Tutorial 6: Basic shape grammar

Table of Contents

Tutorial 6: Basic shape grammar

Download items

- Tutorial data
- Tutorial PDF

Model a simple building

This tutorial introduces the basics of the CGA shape grammar of CityEngine. You'll analyze a finished rule file that contains all the steps to create a basic building.

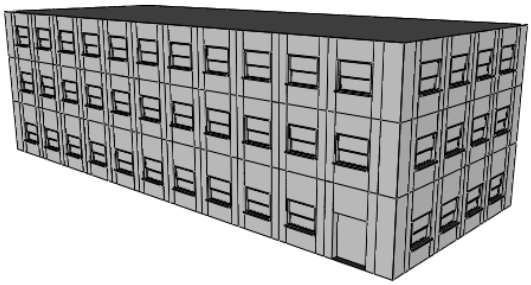
Tutorial setup

Steps:

1. Import the Tutorial_06_Basic_Shape_Grammar project into your CityEngine workspace.
2. Open the `Tutorial_06_Basic_Shape_Grammar/scenes/01_SimpleBuilding.cej` scene.

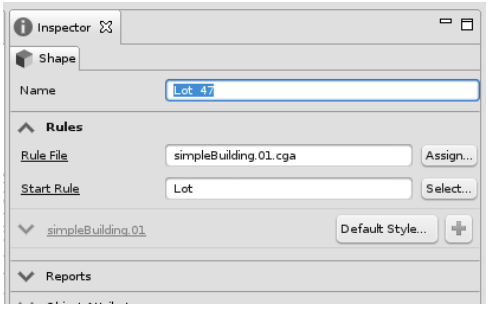
Simple building

You'll construct a simple building with a typical facade. The result of this section will look like the following image:



Steps:

1. Select the lot (or the model if it's already generated) in the 3D viewport, and review the information in the Inspector window (if it's not open, open it by clicking **Window** > **Inspector**).



Two important parameters are found here:

- Rule File—Contains a link to the `rules/simpleBuilding.01.cga` rule file. This rule file is executed when the generation is triggered.
- Start Rule—Defines the first rule that is executed within the rule file. In this case, the start rule is Lot.

2. Open the CGA rule file by either clicking the rule file link in the Inspector or double-clicking the`rules/simpleBuilding.01.cga` file in the Navigator to open the file in the CGA Editor.

Simple building rule set

Building attributes

Building attributes are typically defined at the beginning of the rule file (although they can be put anywhere in the rule file). These attributes are used throughout the entire rule set and appear in the CGA Attribute Mapping Area of the Inspector where their values can be set and modified outside the CGA Grammar Editor as well.

```
attr groundfloor_height        = 4
attr floor_height                 = 3.5
attr tile_width                 = 3
attr height                          = 11
attr wallColor                 = "#fefefe"
```

Window asset

The window asset used for the creation of the simple building is defined here. The actual asset is loaded from the project's assets folder in the Navigator.

```
// geometries
window_asset         = "facades/window.obj"
```

Lot rule

The actual creation of the building starts now. The first rule is called Lot. Remember the assigned start rule in the Inspector. The mass model is created with the extrude operation as follows:

```
Lot -->
    extrude(height) Building
```

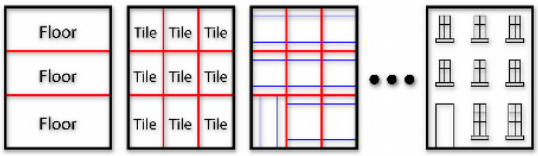
Building rule

Typically in this step, a mass model can be divided into its facades by applying the component split.

```
Building -->
    comp(f){ front : FrontFacade | side : SideFacade | top: Roof}
```

This rule splits the shape named Building, the mass model, into its faces by applying a component split. This results in a front shape (usually the main front facade with entrance), several side shapes as facades, and a roof shape.

The facades can now be modeled. The typical facade modeling workflow is as follows: First, the facade can be decomposed into floors. Next, the floors are further broken down into elements called tiles (floor subdivisions). A tile typically consists of wall and window elements. This subdivision scheme can be implemented in the CGA shape grammar as follows:



FrontFacade rule

The FrontFacade rule splits the front face into a groundfloor shape of height 4 and repeats (using the repeat operator [*]) upperfloor shapes of approximate height 3.5. The tilde operator (~) guarantees that regardless of the building's actual height, a full number of upper floors is always created. The appearance of the ground floor is often different from the other floors, especially for front facades. They differ not only due to the existence of an entrance, but often also due to different sized floor heights, window appearance, color, and so on.

```
FrontFacade -->
    split(y){ groundfloor_height : Groundfloor |
        { ~floor_height: Floor }* }
```

SideFacade rule

The SideFacade rule splits the side facades into floor shapes. The subdivision split is performed in the same way to assure that the floor heights are in sync with the front facade.

```
SideFacade -->
    split(y){ groundfloor_height: Floor | { ~floor_height: Floor }* }
```

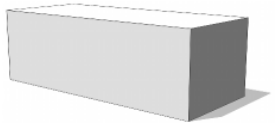
Floor rule

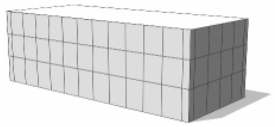
The Floor rule is a typical example of the subdivision of a floor into tiles of an approximate width of 3. To make the floor design slightly more interesting, you'll split a wall element of width 1 on each side.

```
Floor -->
    split(x){ 1: Wall
            | { ~tile_width: Tile }*
            | 1 : Wall }
```

Groundfloor rule

The Groundfloor rule refines the groundfloor shape with a similar subdivision split, with the difference that an entrance is placed on the right. The following figures depict the extruded mass model first and the described decomposition into floors and tiles below.





```
Groundfloor -->
    split(x){ 1: Wall
            |{ ~tile_width: Tile }*
            | ~tilewidth: EntranceTile
            | 1: Wall }
```

Tile rule

After the initial facade structure has been defined, the tiles can be modeled.

```
Tile -->
    split(x){ ~1 : Wall
            |  2 : split(y){ 1: Wall | 1.5: Window | ~1: Wall }
            | ~1 : Wall }
```

The Tile rule defines the appearance of the tile by splitting along x- and y-axis (with a nested split). In this design, the wall elements are floating (with tilde), and the window has a fixed size of 2 in width and 1.5 in height.

EntranceTile rule

The EntranceTile rule defines the entrance shape in a similar way as the tile shape (but, obviously, with no wall on the bottom).

```
EntranceTile -->
    split(x){ ~1 : SolidWall
            |  2 : split(y){ 2.5: Door | ~2: SolidWall }
            | ~1 : SolidWall }
```

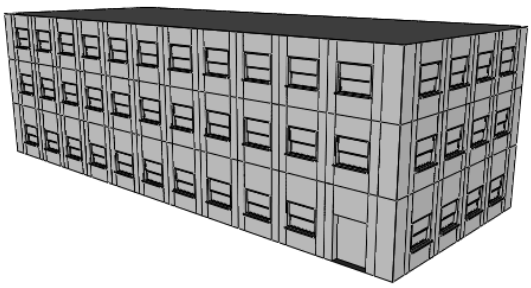
Window, Door, and Wall rules

The final rules replace the geometry of the window, door, and wall shapes with the corresponding assets, positions them, and sets the texture.

```
Window -->
    s('1,'1,0.4)
    t(0,0,-0.25)
    i(window_asset)

Door -->
    s('1,'1,0.1)
    t(0,0,-0.5)
    i("builtin:cube")

Wall -->
    color(wallColor)

SolidWall -->
    color(wallColor)
    s('1,'1,0.4)
    t(0,0,-0.4)
    i("builtin:cube:notex")
```

The current shape is translated -0.25 in the z-direction using the operation t(x,y,z). This way, the windows and textures are set back 0.25 meters into the facade. Next, the insert operation i(objectname) inserts an asset into the current scope. If the dimensions are not set the same as the Window or Door rule, the sizes are adapted automatically—otherwise the given dimensions are used. Using the operation s(x,y,z), the size of the scope can be set in the Wall rule. The width and height of the scope are not affected since relative coordinates are used: the x and y dimensions of the current scope are scaled by one ('1) resulting in no change. The z dimension is set to -0.4 resulting in a wall with a thickness of 0.4 meters (pointing inwards).

Putting all of these rules together, you get the final untextured simple building:



In the next section, you'll learn how to add textures to the simple building.

Texture the simple building

In this section, you'll learn how to apply textures to window and wall elements of the simple building.

Tutorial setup

Steps:

1. Open the `Tutorial_06_Basic_Shape_Grammar/scenes/01_SimpleBuilding.cej` scene if it's not already open.
2. Open the rule file. Use the rule you just created, or double-click the `Tutorial_06_Basic_Shape_Grammar/rules/simpleBuilding.01.cga` file in the Navigator to open the CGA Rule Editor.

Texture declaration

As with the assets, you'll define the textures you'll use at the top of the rule file. The textures are loaded from the assets folder.

Steps:

1. Add the following new lines to your rule file below the window_asset declaration.

```
// textures
frontdoor_tex                = "facade/shopdoor.tif"
wall_tex                   = "facade/brickwall.jpg"
dirt_tex                  = "facade/dirtmap.15.tif"
roof_tex                   = "roof/roof.tif"
```

Nine different window textures are prepared in the project's `assets/facade` folder. Rather than listing all nine texture names as single assets, the function in step 2 returns one of the nine window textures in the asset folder.

2. Add the following line to the rule file:

```
randomWindowTexture = fileRandom("*facades/textures/window.*.tif")
```

3. Add the two lines in red to the Frontfacade and Sidefacade rules:

```
Frontfacade -->
        setupProjection(0, scope.xy 1.5, 1, 1)
        setupProjection(2, scope.xy, scope.sx, scope.sy)
        split(y){ groundfloor_height : Groundfloor | { ~floor_height: Floor }* }

Sidefacade -->
        setupProjection(0, scope.xy, 1.5, 1, 1)
        setupProjection(2, scope.xy, scope.sx, scope.sy)
        split(y){ groundfloor_height: Floor | { ~floor_height: Floor }* }
```

The setupProjection() command prepares the UV coordinate projections on the facades for color (channel 0) and dirt map (channel 2), projected onto the scopes xy plane; therefore, scope.xy, is set as the second parameter.

The brick texture (channel 0) will be repeated every 1.5m in X and every 1m in Y axis, whereas the dirt map (channel2) will span the entire facade and uses scope.sx and scope.sy as size parameters.

4. Add a new Roof rule.

```
Roof -->
        setupProjection(0, scope.xy, scope.sx, scope.sy)
        texture(roof_tex)
        projectUV(0)
```

The Roof rule prepares the UV coordinates to cover the entire roof face, sets the roof texture, and applies the texture coordinates to the geometry.

5. Add the red lines to the subsequent rules.

```
Window -->
        s('1,'1,0.4)
        t(0,0,-0.25)
        texture(randomWindowTexture)
        i(window_asset)

Door -->
        s('1,'1,0.1)
        t(0,0,-0.5)
        texture(frontdoor_tex)
        i("builtin:cube")
```

6. For window and door elements, you only set the colormap to the desired texture. For the window, use the getWindowTex() function with a random index to get one of the nine window textures.

```
Wall -->
        color(wallColor)
        texture(wall_tex)
        set(material.dirtmap, dirt_tex)
        projectUV(0) projectUV(2)

SolidWall -->
        color(wallColor)
        s('1,'1,0.4)
        t(0,0,-0.4)
        texture(wall_tex)
        set(material.dirtmap, dirt_tex)
        i("builtin:cube:notex")
        projectUV(0) projectUV(2)
```

Wall and SolidWall use the UVs prepared in the Facade rules. In addition to choosing the textures for color and dirt channels, you also need to project the UVs on those two channels.

The following image shows the final building model:

A close-up view of the same model:

The following image shows the rule set applied to arbitrary mass models:

Add level of detail

In this section, you'll add a simple level of detail (LOD) mechanism to your simple building. You'll reduce the complexity (polygon count) of the model, which is helpful when creating larger areas of simple buildings.

Tutorial setup

Steps:

1. Open the `Tutorial_06_Basic_Shape_Grammar/scenes/02_SimpleBuilding.cej` scene if it's not already open.
2. Open the rule file. Use the rule you just created, or double-click the `Tutorial_06_Basic_Shape_Grammar/rules/simpleBuilding.02.cga` file in the Navigator to open the CGA Rule Editor.

Add a level of detail attribute

Add a new attribute LOD to the existing attributes in the .cga file.

```
attr LOD = 1
```

In this example, you'll define the following two levels of detail:

- LOD 0—Low level of detail, low complexity
- LOD 1—High level of detail, high complexity

The simple building you already modeled will be your high resolution model. You'll now create a low resolution version in a few steps.

Examining your current model, you'll see that you can save polygons mainly on the window assets. You'll use textured planes instead of the complex window asset.

Steps:

1. Add the red lines to the Window rule.

```
Window -->
        case LOD > 0 :
          s('1,'1,0.4)
          t(0,0,-0.25)
          texture(randomWindowTexture)
          i(window_asset)
        else :
          setupProjection(0,scope.xy,scope.sx,scope.sy)
          texture(randomWindowTexture)
          projectUV(0)
```

You added a condition to the Window rule: If the LOD value is greater than 0 (your high-res), use the old, high-res window asset. Otherwise (LOD equals zero), do not load the window asset, but use a simple plane coming from the facade instead.

2. Likewise with the Door rule: Load a simple plane rather than a cube.

```
Door -->
        case LOD > 0 :
         s('1,'1,0.1)
         t(0,0,-0.5)
         texture(frontdoor_tex)
        else :
         setupProjection(0,scope.xy,scope.sx,scope.sy)
         texture(frontdoor_tex)
         projectUV(0))
```

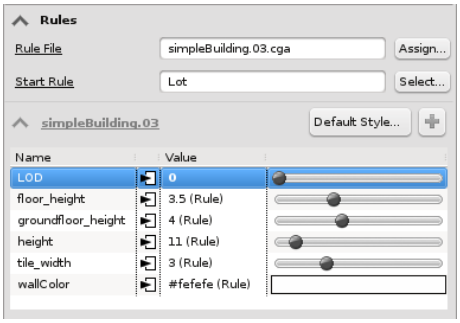
3. Now for the SolidWall elements. Because you removed the inset of the door, you don't need solid elements anymore.

```
SolidWall -->
        case LOD > 0 :
            color(wallColor)
            s('1,'1,0.4)
            t(0,0,-0.4)
            texture(wall_tex)
            set(material.dirtmap, dirt_tex)
            i("builtin:cube:notex")
            projectUV(0) projectUV(2)
    else :
            Wall
```

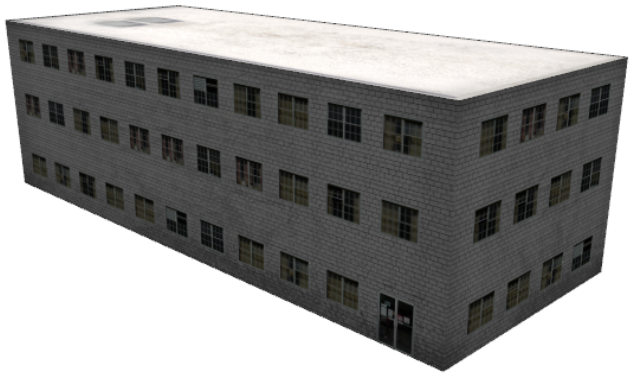
4. With the building selected, review the attribute field of the Inspector. The new LOD attribute is listed here. Change the value to 0.

   The source field changes from Rules to Value. This means that on the next generation of the building, the LOD value in the rule file will be set by the value 0 in the Inspector.

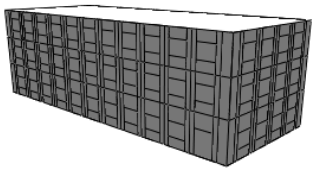
   The following screen shot shows the new LOD attribute in the Inspector with LOD = 0:

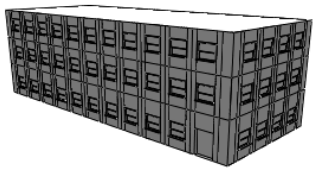


5. Generate the building to get the low level version shown in the following image with LOD = 0:



6. Displaying the model with wireframe on shaded (press 7) and untextured (press 5), the differences are clearly visible. Press d-d to show the headup display at the top of the viewport containing the polygon count. The model was reduced from 3699 to 475 polygons.





In the next section, you'll add random variation to your simple buildings.

Random variation of building attributes

This section shows how to add variation to generated buildings by defining random attributes.

Tutorial setup

Steps:

1. Open the `Tutorial_06_Basic_Shape_Grammar/scenes/03_SimpleBuilding.cej` scene if it's not already open.
2. Open the rule file. Use the rule you just created, or double-click the `Tutorial_06_Basic_Shape_Grammar/rules/simpleBuilding.03.cga` file in the Navigator to open the CGA Rule Editor.

Add random attributes

Steps:

1. You'll add variation to three of your building attributes: A random tile width between 2.5 to 6 meters, a random building height between 8 and 35 meters, and three colors that are chosen randomly for each building.

```
attr tile_width        = rand(2.5, 6)
attr height               = rand(8, 35)
attr wallColor        = 33%  :  "#ffffff"
                           33%  :  "#999999"
                      else :  "#444444"
```

2. Since you'll generate a larger number of buildings, change the default LOD value from 1 to 0.

```
attr LOD = 0
```

3. Select the Lots 2 layer.
4. Generate the buildings by clicking **Shapes** > **Generate**.