Agenda

• CityEngine fast forward
• CGA 101
• Python Scripting
• Outlook
CityEngine

3D procedural modeling and design solution

- **3D City Content**
  - Model cities in 3D using parametric rules

- **3D City Design**
  - Rule driven design in 3D
Procedural modeling

3D model creation using rules / algorithms

- Base geometry

Iterative refinement

iteratively refine a design by creating more and more detail
Procedural modeling vs. Manual modeling

Time reduction / cost saving
3D city content creation

procedural city modeling

Geometry

Attributes

Rules
3D city content creation

*procedural city modeling*

Rule based 3D cities
3D city design

3D procedural design

Parametric editing

Dynamic editing

Procedural reporting

Add a floor

Add a roof

Rule based design
3D city (Geo)design

Iterative analysis while designing

- Design
- Analyze
- Compare

Mass modeling

Visibility analysis

Façade design

Detailed Façades

Shadow analysis

Skyline Analysis

Design, analyze, compare
CGA 101

Simon Schubiger
CGA Shape Grammar

Scripting Geometries with Shape Grammar Rules:
- Rule-driven modification and replacement of shapes
- Iteratively evolve a design by creating more and more details
CGA: Example Building

- Example building rule file
CGA Shape Grammar

• Rules
  - A rule describes the transformation of a shape into one or more successor shapes

  - A shape consists of:
    - Symbol
    - Attributes
    - Geometry (polygonal mesh)
    - Oriented bounding box called scope (numeric attributes)
A CGA rule is an instruction to process shapes

• A and B are shapes
• A modified copy of shape A becomes shape B
• B is called a leaf shape

A --> extrude(10) B
A lot is a shape as well
- Its geometry consists only of one face
- Its symbol is displayed as Start Rule in the Inspector
- Is also called Initial Shape because it is the first shape that is processed by the CGA rule set
CGA: An actual rule

Lot $\rightarrow$ extrude(10) Mass

- The resulting geometry of leaf shapes forms the Model (geometry)
  - Models are displayed in the 3D Viewport

Lot with shape symbol **Lot**

Rule application (generation)

Resulting shape **Mass**

Displayed geometry
CGA : Shape Replacement

<table>
<thead>
<tr>
<th>Lot</th>
<th>--&gt;</th>
<th>extrude(10) Mass</th>
<th>Rule #1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mass</td>
<td>--&gt;</td>
<td>C, D</td>
<td>Rule #2</td>
</tr>
</tbody>
</table>

- Rule #2 is a matching rule for Shape Mass
- Shape Mass is replaced by shapes C and D
CGA Syntax Example

attr height = 20
const groundfloor_height = 20
Lot --> extrude(height) Mass
Mass --> comp(f) { top : Roof.
    | front : Frontfacade
    | side : Facade}
# Facade
Facade -->
    setupProjection(0, scope.xy, 1,0.5, 1)
    split(y){groundfloor_height : Groundfloor |
    ~1 : UpperFloors}
Groundfloor -->
    case scope.sx > 10 : color("#cccccc")
    else : color("#ffcccc")

- Boolean, float and string expressions
  1, 0.5, ("#cccccc"), scope.sx > 10
- CGA-specific keywords
  attr, top, front, case
- CGA operations (may have parameters)
  extrude(height), comp(f)
- Rules (may have parameters)
  Lot, Mass, Facade
- User-defined attributes, constants and functions
  height, groundfloor_height
- Comments
  #Facade, //, /* ... */
CGA Text Editor

- Opens with .cga files
- Ctrl-Space gives command completion
- Red underlines denote errors
- Yellow underlines denote warnings
- Split Screen with Visual CGA Editor (VCGA)
CGA : Create new rule

- Menu : File ➤ New… ➤ CityEngine ➤ CGA Grammar File

# my first CGA rule
Lot --> extrude(10) Mass
Now we hook up our first rule to an actual building lot

Rule File and Start Rule are shown in the Inspector

The Start Rule defines the rule that is applied first
CGA : The first building

- Menu : Initial Shape ➔ Generate

- Our first “building”!
CGA Live Mode

- Live Mode enabled
- Use transform tools to update the lot shape
- The generated model is updated on-the-fly
- Helps to analyze and test rules
CGA Attributes

- Add attribute *height* to CGA file
  - `attr height = 10`
  - Lot --> extrude(height) Mass
- Attribute appears in rule parameters in the Inspector
- Rule attributes can be externally controlled (e.g. through Inspector)
CGA Model Hierarchy Viewer

- Generated Model can be viewed in Model Hierarchy Viewer
  - Menu: Window † Show Model Hierarchy
  - Toolbar: Edit Model
- Very helpful for writing and analyzing rules
  - Displays additional info (e.g. scope)
  - Shows generated structure (tree)
CGA Shape Grammar: Simple Building
CGA: A complete ruleset

- To create a simple L-Shaped building, we need these additional CGA commands:
  - Set material color: `color("#ff7777")`
  - `split()`
  - The relative Operator
  - The floating operator
To use our building rule for more than a single building, we need more variation:

**Stochastic:**
- A --> 30% : A  
  else : B

**Conditionals provide additional control in rules**
- A -->
  case <bool_expr> : B  
  else : C

where <bool_expr> is a boolean expression
New commands we need:
- Component split: `comp()`
- `texture("file.png")`
- `setupProjection()`
- `projectUV()`
CGA : Component split

- Get face components:
  - Component split: \( \text{comp}(f) \{ \text{top} : \text{Roof} \mid \text{side} : \text{Facade} \} \)
  - Also works for edges and vertices (\( \text{comp}(e), \text{comp}(v) \))
- Different semantic selectors such as top, side, vertical, left, aslant, ...
**CGA : Applying UV’s**

- **setupProjection(uvset, axes, width, height)**
  - Sets the projection matrix for later UV projection depending on the current scope
  - UV scaling is controlled using the width and height arguments

- **projectUV(uvset)**
  - Creates texture coordinates by applying projection matrix
CGA Shape Grammar: High-rise Tower
CGA : Highrise tower

- Additional CGA commands and concepts:
  - Recursion: \( A \rightarrow B \ A \)
  - Insert external geometry: \( \rightarrow i("geometry.obj") \)
  - Functions vs. constant functions: \( const \)
  - Set scope size: \( s(x, y, z) \)
  - center shape: \( center(x) \)
  - Push and pop shapes: \[ \]
Some Building styles can be encoded in an elegant way using recursive rules, i.e. a rule calls itself.

- A → B A
- Endless recursion ‡ We need a stop condition

Lot → extrude(50) Recursion

Recursion →
  case scope.sy > 1 :
    split(y){
      '0.5 : Mass |
      ~1 : s('0.5,'1,'1) center(x) Recursion}
  else : Mass
CGA : insert geometry

• Load external geometry into the current shape
  - Asset --> i("asset.obj")
  - Arbitrary obj files can be inserted (with some limitations)
  - Insert unmodified assets (e.g. trees, pre-modeled buildings)
  - Inserted objects can be processed further with CGA rules
CGA: insert examples
CGA Shape Grammar:
Façade Modeling
General Facade Schemes

Most common subdivision scheme:

Facade 4  Floor 4  Tile 4  Wall & Window/Door
CGA : Repeat split

- Asterix marks a repeating split.
  - \texttt{split(y)\{\sim\text{width} : A\}*}
  - Floating operator \(\sim\) ensures fitting sizes

- Normal and repeating splits can be nested:
  - \texttt{split(y)\{\texttt{groundfloorheight} : \texttt{Groundfloor}\
  | \{\sim\texttt{floorheight} : \texttt{Floors}\}* \}}
CGA: Rhythm split

- Nesting normal and repeating splits with rhythms

\[ \text{split}(x) \{
\begin{align*}
\text{widthA} & : \text{TileA} \\
\{ \sim \text{widthB} & : \text{TileB}\}\} \\
\text{widthA} & : \text{TileA}
\end{align*}
\]

\[ \text{split}(x) \{
\begin{align*}
\{ \text{widthA} & : \text{TileA} \\
\sim \text{widthB} & : \text{TileB}\}\} \\
\text{widthA} & : \text{TileA}
\end{align*}
\]
CGA: Facade assets

- Inserting a window asset
  - LOD: Texture or geometry asset
CGA : function, const and attr

- `randomHeight = rand(10,20)`
  - Functions are always evaluated
  - Do not appear in inspector, may have parameters

- `const randomHeight = rand(10,20)`
  - Consts are evaluated once per generation and then stays constant
  - Do not appear in inspector

- `attr randomHeight = rand(10,20)`
  - Attrs are like consts
  - Attrs can be set externally (e.g. through Inspector)
  - Attrs define the “interface” to a rule file
CGA Shape Grammar: Residential Building
CGA: Residential Building

- Additional CGA commands and concepts:
  - Roof command: `roofHip()`, `roofShed()`
  - Find inner Rectangle: `innerRect`
  - Placing assets
  - Simple LOD
CGA: Roof commands

- Special commands create roof shapes:
  - `roofGable()`
  - `roofHip()`
  - `roofPyramid()`
  - `roofShed()`

- With additional settings such as angle and overhang
CGA : innerRect

- innerRect
- Transforms shape into a rectangle fitting into current geometry
CGA : LOD

- Simple LOD approach that loads different assets
- Controllable with CGA attributes
  - Can be manually adjusted or controlled globally with maps

```python
attr LOD = 1

case LOD > 0 : "hires_asset.obj"
else : "billboard_asset.obj"
```
Python Scripting

Simon Schubiger
Python Scripting

- Python Console
- Python Editor

- Extensive command set
  see CityEngine Help for reference

- Use your own Python modules
def exportToObj(shapes, exportName):
    # create new export settings class, define export format
    objExportSettings = OBJExportModelSettings()
    # specify export settings
    objExportSettings.setGeneralName(exportName)
    # do the export
    ce.export(shapes, objExportSettings)

if __name__ == '__main__':
    exportToObj("pythonExported")
def exportMulti(shapes, exportName):
    for i in range(10, 20):
        # set value of height attribute
        ce.setAttribute(shape, "/ce/rule/height", i)
        # call export function
        exportToObj(shape, exportName + str(i))

if __name__ == '__main__':
    exportMulti("pythonExported")
Python: Script Based Export

- Python scripts can run parallel to the export
- Can process arbitrary report data via callback functions
- Powerful mechanism in combination with CGA `report()`

```python
# Called before the export starts.
def initExport():

# Called for each initial shape before generation.
def initModel():

# Called for each initial shape after generation.
def finishModel():

# Called after all initial shaped are generated.
def finishExport():
```
def finishModel(exportContextUUID, shapeUUID, modelUUID):
    shape = Shape(shapeUUID)
    model = Model(modelUUID)

    # get report variable 'LotArea' of generated model
    reports = model.getReports()
    shapeName = ce.getName(shape)
    lotAreaSum = sum(reports['LotArea'])

    # storing data to global variable
    global REPORT
    REPORT += "%s,%f\n" (shapeName, lotAreaSum)

def finishExport(exportContextUUID):
    # write collected report data to file
    global REPORT
    filename = ce.toFSPath("data/report_LotAreas.txt")
    file = open(filename, "w")
    file.write(REPORT)
    file.close()
Python: Write report data to file 2

- Start the script based exporter with python script containing the callback functions

- Collected report data is written to file `data/report_LotAreas.txt`

<table>
<thead>
<tr>
<th>Lot</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2551.697510</td>
</tr>
<tr>
<td>1</td>
<td>1365.432495</td>
</tr>
<tr>
<td>2</td>
<td>2573.790283</td>
</tr>
<tr>
<td>3</td>
<td>2615.475098</td>
</tr>
<tr>
<td>4</td>
<td>2815.327881</td>
</tr>
<tr>
<td>5</td>
<td>2069.638184</td>
</tr>
<tr>
<td>6</td>
<td>2164.343994</td>
</tr>
<tr>
<td>7</td>
<td>1753.116943</td>
</tr>
</tbody>
</table>
Procedural Runtime

Simon Schubiger
Procedural Runtime Use Cases

- **Entertainment Pipelines**
  - DLL, no ArcGIS e.g. proprietary exporters or rendering with generation on demand

- **ArcGIS Desktop**
  - GP Tool e.g. attribute-driven building geometries or parametric power poles in ArcScene

- **ArcGIS Server**
  - GP Service for 3D maps, 3D analytics
Procedural Runtime Architecture

CGA compilation

Client App

Resources

Procedural Runtime

CGA

cgac

CGB

Shape Processing Unit

Adaptors

Codecs
Procedural Runtime Architecture

Client Binding
- GP Tool / COM
- ArcObjects / COM
- JNI / C++
- JavaScript / REST

Adaptors
- ArcObjects
- JNI / C++
- File System
- SQL

Codecs
- MultiPatch
- Collada
- OBJ
- VOB
- FBX
- MI
- OpenGL
- WebGL
...

Shape Processing Unit

Procedural Runtime
There is more about 3D Cities

Creating, managing and utilizing a 3D Virtual City

- Tamrat Belayneh, Eric Wittner, Nathan Shepard

Smoketree A-E