Developing with Esri CityEngine

Gert van Maren, Simon Schubiger
Esri R&D Center Zurich
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

- Procedural rules

Base geometry → Final 3D model

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

![Add a floor](image1.png)

![Add a roof](image2.png)

Dynamic editing

Procedural reporting

![Dynamic editing](image3.png)

Rule based design

![Procedural reporting](image4.png)
3D city (Geo)design

*Iterative analysis while designing*

- Design
- Analyze
- Compare

- Mass modeling
- Visibility analysis
- Shadow analysis
- Façade design
- Detailed Façades
- Skyline Analysis

Design, analyze, compare
CGA 101
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 CGA rule can modify shapes

A and B are shapes

A modified copy of shape A becomes shape B

B is called a leaf shape
CGA : Lot (Feature) = Shape

- 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 $\textbf{Lot}$

Rule application (generation)

Resulting shape $\textbf{Mass}$

Displayed geometry
CGA: Shape Replacement

<table>
<thead>
<tr>
<th>Lot</th>
<th>extrude(10) Mass</th>
<th>Rule #1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mass</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

```plaintext
# my first CGA rule
Lot --> extrude(10) Mass
```
CGA: Assign rule file to lot

- 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”!

- Thanks to the rules, it updates automatically
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 : Inspect 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: \texttt{color("#ff7777")}
  - \texttt{split()}
  - The relative Operator \textbackslash\textbackslash
  - The floating operator \texttt{~}
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
CGA: Texturing

- 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 : 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
General Facade Schemes

Most common subdivision scheme:

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

- Asterix marks a repeating split.
  - `split(y){~width : A}*`
  - Floating operator `~` ensures fitting sizes

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

- Nesting normal and repeating splits 

\[ \mathcal{B} \ A \ B^* \ A \]

\[
\text{split}(x) \{ \text{widthA} : \text{TileA} \\
| \{ \sim \text{widthB} : \text{TileB}\}^* \\
| \text{widthA} : \text{TileA} \}
\]

\[ \mathcal{B} \{ A \ B \}^* A \]

\[
\text{split}(x) \{ \{ \text{widthA} : \text{TileA} \\
| \sim \text{widthB} : \text{TileB} \}^* \\
| \text{widthA} : \text{TileA} \}
\]
CGA : Facade assets

- Inserting a window asset
  - LOD : Texture or geometry asset
CGA Shape Grammar: Suburban Building
CGA: Suburban 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

attr LOD = 1
case LOD > 0 : "hires_asset.obj"
else : "lowres_asset.obj"
Where to go from here?

- Export to File
- Share on AGOL (3WS)
- Share as a Rule Package
Export to File / Share as AGOL item

Import
- DXF
- DAE
- OBJ
- GDB
- SHP
- OSM
- DDS
- GIF
- JPG
- PSP
- SGI
- BMP
- TGA

Export
- DAE
- FBX
- GDB
- 3DS
- RIB
- OBJ
- KMZ
- VOB
- MI
- 3WS

CityEngine

3D Web Viewer
Share as Rule Package

ArcScene 10.2
Python Scripting
Python Scripting

- Python Console
- Python Editor

- Extensive command set
  see CityEngine Help for reference

- Use your own Python modules
Python: Export via script

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():
```
```python
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 1

scripts/reportExport_1.py
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>Lot_3</td>
<td>2615.475098</td>
</tr>
<tr>
<td>Lot_2</td>
<td>2573.790283</td>
</tr>
<tr>
<td>Lot_7</td>
<td>1753.116943</td>
</tr>
<tr>
<td>Lot_4</td>
<td>2815.327881</td>
</tr>
<tr>
<td>Lot_1</td>
<td>1365.432495</td>
</tr>
<tr>
<td>Lot_6</td>
<td>2164.343994</td>
</tr>
<tr>
<td>Lot_5</td>
<td>2069.638184</td>
</tr>
<tr>
<td>Lot_0</td>
<td>2551.697510</td>
</tr>
</tbody>
</table>
There is more about 3D Cities

Creating, Managing, and Utilizing a 3D Virtual City

Craig McCabe, Eric Wittner, Thorsten Reitz

Wednesday, 10:30am-11:30am,

Mesquite GH

Thanks for filling out the survey

Offering ID: 232