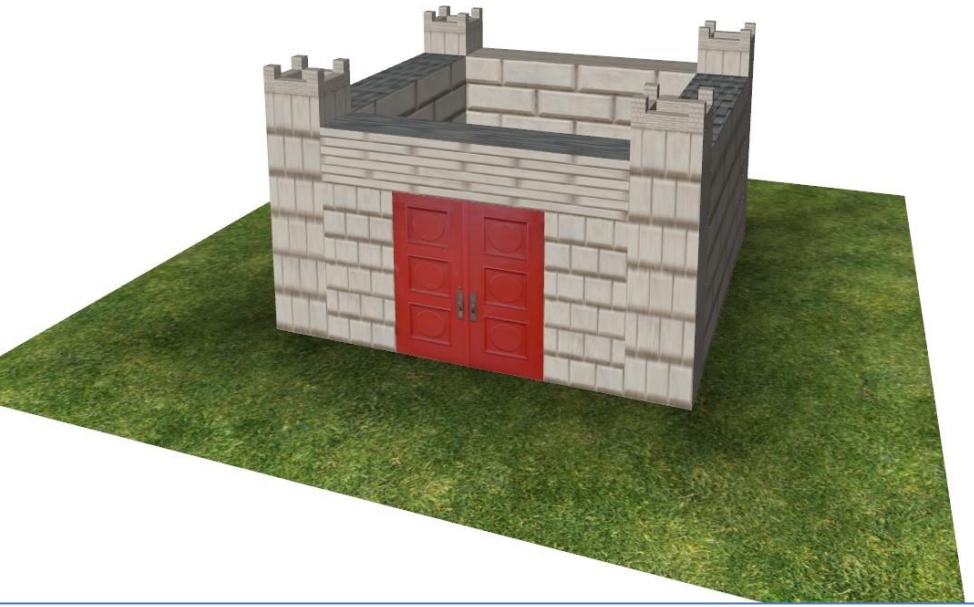
Office Building



- # Lines in the Grammar: 92
- # Nodes in the tree: 11777*
- # Leaf Nodes: 7690
- Depth of the tree: 18

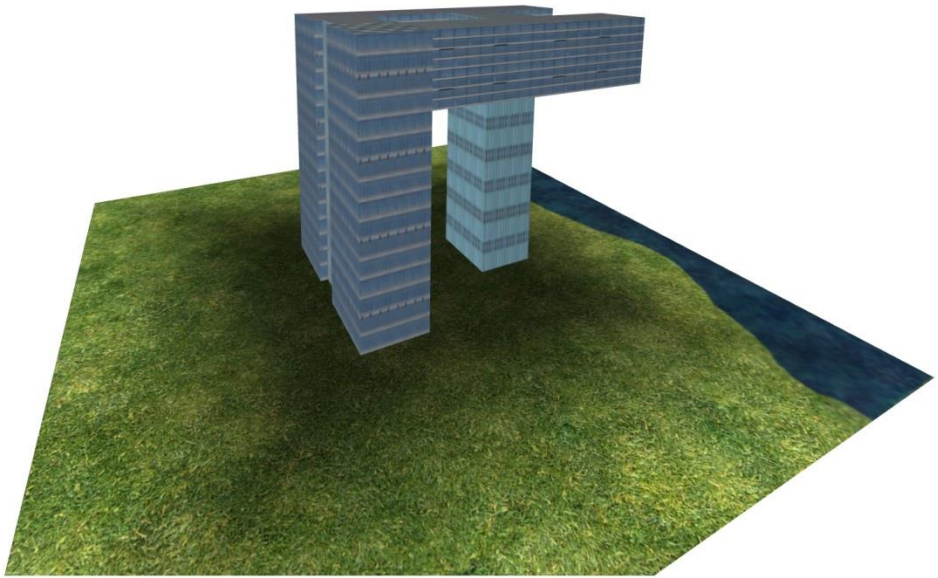
*rand[a,b] function in grammar causes height and width variation

Castle



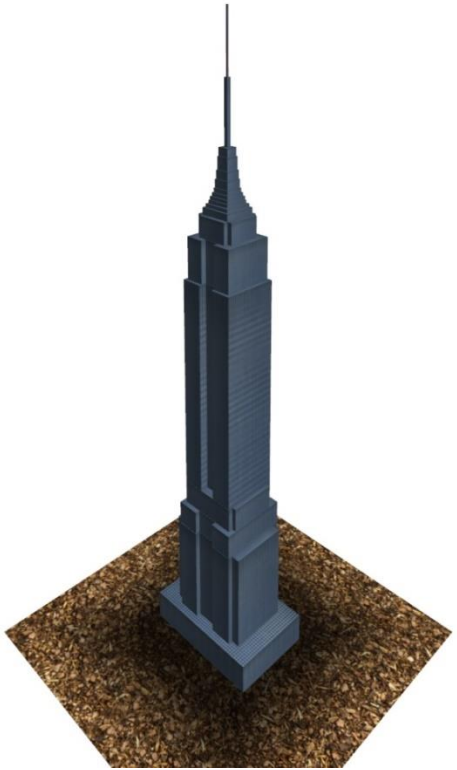
- # Lines in the Grammar: 17
- # Nodes in the tree: 111
- # Leaf Nodes: 53
- Depth of the tree: 9

„Crane“



- # Lines in the Grammar: 29
- # Nodes in the tree: 58
- # Leaf Nodes 38
- Depth of the tree: 7

Empire State Building



- # Lines in the Grammar: 120
- # Nodes in the tree: 223
- # Leaf Nodes 95
- Depth of the tree: 21

Movement in scene

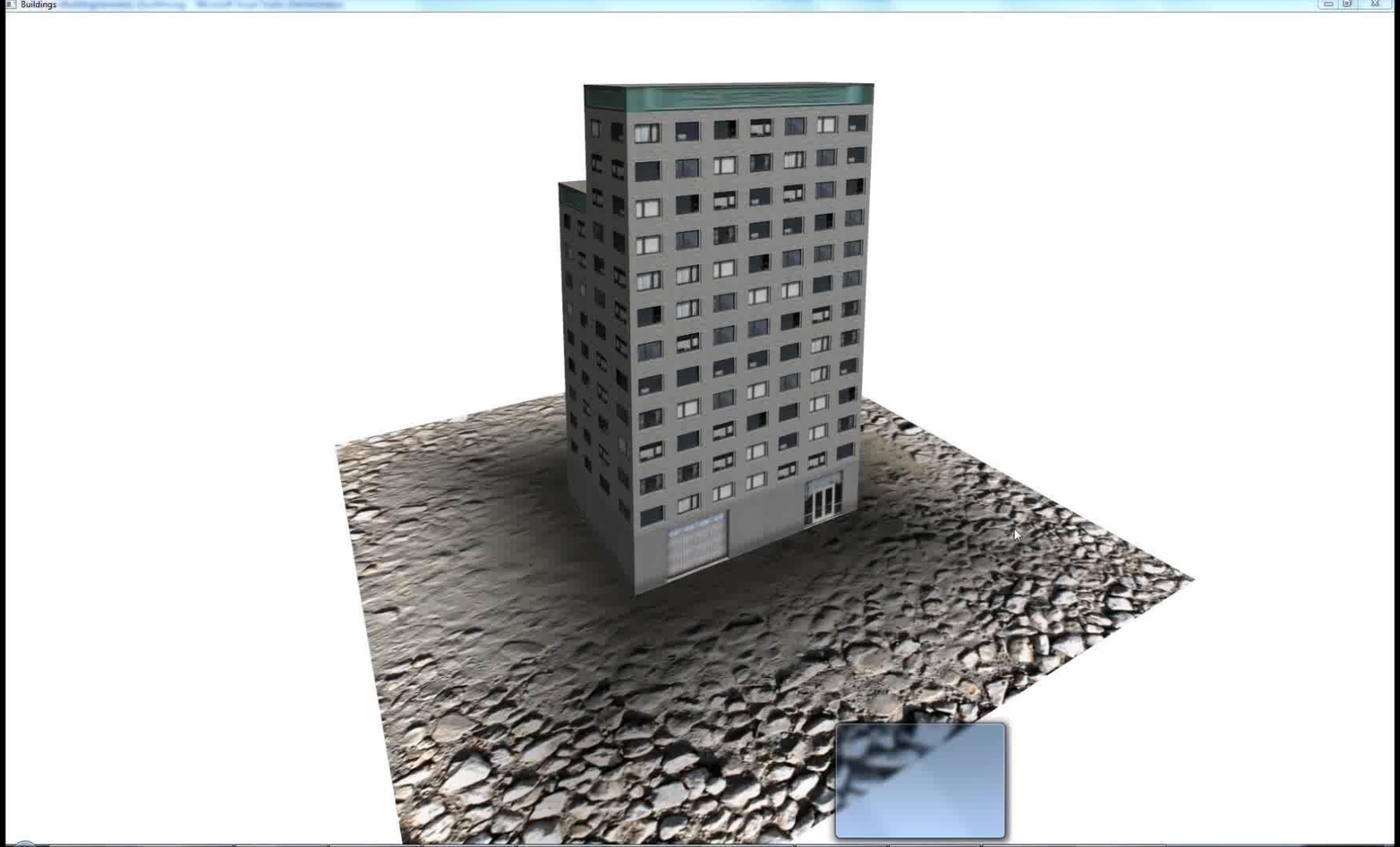
- *Interactive 3D-movement in scene:*

  zoom in/out

  move to left/right/up/down

  circle around building

  pan camera view


Buildings

Navigate tree history

- *Visualisation of bottom-up model of the complete building*
- *Go back and for in the creation of the tree from drawing a simple cube to complex building*

Navigate tree history