

# Tutorial 8: Mass modeling



# Table of Contents

Tutorial 8: Mass modeling . . . . .	3
-------------------------------------	---

# Tutorial 8: Mass modeling

## In this tutorial

- [Download items](#)
- [L and U shapes](#)
- [Mass modeling using recursion](#)
- [Adapt the parcel with setbacks](#)
- [Combine masses and setback parcels](#)
- [Add textured facades](#)

## Download items

- [Tutorial data](#)
- [Tutorial PDF](#)

## L and U shapes

### Tutorial setup

This tutorial shows how mass models of buildings can be created with the shape grammar. Typical architectural volume shapes such as L and U masses will be created.

#### Steps:

1. Import the Tutorial\_08\_Mass\_Modeling project into your CityEngine workspace.
2. Open the MassModeling\_01.cej scene.

### Create the rule file

#### Steps:

1. Click **File > New > CityEngine > CGA Rule File**.
2. Make sure the container is set correctly (Tutorial\_08\_Mass\_Modeling/rules), name the file `myMass_01.cga`, and click **Finish**.  
A new CGA file is created, and the CGA Editor is opened.

## L shape


You'll start with a simple L-shape.

```
attr height = rand(30, 60)
attr wingWidth = rand(10, 20)

Lot --> LShape
LShape -->
  shapeL(wingWidth, wingWidth) { shape : LFootprint }
LFootprint --> extrude(height) Mass
LotInner --> OpenSpace
```

The `shapeL` command creates an L shape footprint, with dimensions ranging between 10 and 20. The second `LFootprint` rule then extrudes the L-shape to its height.

`LotInner` applies `OpenSpace`, leaving the lot shape as is.

 **Note:** Attributes can have optional annotations, such as `@Group` or `@Range`, which control the display of the attributes in the Inspector. See the CGA Reference for details on CGA annotations.

Now you'll apply the rule to the lots.

#### Steps:

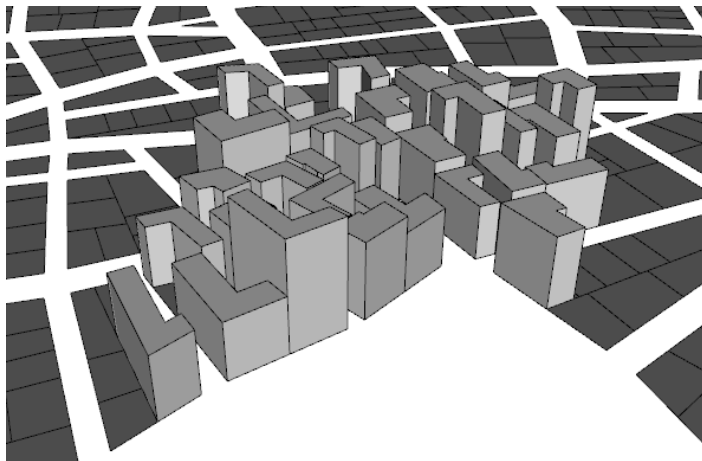
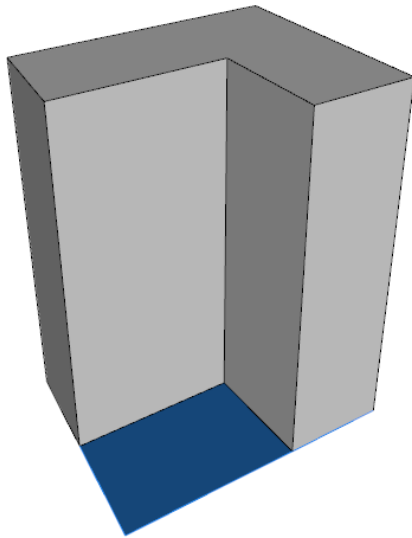
1. Select the Lots layer in the Scene Editor.

2. Click **Shapes > Assign Rule File**.
3. Choose the myMass\_01.cga rule file from the rules directory and click **OK**.

### Generate the buildings

Steps:

1. Select some of the lots in the 3D viewport.
2. Click **Shapes > Generate** or **Ctrl+G** to generate the buildings.



The L-shaped mass models work; however, they don't look very convincing. You need more variation.

### Varying L shape

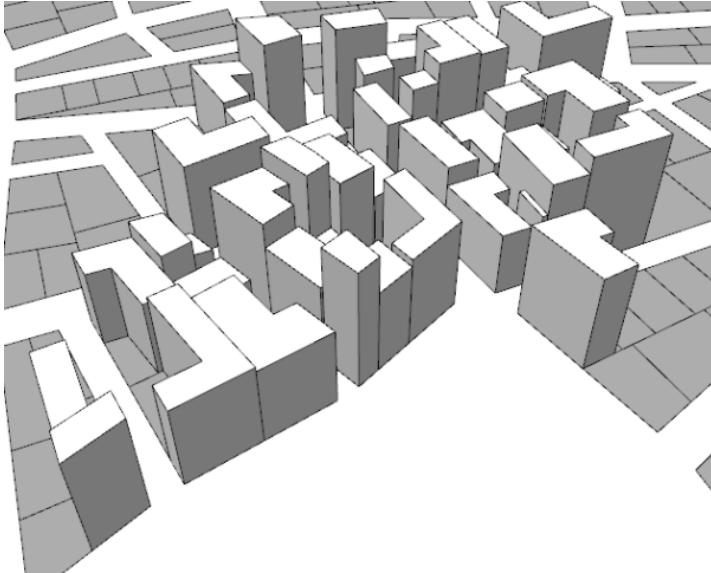
The current L shape's side wing is always positioned on the left side of the main mass. Using `rotateScope`, you can change the L shape to be on the right side as well.

Change the red lines in the LShape rule using the probability operator `%` to improve the L shape to have its side wing on either the left or right side.

```
LShape -->
  50% : shapeL(wingWidth, wingWidth) { shape : LFootprint }
  else : rotateScope(0, 90, 0)
        shapeL(wingWidth, wingWidth) { shape : LFootprint }
```

Using the convexify command, you can split the L shape into its wings, and change the height of the two wings. Again you'll use random probability to add variation.

```
LFootprint -->
  75% : extrude(height) Mass
  else : convexify comp(F) { 0 : extrude(height) Mass |
                           all : extrude(height * 0.7) Mass }
```



Side wings now appear on the left and right sides and vary in height. Now you need additional shapes.

### U shape

You'll add a U shape. Call UShape instead of LShape in the starting Lot rule.

```
Lot --> UShape
```

Similarly, you'll use the shapeU command.

```
UShape -->
  shapeU(wingWidth, wingWidth * 0.7, wingWidth * 0.7)
  { shape : UFootprint }

UFootprint --> extrude(height) Mass
```

Add some variation as well.

```
UShape -->
  80% : rotateScope(0, 180, 0) shapeU(wingWidth, wingWidth * 0.7,
  wingWidth * 0.7) { shape : UFootprint }
  else: shapeU(wingWidth, wingWidth * 0.7, wingWidth * 0.7)
  { shape : UFootprint }
```



When looking at this image, you can see that U shapes do not work well on all lots. You'll correct this in the next section.

### L and U shapes combined

The height distribution is not convincing. To have better control over the building heights, you'll add a condition to the height attribute. Only large area lots should be allowed to create tall buildings.

```
attr height =
  case geometry.area < 1000: rand(20, 50)
  else: rand(50, 150)
```

You'll add a new rule, LUShapes, to control what shape is created on which lots.

U shapes work best on lots that are wider than deep. Or in CGA language, where `scope.sx` is larger than `scope.sz`. In any other case, you'll only trigger L shapes.

```
Lot --> LUShapes
```

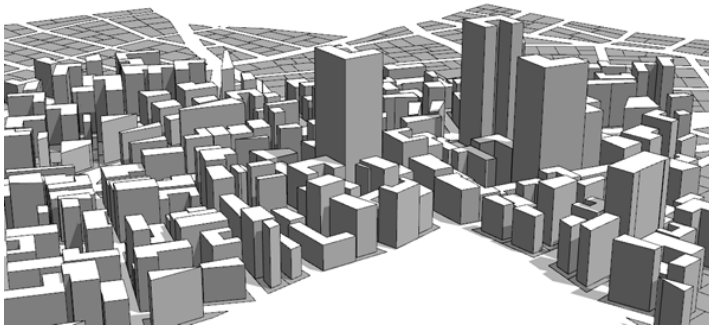
```
LUShapes -->
  case scope.sx > scope.sz :
    60% : UShape
    else : LShape
  else: LShape
```

Neither L nor U shapes work well on non-rectangular lots. The next case statement makes sure UShape and LShape are only created on approximately rectangular lots (with a tolerance of 15 degrees). Otherwise, you'll call a new footprint rule.

```
LUShapes -->
  case geometry.isRectangular(15):
    case scope.sx > scope.sz :
      60% :UShape
      else : LShape
    else: LShape
  else: BasicFootprint
```

Extruded lot shapes will be too large compared to the L and U shapes. Consequently, you'll add a negative offset to the BasicFootprint. This will also allow more space between individual buildings.

```
BasicFootprint --> offset(-5,inside) extrude(height) Mass
```



In the next section, you'll learn how to use recursion in the Shape Grammar for mass modeling.

## Mass modeling using recursion

### Tutorial setup

This tutorial shows how to model repetitive building elements using recursive Shape Grammar calls.

Open the MassModeling\_01.cej scene if it's not already open.

### Create the rule file

Steps:

1. Click **New > CityEngine > CGA Grammar File**.
2. Make sure the container is set correctly (Tutorial\_08\_Mass\_Modeling/rules), name the file `myMass_02.cga`, and click **Finish**.

### Tower shapes

The attribute `height` creates a random value for the building height. The starting rule `Lot` calls `Envelope`, which extrudes the footprint to the towers envelope. `Envelope` calls the recursion rule.

```
height =
  case geometry.area > 1000: rand(50, 200)
  else: rand(20, 50)

Lot --> Tower

Tower --> extrude(height) Envelope

Envelope --> RecursiveSetbacks
```

For the subsequent recursive rules, you need two additional variables: `lowHeight` and `scale`. The latter needs to be constant for a building, so you'll define it as an attribute.

```
// switching between these two values creates visually appealing setbacks:
lowHeight = 50%: 0.4 else: 0.6

// has to be constant:
attr scale = rand(0.75, 0.9)
```

The `RecursiveSetbacks` rule splits the mass, as long as it's higher than two floors, into a lower part mass with relative height `lowHeight`. The upper remaining part generates the shape `Setback`.

In case the `RecursiveSetbacks` shape is smaller than two stories, the remaining part is set to `floorheight` in height, and a `Mass` shape is generated.

```
attr floorheight = rand(4, 5)

RecursiveSetbacks -->
  case scope.sy > 2 * floorheight :
    split(y) { 'lowHeight : Mass | ~1: Setback }
  else:
    s('1, floorheight, '1) Mass
```

The `Setback` rule scales and centers the shape and recursively invokes the `RecursiveSetbacks` rule.

```
Setback -->  
s('scale, '1, 'scale) center(xz) RecursiveSetbacks
```

Now apply the rule to the lots.

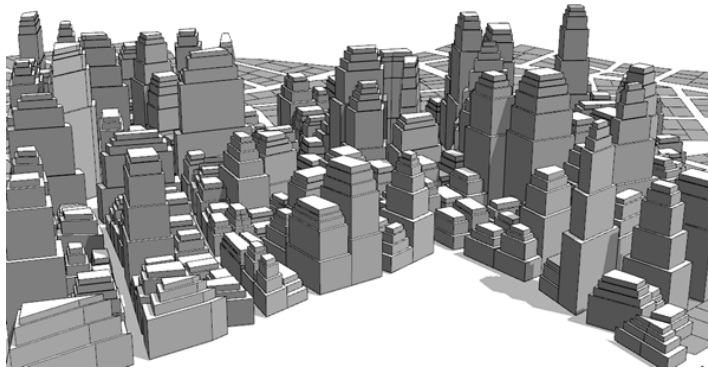
Steps:

1. Select the **Lots** lot layer in the Scene Editor.
2. Click **Shapes > Assign Rule File**.
3. Choose the myMass\_02.cga rule file from the rules directory, and click **OK**.

*Generate the buildings*

Steps:

1. Select some of the lots in the 3D viewport.
2. Click **Shapes > Generate** or press **Ctrl+G** to generate the buildings.



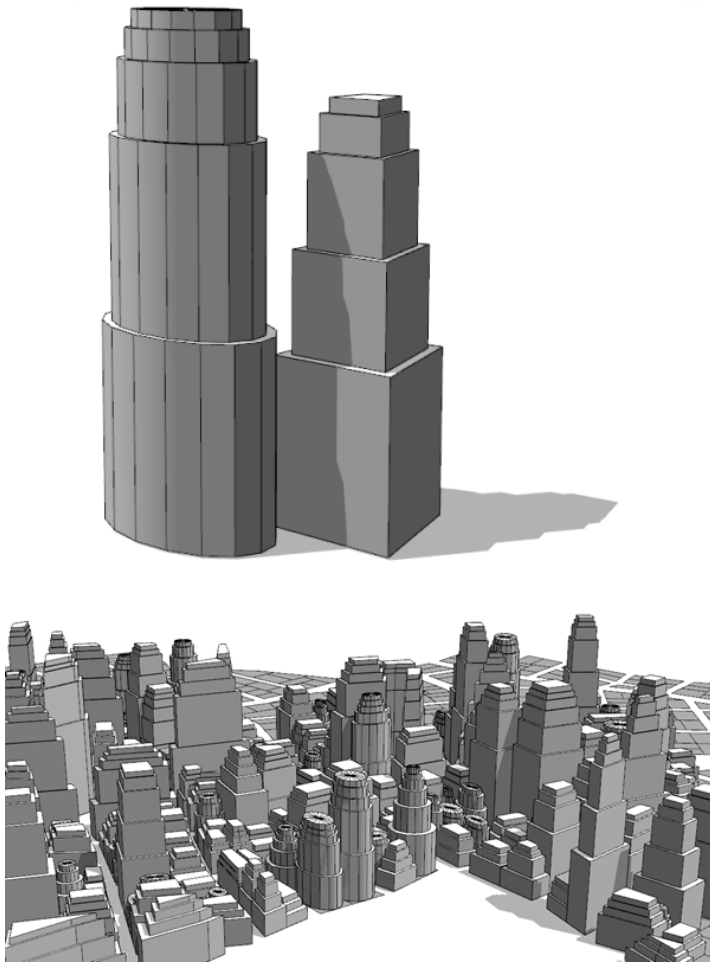
## Round shape

Using an external cylinder asset, you can create round versions of your recursive towers. Modify the Tower rule as follows:

```
Tower --> extrude(height) Envelope  
Envelope -->  
  case geometry.isRectangular(20):  
    20% : i("cyl.obj") RecursiveSetbacks  
    else: RecursiveSetbacks  
  else: RecursiveSetbacks
```

In 20 percent of all towers, you'll insert a cylinder asset instead of using the implicit cube as underlying shape.





In the next section, you'll modify the lot parcels with setbacks.

## Adapt the parcel with setbacks

### Tutorial setup

In this section, you'll apply setbacks to the lot shapes.

Open the MassModeling\_01.cej scene if it's not already open.

### Create the rule file

Steps:

1. Click **New > CityEngine > CGA Grammar File**.
2. Make sure the container is set correctly (Tutorial\_08\_Mass\_Modeling/rules), name the file `myMass_03.cga`, and click **Finish**.

### Street setback

The Parcel rule applies a setback on all street sides of the lot and forwards this area to the OpenSpace rule. The inner part, away from street sides, is forwarded to Footprint, which extrudes to a random height.

```

attr height =
  case geometry.area > 1000: rand(50, 200)
  else: rand(20, 50)

attr distanceStreet =
  20%: 0
  else: rand(3, 6)

Lot --> Parcel
LotInner --> OpenSpace

Parcel -->
  setback(distanceStreet)
  { street.front: OpenSpace
  | remainder: Footprint }

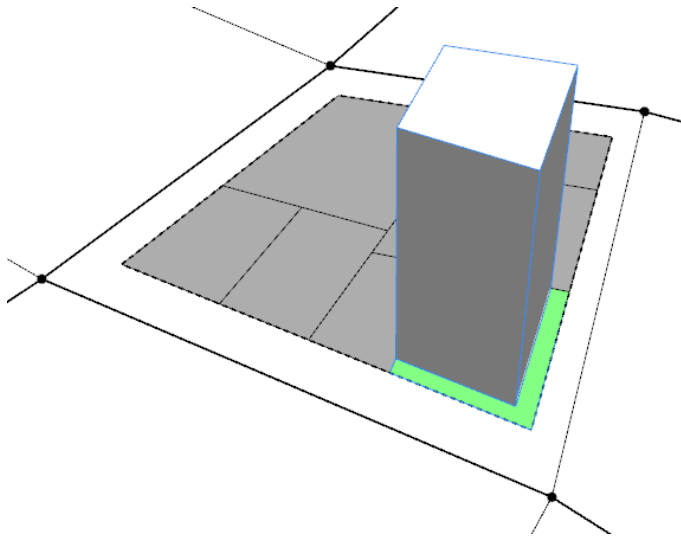
Footprint --> extrude(height)
OpenSpace --> color("#77ff77")

```

**Note:** The street.front selector evaluates the shapes streetWidth object attributes, which is automatically set for lots created from blocks but may not be present on manually created shapes.

Steps:

1. Choose the Lots lot layer in the Scene Editor.
2. Click **Shapes > Assign Rule File**.
3. Select the myMass\_03.cga rule file from the rules directory, and click **OK**.
4. Select a lot in the 3D viewport.



5. Click **Shapes > Generate** or press **Ctrl+G** to generate the buildings.

## Buildings distance

You'll now add a similar setback to control distance between buildings. Add the attr distanceBuildings, modify the Parcel rule, and add a new SubParcel rule as shown:

```

attr distanceBuildings =
  30%: 0
  else: rand(4, 8)

Parcel -->
  setback(distanceStreet)
  { streetSide: OpenSpace
  | remainder: SubParcel }

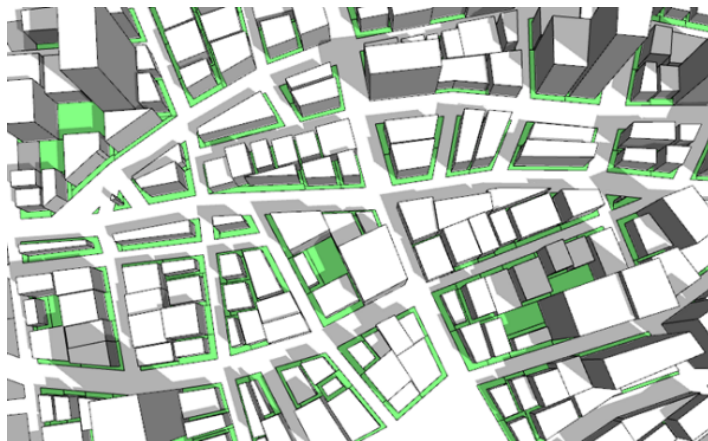
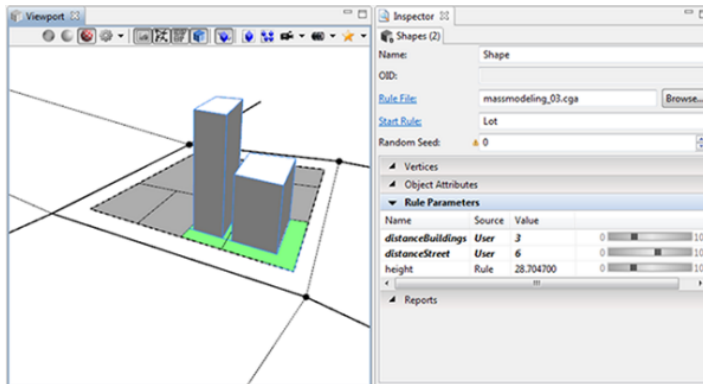
SubParcel -->
  setback(distanceBuildings / 2)
  { street.noStreetSide: OpenSpace
  | remainder: Footprint }

```

SubParcel will again apply a setback, but this time on the non-street edges.

Steps:

1. Save the .cga file.
2. Select some lots in the 3D viewport.
3. Click **Shapes > Generate** or press **Ctrl+G** to generate the buildings.
4. Select a generated model and experiment with the distanceBuildings and distanceStreet rule parameters. You can set the values on a single selected model or simultaneously select multiple models.
5. To reset the user-defined value back to the random value coming from the rule file, change the Source from User back to Rule.



In the next section, you'll combine your mass models from the previous sections with the setback parcels, and add texture facades.

## Combine masses and setback parcels

### Tutorial setup

Open the MassModeling\_01.cej scene if it's not already open.

### Create the rule file

Open the massmodeling\_03.cej rule file, and save it as myMass\_04.cga.

### Import LU shapes and tower mass rules

Steps:

1. Add the following two import commands:

```
import lushapes : "massmodeling_01.cga"
import towers : "massmodeling_02.cga"
```

2. Modify the Footprint rule as follows:

```
Footprint -->
case geometry.isRectangular(15):
  25% : towers.Tower
  else : lushapes.LUShape
else:
  25%: towers.Tower
  else: offset(-5, inside) lushapes.BasicFootprint
```

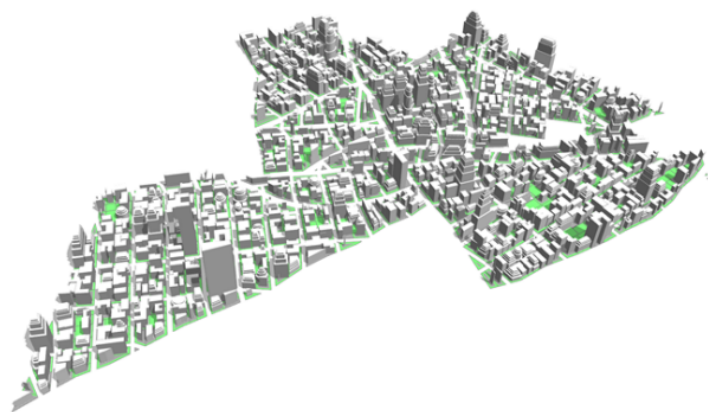
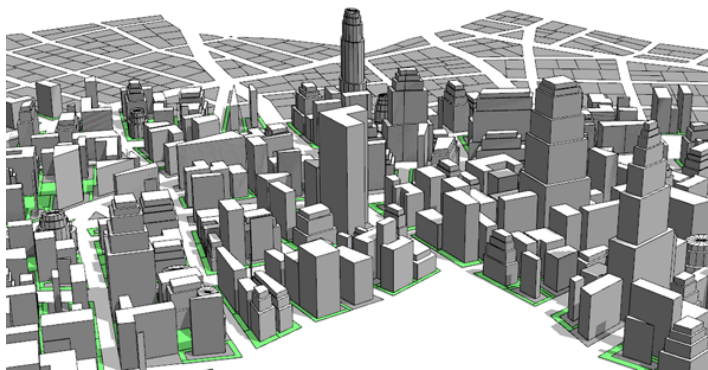
3. Save the rule file.
4. Select the **Lots** lot layer in the Scene Editor.
5. Click **Shapes > Assign Rule File**.
6. Select the myMass\_04.cga rule file from the rules directory, and click **OK**.

Generate the buildings:

*Generate the buildings*

Steps:

1. Select some of the lots in the 3D viewport.
2. Click **Shapes > Generate** or press **Ctrl+G** to generate the buildings.



In the next section, you'll add textured facades to your mass models.

## Add textured facades

### Tutorial setup

Open the MassModeling\_01.cej scene if it's not already open.

## Create the rule file

Open the massmodeling\_04.cej rule file, and save it as myMass\_05.cej.

To add simple textured facades to your mass models, you need a function that randomly selects one of your 12 facade texture tiles.

```
const randomFacadeTexture = fileRandom("**facade_textures/f*.tif")
```

To correctly map the texture tiles to your facade, you'll define two functions that calculate the actual floor height and tile width. With these functions, you'll ensure that no texture tiles are cut off at the edge of the facade.

```
attr floorheight = rand(4,5)
actualFloorHeight =
  case scope.sy >= floorheight : scope.sy/rint(scope.sy/floorheight)
  else : scope.sy
actualTileWidth =
  case scope.sx >= 2 : scope.sx/rint(scope.sx/4)
  else : scope.sx
```

With the component split, you'll get the facade components from your mass models.

```
Mass -->
  comp(f){ side: Facade | top: Roof. }
```

Instruct the imported rules to use this Mass rule.

```
towers.Mass --> Mass
lushapes.Mass --> Mass
```

Finally, set the UV coordinates on the facade, define the texture file using the **randomFacadeTexture** function, and project the UVs.

```
Facade -->
  setupProjection(0, scope.xy, 8*actualTileWidth, 8*actualFloorHeight)
  texture(randomFacadeTexture)
  projectUV(0)
```

### Steps:

1. Save the rule file.
2. Select the **Lots** lot layer in the Scene Editor.
3. Click **Shapes > Assign Rule File**.
4. Select the myMass\_05.cga rule file from the rules directory, and click **OK**.

And generate the buildings:

*Generate the buildings*

Steps:

1. Select some of the lots in the 3D viewport.
2. Click **Shapes > Generate** or press **Ctrl+G** to generate the buildings.

