

Esri Developer Summit

March 26–29, 2012 | Palm Springs, California

esri.com/events/devsummit



Developing with Esri CityEngine

Gert van Maren, Nathan Shephard, Simon Schubiger

```
esri.symbol.SimpleLineSymbol({
  color: [0, 0, 255],
  width: 2
});
feature.setSymbol(new esri.symbol.SimpleLineSymbol({
  color: [0, 0, 255],
  width: 2
}));
```

Agenda

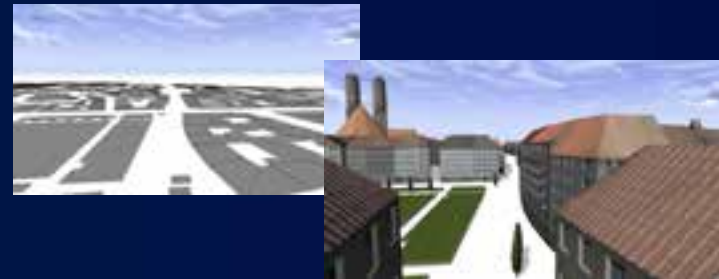
- **CityEngine fast forward**
- **CGA 101**
- **Python Scripting**
- **Outlook**

CityEngine

3D procedural modeling and design solution

- 3D City Content

- Model cities in 3D using parametric rules



Geometry + Attributes + Rules

- 3D City Design

- Rule driven design in 3D



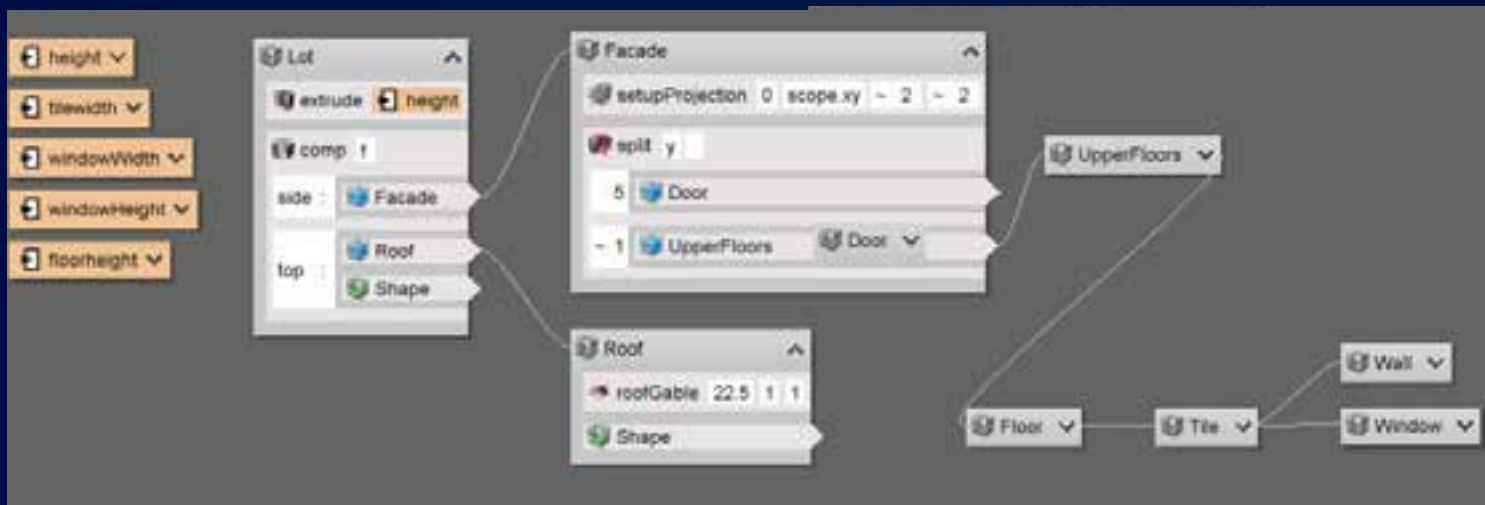
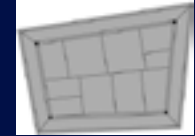
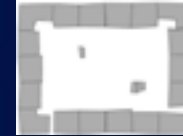
Dynamic + Parametric editing

Rule based 3D content and design

Procedural modeling

3D model creation using rules / algorithms

- Base geometry



Base geometry

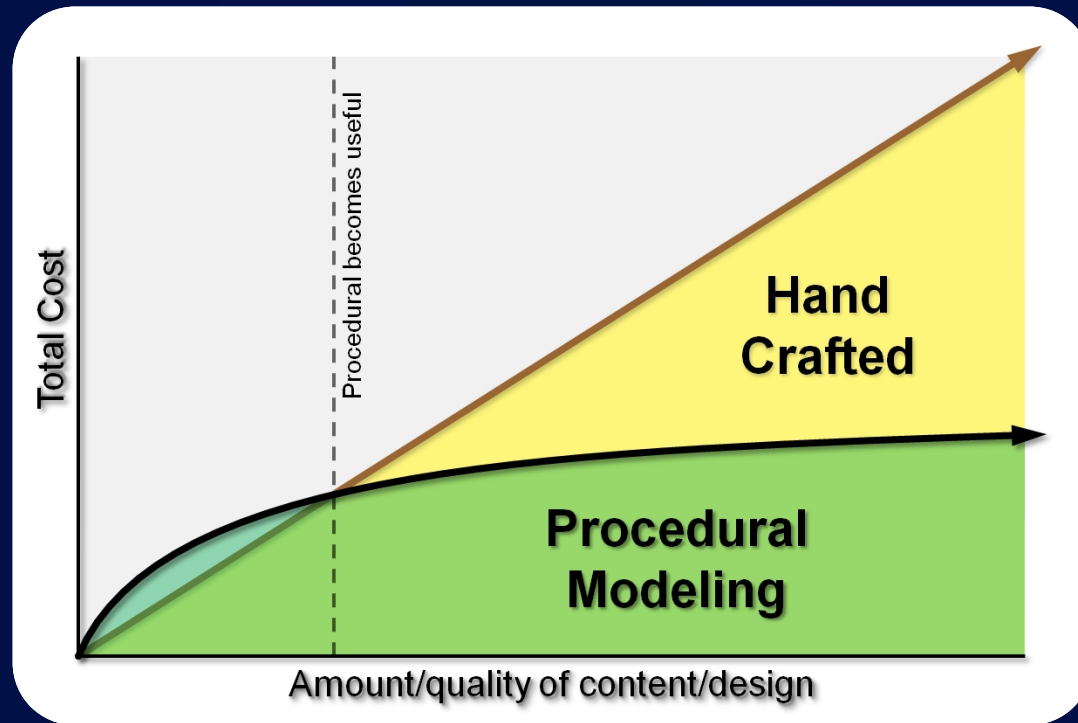


Final 3D model

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

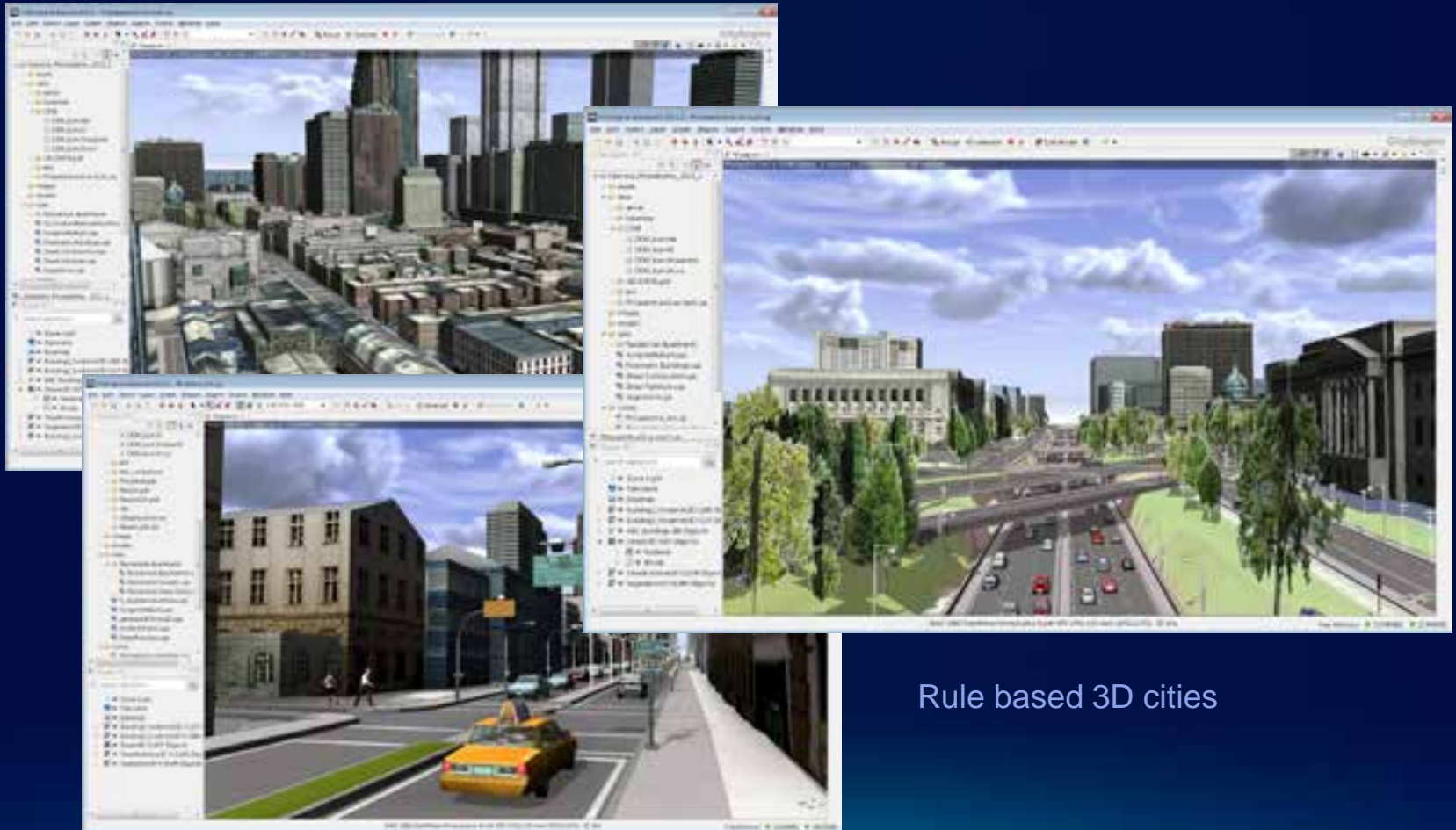


Rules



3D city content creation

procedural city modeling



Rule based 3D cities

3D city design

3D procedural design

Parametric editing



Add a floor



Add a roof

Dynamic editing



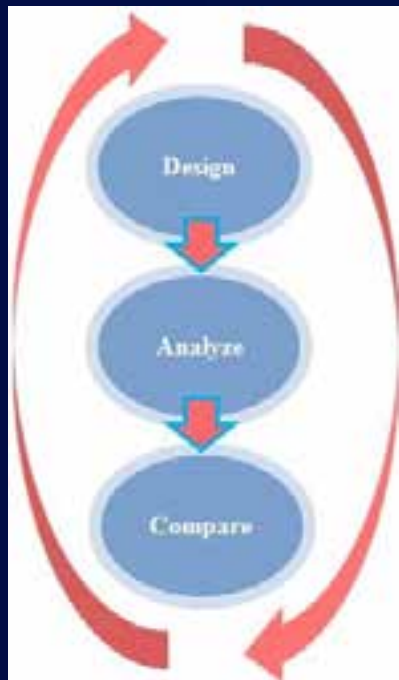
Procedural reporting



Rule based design

3D city (Geo)design

Iterative analysis while designing



Mass modeling



Visibility analysis



Shadow analysis



Façade design



Detailed Façades



Skyline Analysis



Design, analyze, compare

CGA 101

Simon Schubiger



```
esri.symbol.SimpleLineSymbol({
  color: [0, 0, 255],
  width: 2
});
feature.setSymbol(new esri.symbol.SimpleLineSymbol({
  color: [0, 0, 255],
  width: 2
}));
else if(f == 1) {
  var polysymbolGreen = new esri.symbol.PolySymbolGreen({
    color: [0, 0, 0.5]
  });
  polysymbolGreen.setOutline(new esri.symbol.SimpleLineSymbol({
    color: [0, 0, 0.5],
    width: 1
  }));
  feature.setSymbol(polysymbolGreen);
} else if(f == 2) {
  polysymbolBlue = new esri.symbol.PolySymbolBlue({
    color: [0, 0, 255]
  });
  polysymbolBlue.setOutline(new esri.symbol.SimpleLineSymbol({
    color: [0, 0, 255],
    width: 1
  }));
  feature.setSymbol(polysymbolBlue);
}
```

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

A screenshot of a CGA editor window. The top part shows a code editor with the following code:

```
13  
14 attr windowWidth = 2.2  
15 attr windowHeight = 2.8  
16 attr floorheight = 4.0  
17  
18  
19 Lot -->  
20 extrude height: comp(0) | side : Facade | top(2): Shape |  
21  
22 Facade -->  
23 setupProjection(0, wopee.xy, -2, -2)  
24 wplst(y) { $ : Door | -1: UpperFloors }
```

The bottom part shows a visual graph with nodes for 'Lot', 'Facade', 'Door', 'UpperFloors', 'Flow', and 'Tile'. On the left, there are sliders for 'height', 'tilewidth', 'windowWidth', 'windowHeight', and 'floorheight'. The 'Lot' node is expanded to show 'extrude height', 'comp 1', 'side Facade', and 'top Shape'.An Inspector window showing the following details:

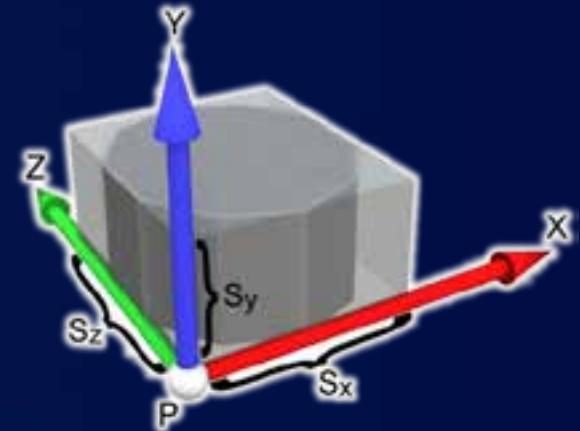
- Shape: Shape_2
- Rule File: solution/simple_building.cga
- Start Rule: Lot
- Random Seed: -93640
- Object Attributes: (collapsed)
- Rule Parameters:

Name	Source	Value
floorheight	Rule 4	0 [slider] 10
height	Rule 18.9770...	0 [slider] 100
tilewidth	Rule 3	0 [slider] 10
windowHeight	Rule 2.800000	0 [slider] 10
windowWidth	Rule 2.200000	0 [slider] 10
- Reports: (collapsed)

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)**



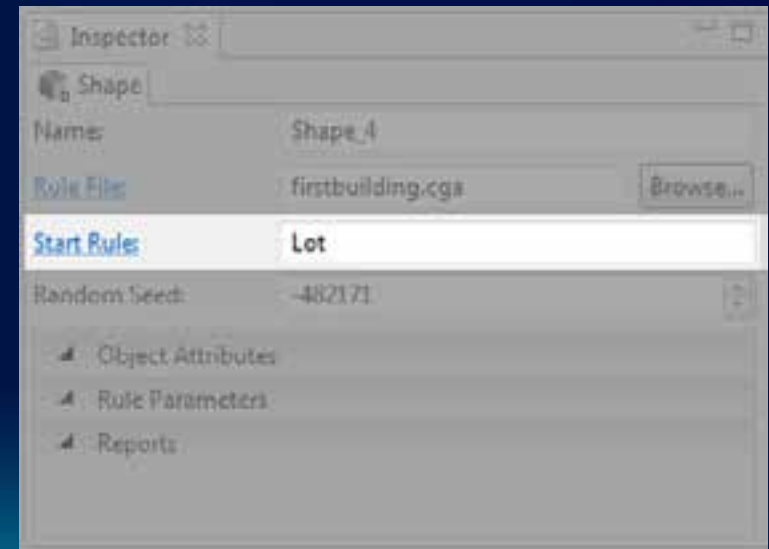
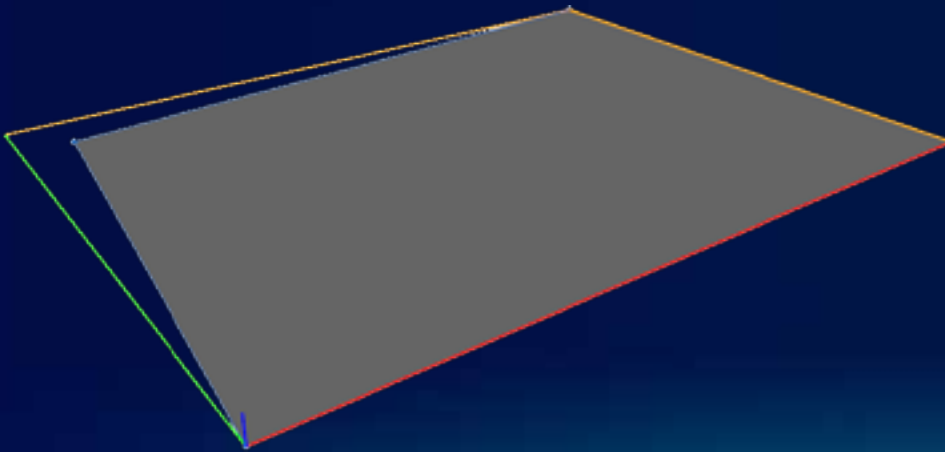
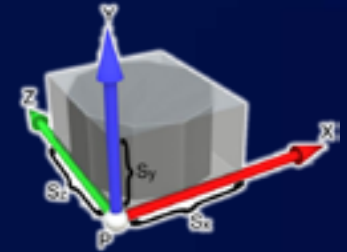
CGA : What is a rule

```
A --> extrude(10) B
```

- A CGA rule is an instruction to process shapes
 - à CGA rules 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 = Shape

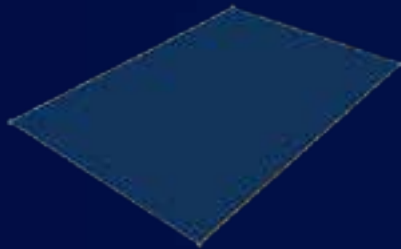
- A lot is a shape as well
 - Its geometry consists only of one face
 - Its symbol is displayed as Start Rule in the Inspector
 - Is also called Initial Shape because it is the first shape that is processed by the CGA rule set



CGA : An actual rule

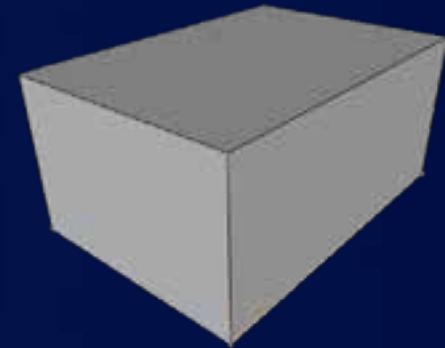
```
Lot --> extrude(10) Mass
```

- 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

Lot	-->	extrude(10)	Mass	<i>Rule #1</i>
Mass	-->	C D		<i>Rule #2</i>

- 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)



```
13f our first rule
14A --> extru
15
16 extrude(float a) -
17 extrude(x|y|z|world.x|world.y|world.z, float b) -
18
```

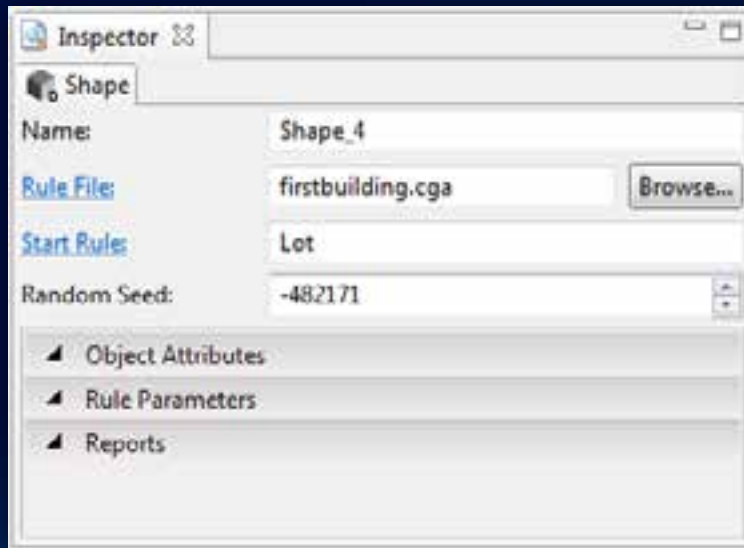
CGA : Create new rule

- Menu : File à New... à CityEngine à CGA Grammar File

```
# 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

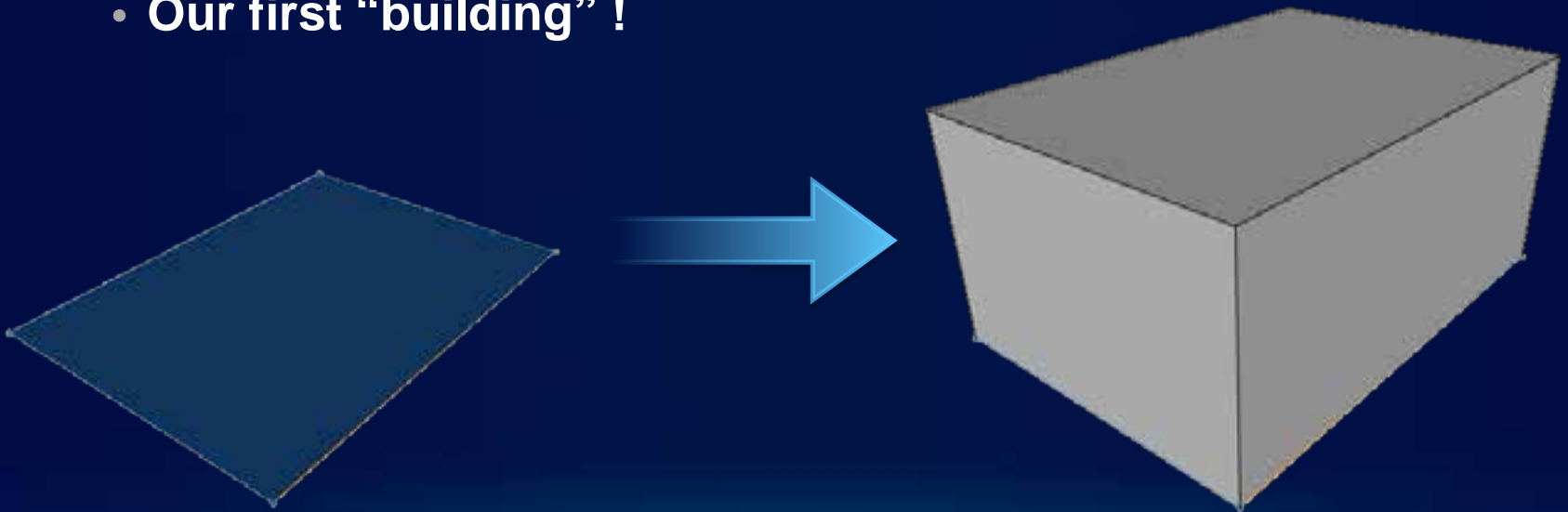


CGA : The first building

- Menu : Initial Shape à Generate



- Our first “building” !

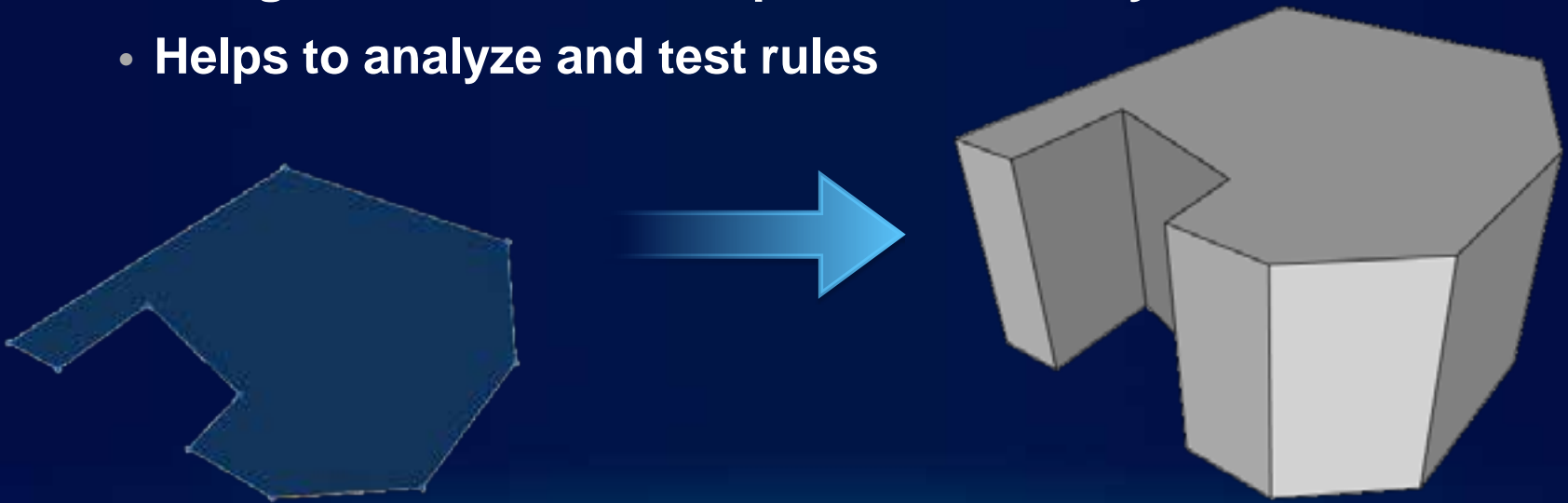


CGA Live Mode

- Live Mode enabled

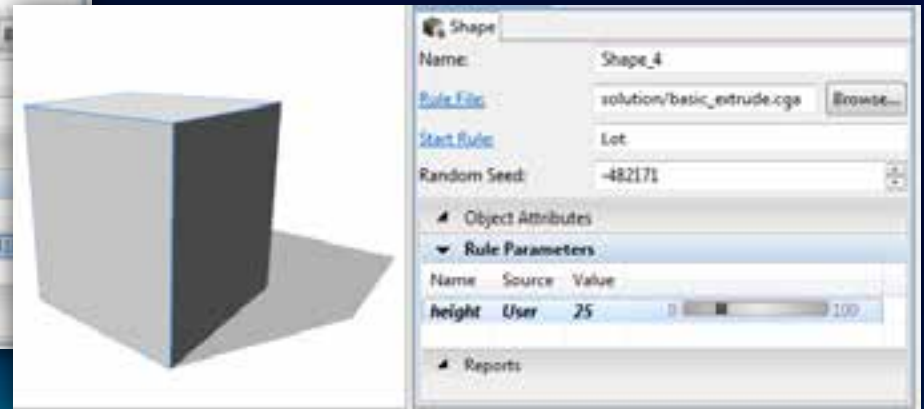
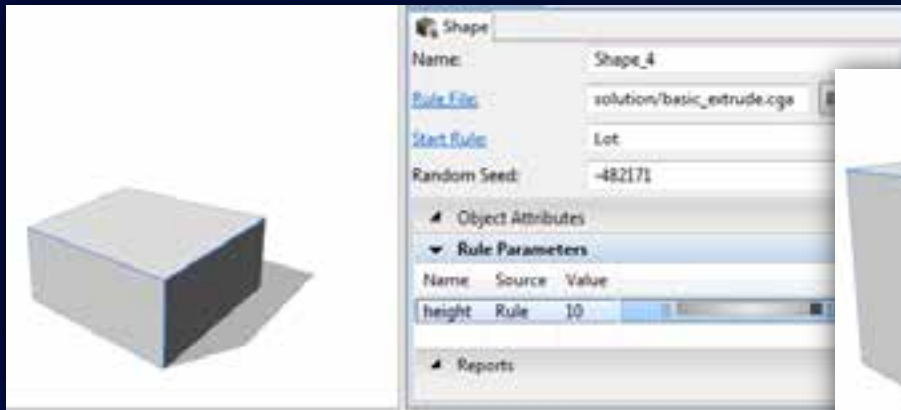


- Use transform tools to update the lot shape
- The generated model is updated on-the-fly
- Helps to analyze and test rules



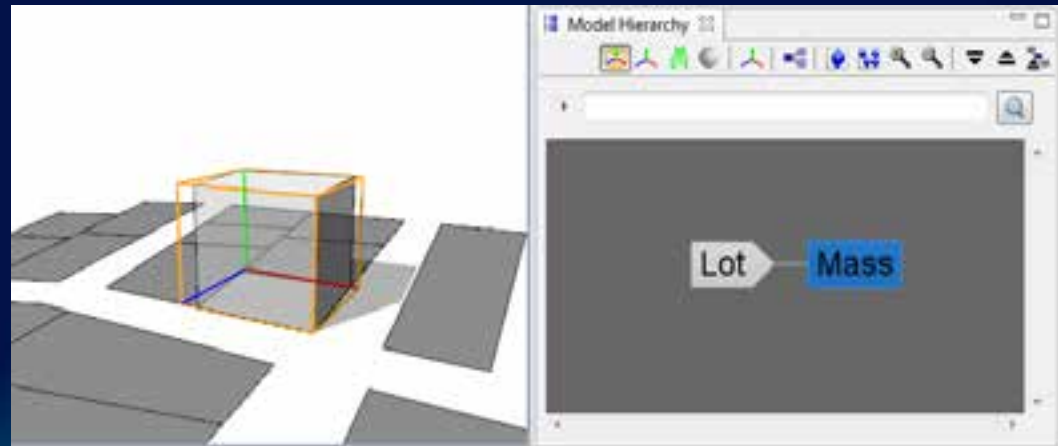
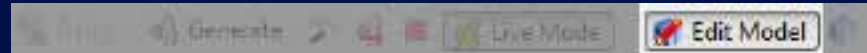
CGA Attributes

- **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 à Show Model Hierarchy**
 - **Toolbar : Edit Model**
- **Very helpful for writing and analyzing rules**
 - **Displays additional info (e.g. scope)**
 - **Shows generated structure (tree)**

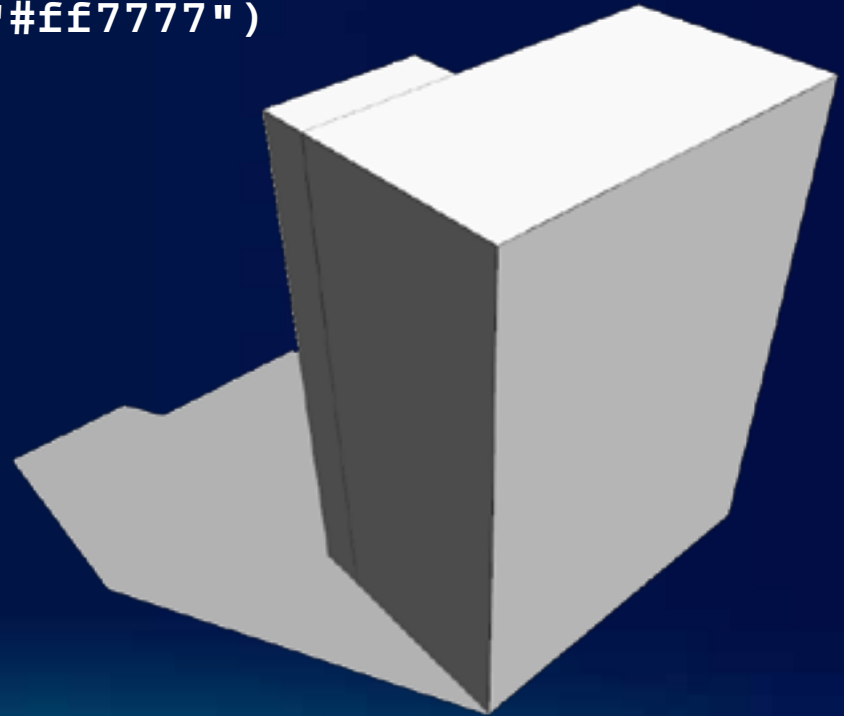


CGA Shape Grammar: Simple Building

```
esri.symbol.SimpleLineSymbol({color:[0,0,0],width:2});
new dojo.Color([0,0,0]);
feature.setSymbol(new esri.symbol.SimpleLineSymbol({color:[0,0,0],width:2}));
} else if(f == 1) {
var polysymbolGreen = new esri.symbol.SimplePolygonSymbol({color:[0,0,0.5]});
polysymbolGreen.setOutline(new esri.symbol.SimpleLineSymbol({color:[0,0,0],width:2}));
feature.setSymbol(polysymbolGreen);
} else if(f == 2) {
var polysymbolBlue = new esri.symbol.SimplePolygonSymbol({color:[0,0,1]});
polysymbolBlue.setOutline(new esri.symbol.SimpleLineSymbol({color:[0,0,0],width:2}));
feature.setSymbol(polysymbolBlue);
}
```

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 

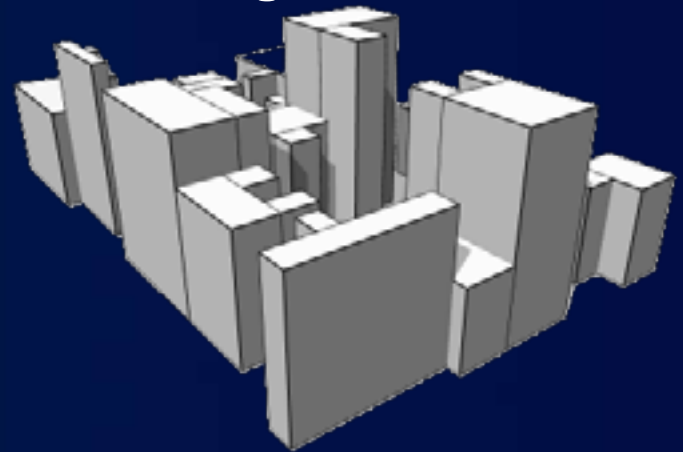


CGA : Variation and Conditions

- To use our building rule for more than a single building, we need more variation:

Stochastic:

```
- A -->  
  30%   : A  
  else  : B
```



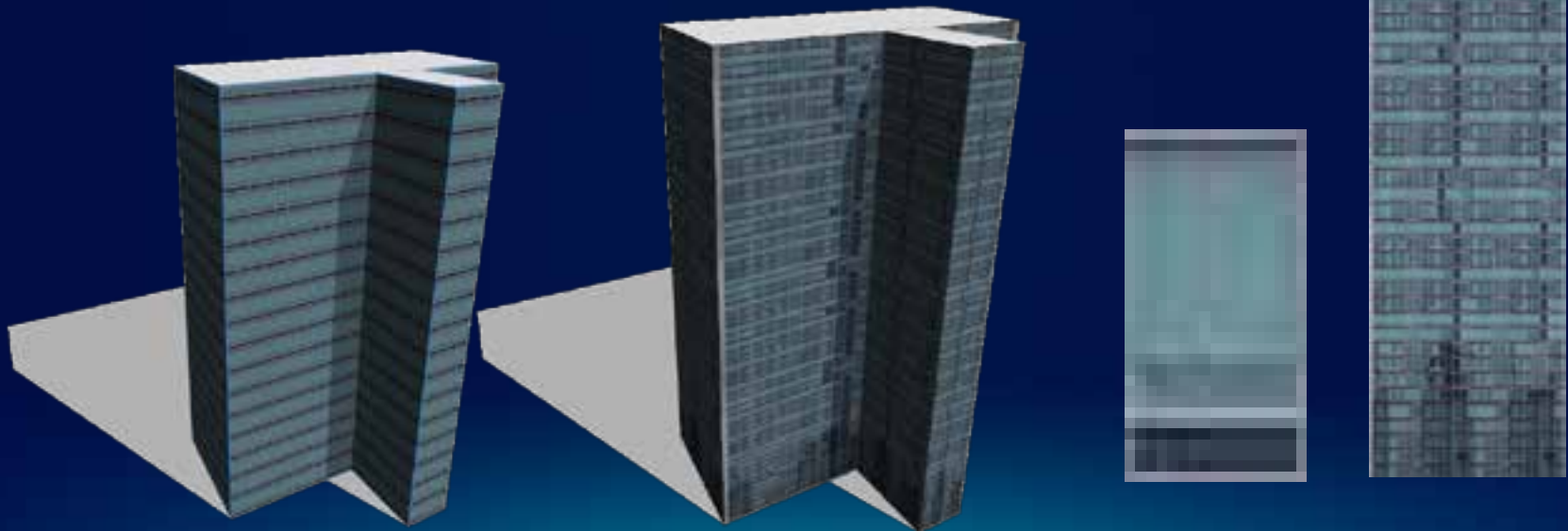
- Conditionals provide additional control in rules

```
- A -->  
  case <bool_expr> : B  
  else              : C
```

- where <bool_expr> is a boolean expression

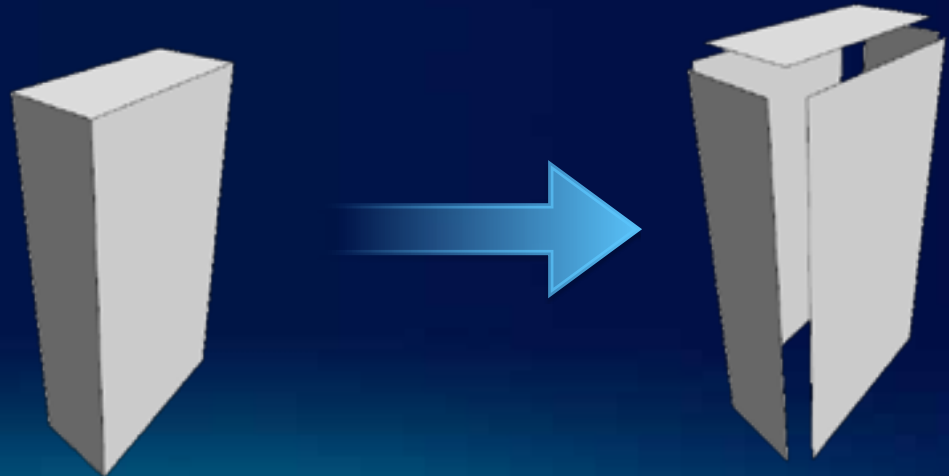
CGA : Texturing

- New commands we need:
 - Component split : `comp()`
 - `texture("file.png")`
 - `setupProjection()`
 - `projectUV()`



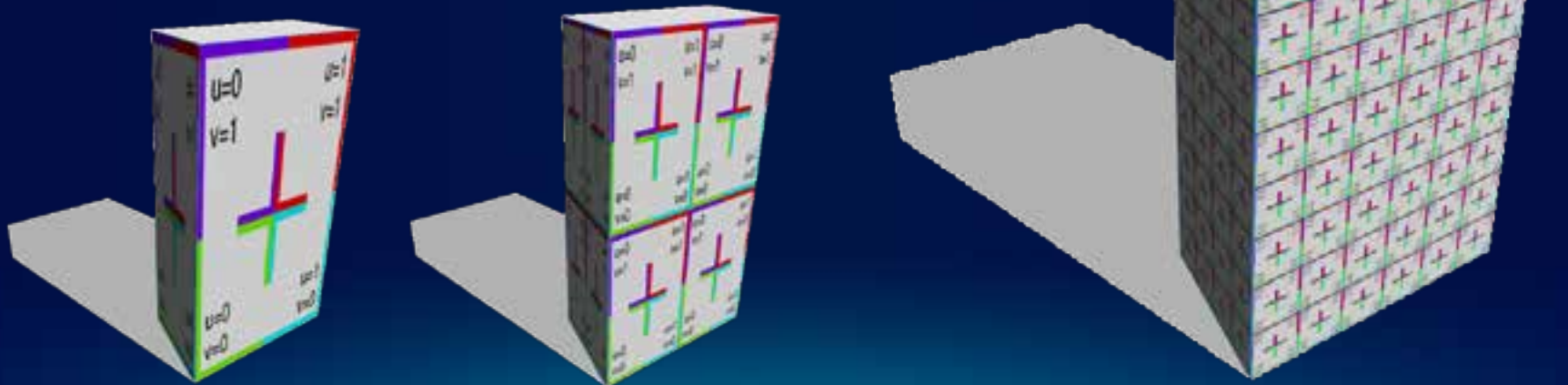
CGA : Component split

- **Get face components:**
 - **Component split** : `comp(f) { top : Roof | side :`
`Facade }`
 - **Also works for edges and vertices** (`comp(e)` , `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

```
esri.symbol.SimpleLineSymbol({color:[0,0,0],width:2});
new dojo.Color([0,0,0]);
feature.setSymbol(new esri.symbol.SimpleLineSymbol({color:[0,0,0],width:2}));
} else if(f == 1) {
var polysymbolGreen = new esri.symbol.PolySymbol({color:[0,0,0],width:2});
polysymbolGreen.setOutline(new esri.symbol.SimpleLineSymbol({color:[0,0,0],width:2}));
feature.setSymbol(polysymbolGreen);
} else if(f == 2) {
var polysymbolBlue = new esri.symbol.PolySymbol({color:[0,0,0],width:2});
polysymbolBlue.setOutline(new esri.symbol.SimpleLineSymbol({color:[0,0,0],width:2}));
feature.setSymbol(polysymbolBlue);
}
```


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 : []



CGA : Highrise tower

- Some Building styles can be encoded in an elegant way using recursive rules, i.e. a rule calls itself
 - $A \rightarrow B A$
 - Endless recursion \rightarrow We need a stop condition

```
Lot --> extrude(50) Recursion
```

```
Recursion -->
```

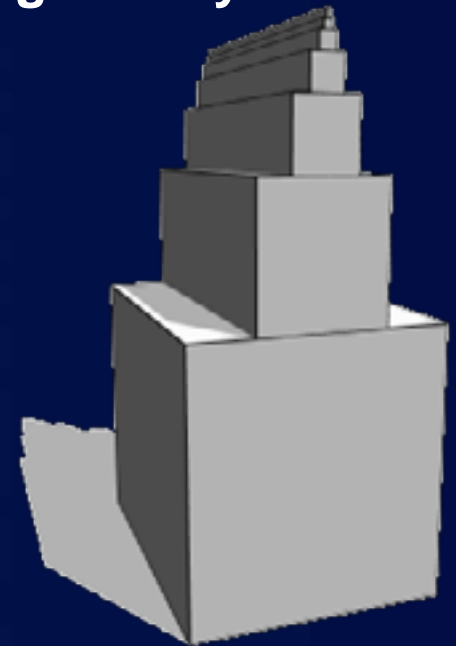
```
  case scope.sy > 1 :
```

```
    split(y){
```

```
      '0.5 : Mass |
```

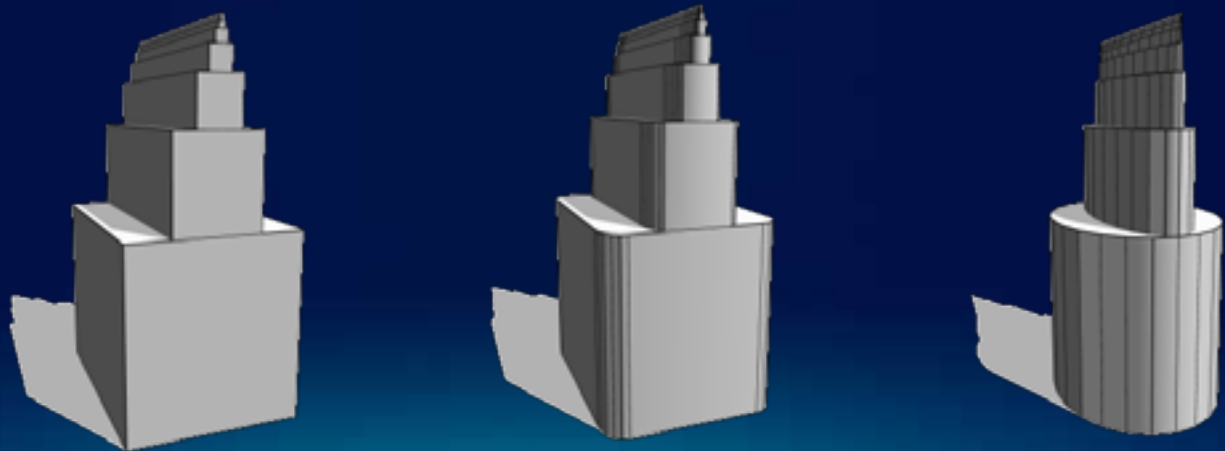
```
      ~1 : s('0.5','1','1) center(x) Recursion}
```

```
  else : Mass
```

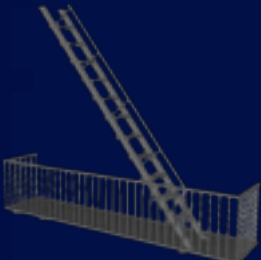
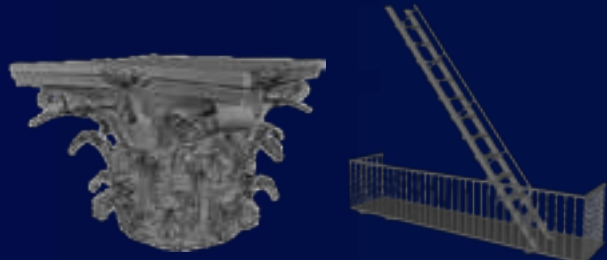


CGA : insert geometry

- 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

```
esri.symbol.SimpleLineSymbol({color:[0,0,0],width:2})
new dojo.Color([0,0,0]).toHex();
feature.setSymbol(symbol);
} else if(f == 1) {
var polysymbolGreen = new esri.symbol.SimpleLineSymbol({color:[0,0,0.5],width:2});
polysymbolGreen.setOutlineColor([0,0,0.5], 1);
feature.setSymbol(polysymbolGreen);
} else if(f == 2) {
var polysymbolBlue = new esri.symbol.SimpleLineSymbol({color:[0,0,1],width:2});
polysymbolBlue.setOutlineColor([0,0,1], 1);
feature.setSymbol(polysymbolBlue);
}
```

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}* }`

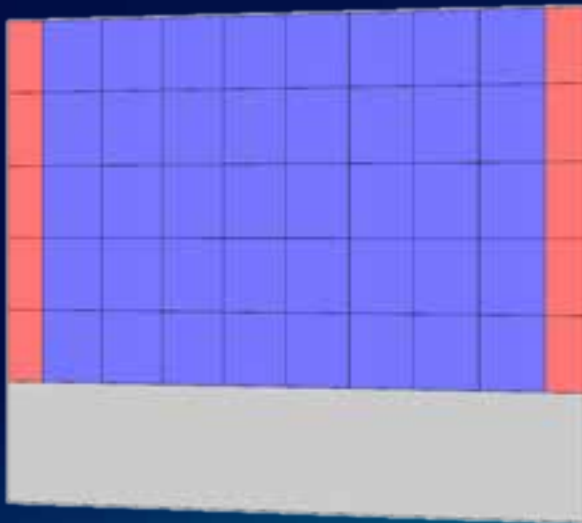


CGA : Rhythm split

- Nesting normal and repeating splits à rhythms

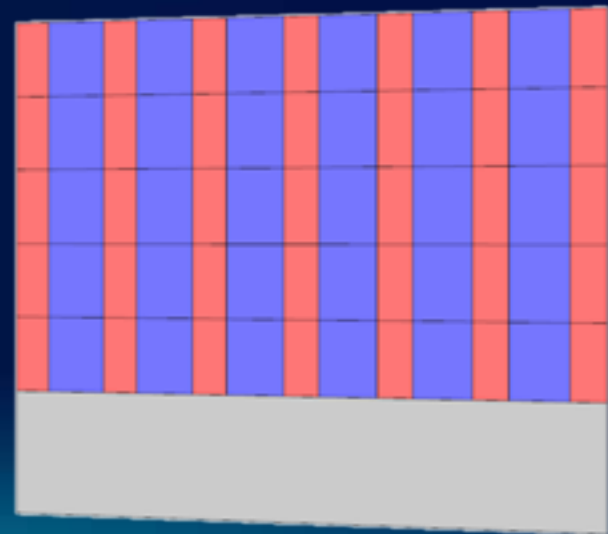
§ A B* A

```
split(x) { widthA: TileA  
  | {~ widthB : TileB}*  
  | widthA : TileA }
```



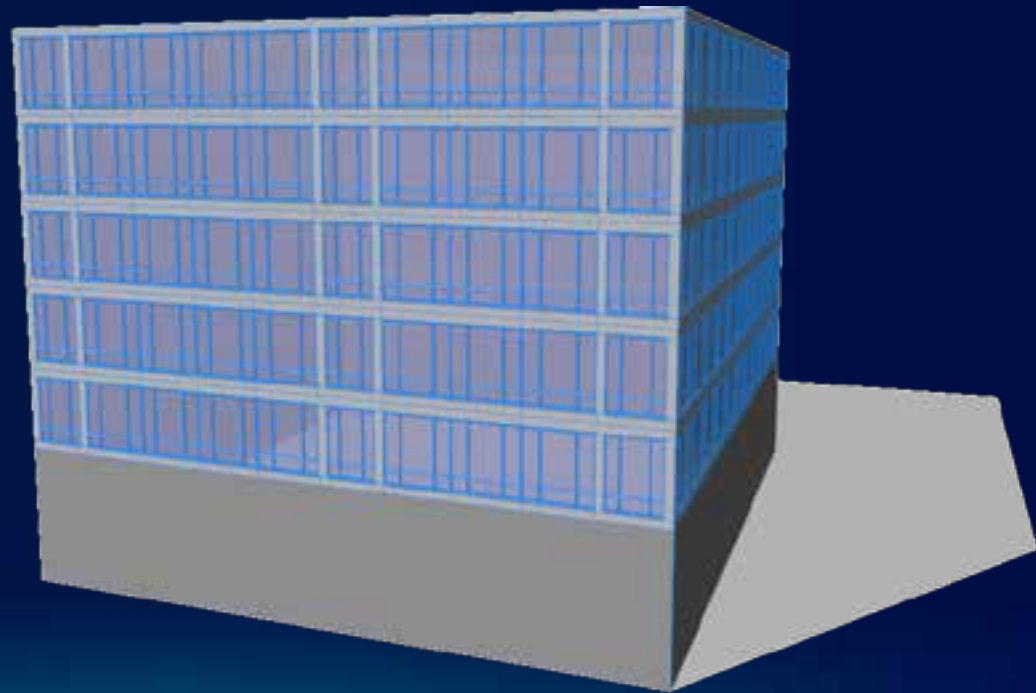
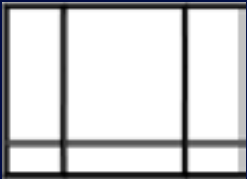
§ { A B }* A

```
§ split(x) { { widthA: TileA  
  | ~ widthB : TileB }*  
  | widthA : TileA }
```



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

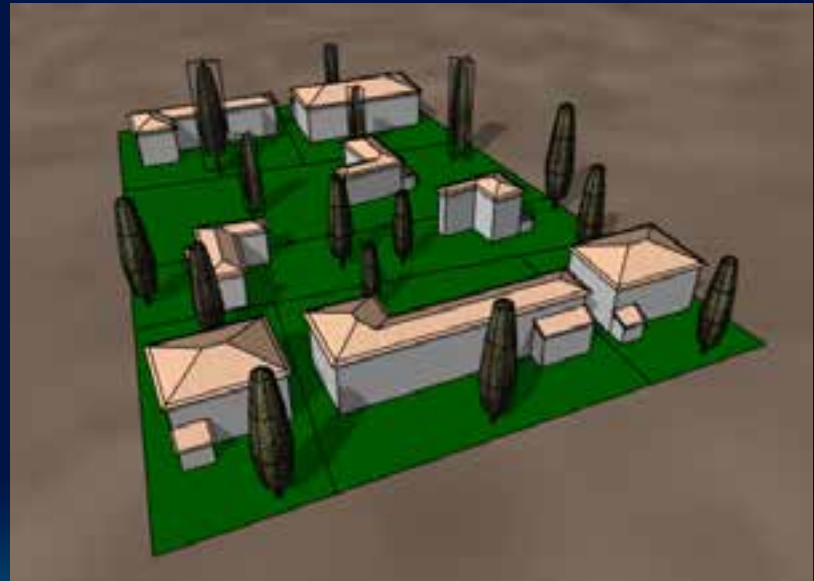
CGA Shape Grammar: Residential Building

A decorative graphic at the bottom of the slide features a satellite-style map of a residential area with green trees and blue roads. Overlaid on this map are several lines of code in a light blue font, which are partially obscured by a thick, curved orange band. The code appears to be JavaScript or a similar programming language, likely related to the CGA Shape Grammar mentioned in the title.

```
esri.symbol.SimpleLineSymbol({color:[0,0,0],width:2})  
new dojo.Color([0,0,0]);  
feature.setSymbol(symbol);  
} else if(f == 1) {  
    var polysymbolGreen = new esri.symbol.SimpleLineSymbol({color:[0,0,0.5],width:2});  
    polysymbolGreen.setOutlineColor([0,0,0.5], 1);  
    polysymbolGreen.setSymbol(feature);  
    feature.setSymbol(polysymbolGreen);  
} else if(f == 2) {  
    polysymbolBlue = new esri.symbol.SimpleLineSymbol({color:[0,0,1],width:2});  
    polysymbolBlue.setOutlineColor([0,0,1], 1);  
    polysymbolBlue.setSymbol(feature);  
    feature.setSymbol(polysymbolBlue);  
}
```

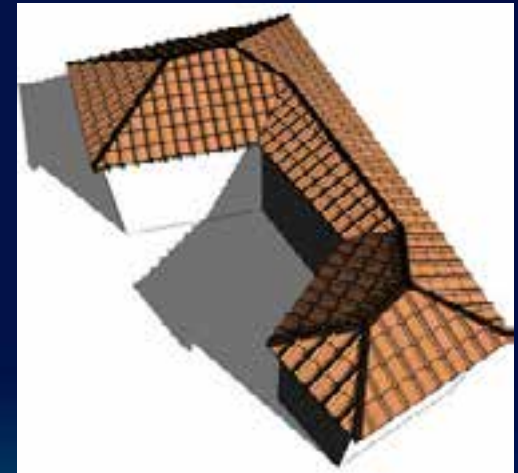
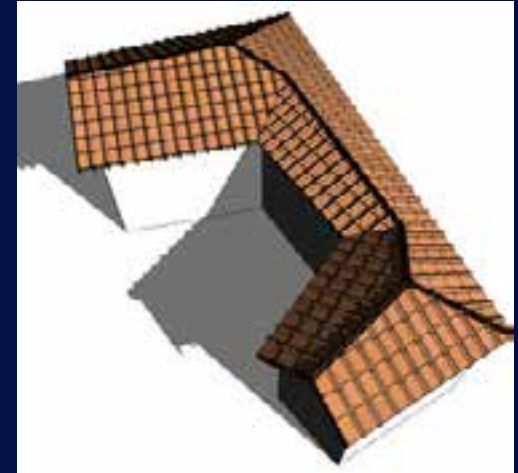
CGA : Residential Building

- **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

```
attr LOD = 1  
case LOD > 0 : "hires_asset.obj"  
else         : "billboard_asset.obj"
```



Python Scripting

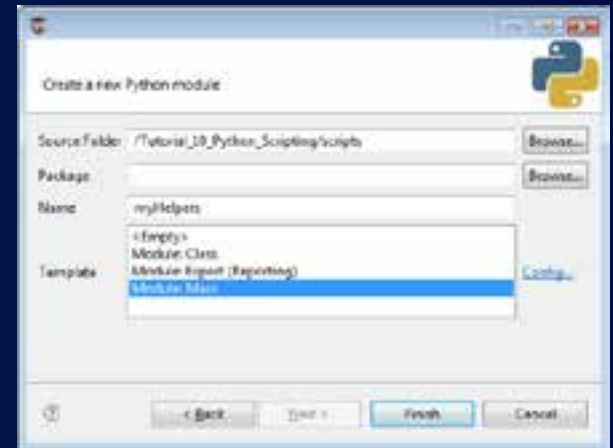
Simon Schubiger



```
esri.symbol.SimpleLineSymbol([0,0,0], 1)
new dojo.Color([0,0,0], 1)
feature.setSymbol(polySymbol)
}
else if(f == 1) {
var polySymbolGreen = new esri.symbol.SimpleLineSymbol([0,0,0], 1);
polySymbolGreen.setOutlineColor([0,0,0,0.5]);
feature.setSymbol(polySymbolGreen);
}
else if(f == 2) {
var polyBlue = new esri.symbol.SimpleLineSymbol([0,0,0], 1);
polyBlue.setOutlineColor([0,0,0,0.5]);
feature.setSymbol(polyBlue);
}
```


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")
```



scripts/export.py

Python: Export to a set of files

```
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")
```



scripts/export.py

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()`



```
# 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: Write report data to file 1

```
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()
```

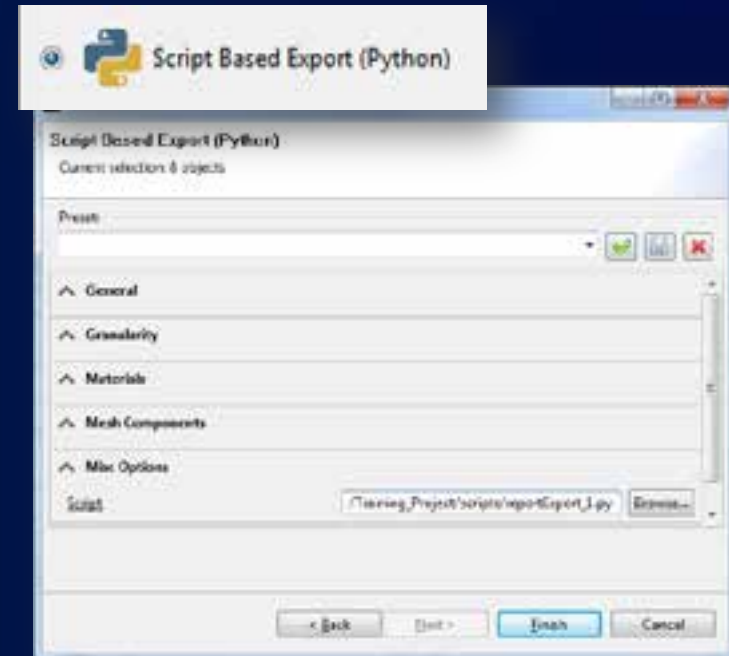


scripts/reportExport_1.py

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*

```
Lot_3,2615.475098  
Lot_2,2573.790283  
Lot_7,1753.116943  
Lot_4,2815.327881  
Lot_1,1365.432495  
Lot_6,2164.343994  
Lot_5,2069.638184  
Lot_0,2551.697510
```





Procedural Runtime

Simon Schubiger

```
esri.symbol.SimpleLineSymbol({
  color: [0, 0, 255],
  width: 2,
  style: 'solid'
});
new dojo.Color([0, 0, 255]);
feature.setSymbol(esri.symbol.SimpleLineSymbol({
  color: [0, 0, 255],
  width: 2,
  style: 'solid'
}));
else if(f == 1) {
  var polysymbolGreen = new esri.symbol.SimpleLineSymbol({
    color: [0, 0, 0.5],
    width: 2,
    style: 'solid'
  });
  polysymbolGreen.setOutlineColor([0, 0, 0.5]);
  polysymbolGreen.setSymbol(polysymbolGreen);
  feature.setSymbol(polysymbolGreen);
}
}
polylue = new esri.symbol.SimpleLineSymbol({
  color: [0, 0, 255],
  width: 2,
  style: 'solid'
});
setOutlineColor([0, 0, 255]);
esri.symbol.SimpleLineSymbol({
  color: [0, 0, 255],
  width: 2,
  style: 'solid'
});
new dojo.Color([0, 0, 255]);
polylue;
```

DISCLAIMER

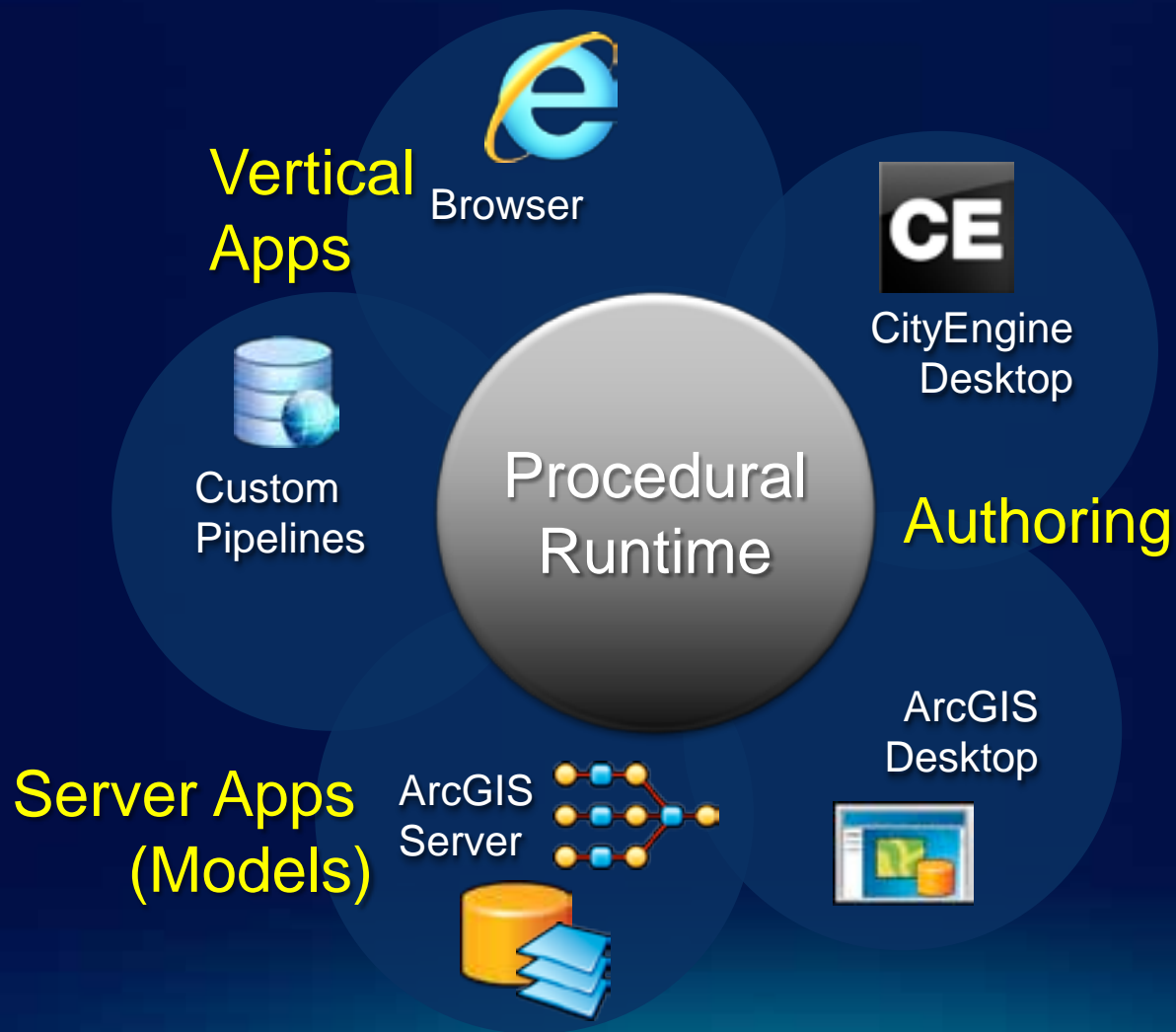


THIS IS PRELIMINARY INFORMATION

SUBJECT TO CHANGE

Procedural Runtime Eco-System

Sneak Preview



Procedural Runtime Use Cases



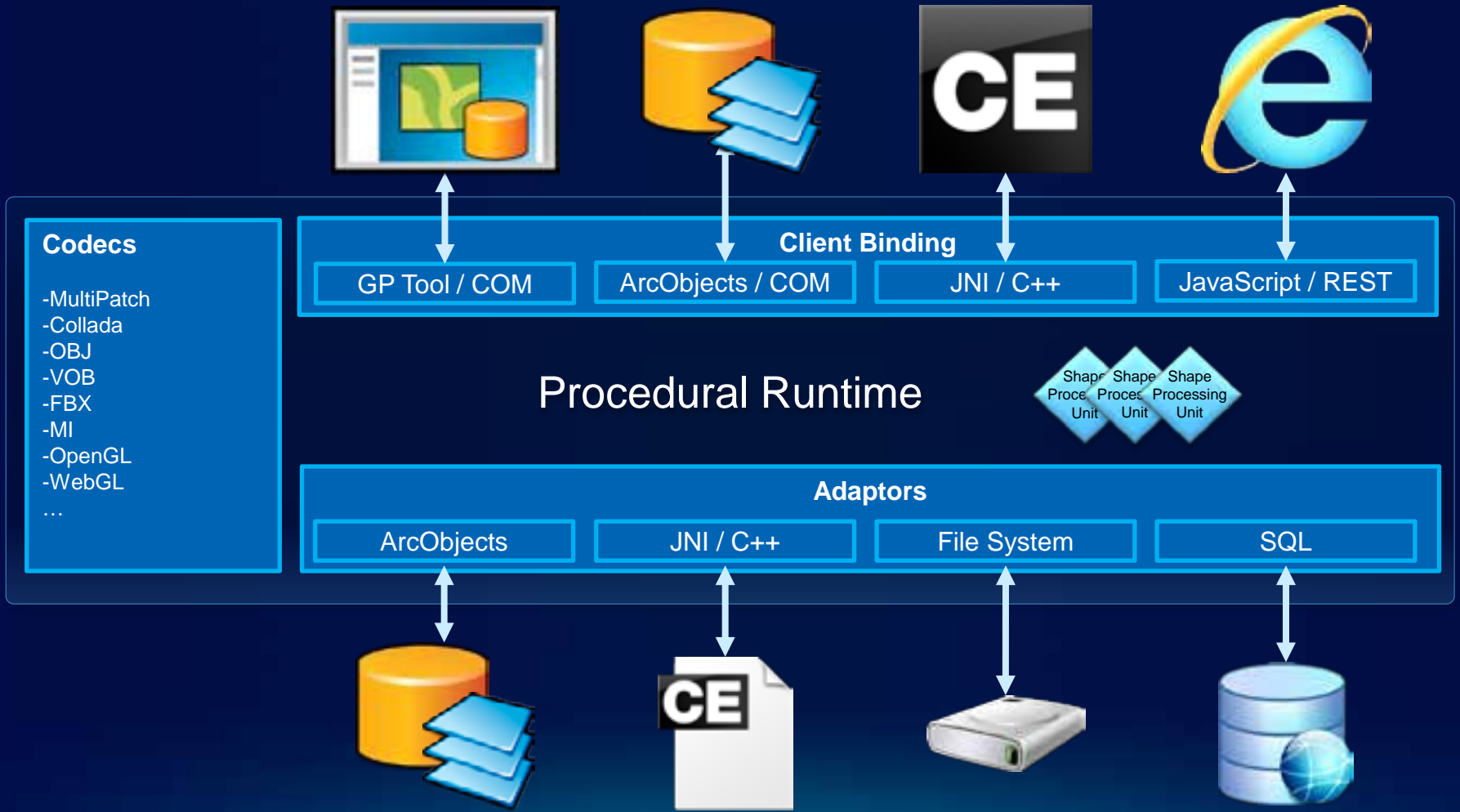
- **Entertainment Pipelines**
 - DLL, no ArcGIS e.g. proprietary exporters or rendering with generation on demand
- **ArcGIS Desktop**
 - GP Tool e.g. attribute-driven building geometries or parametric power poles in ArcScene
- **ArcGIS Server**
 - GP Service for 3D maps, 3D analytics

Procedural Runtime Architecture



Procedural Runtime Architecture

Sneak Preview



There is more about 3D Cities

Creating, managing and utilizing a 3D Virtual City

- Tamrat Belayneh, Eric Wittner, Nathan Shepard

Smoketree A-E



esri