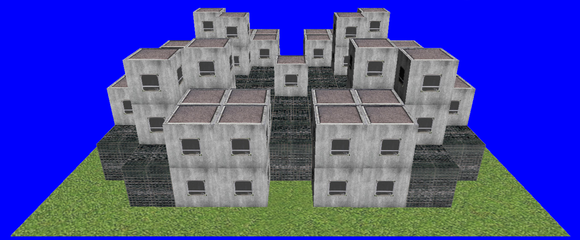
BK7084
Final Project
report

Imke den Boogert
Lucas de Vaan

Office Building



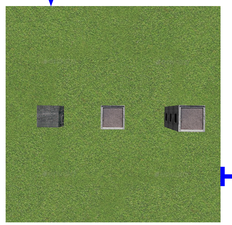
Approach

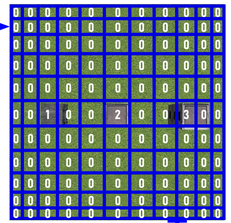
When designing the office building we ran into a few issues. We did not have coding experience before this course and we are accustomed to designing meshes by hand in a CAD software. In CAD software you design a building with direct visual feedback. When we started designing our first building by code, we missed this feedback. As a result of this we made our own tool to make it possible to design a building in a "semi-visual" way, by creating a piece of code that will litterally look like the building it will generate. We called it a so-called "config-grid".

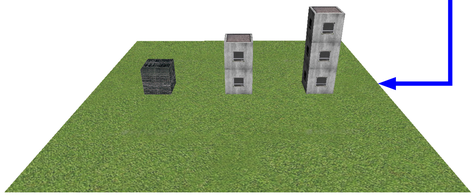
This config-grid works such a way that each number in the grid generates the corresponding amount of floors in the 3D-model on the correct position. The config-grid to the right of this text generates the model below the piece of code.

```
grid_config = [
    ['0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0'],
    ['0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0'],
    ['0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0'],
    ['0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0'],
    ['0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0'],
    ['0', '0', '1', '0', '0', '2', '0', '0', '3', '0', '0', '0'],
    ['0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0'],
    ['0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0'],
    ['0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0'],
    ['0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0'],
    ['0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0'],
]
office = Office(app, 3, grid_config=grid_config)
office.building.set_transform(Mat4.from_translation(Vec3(0, 0, 0)))
```

config-grid



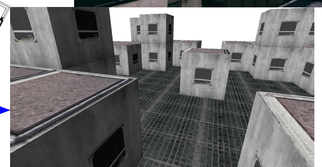




Reference

The office building is inspired by a dutch 20th century architectural movement called structuralism. Structuralism caharacterizes itself by its often geometrical looks. Structuralistic buildings often consist of small repeating parts that are designed for a human scale.

The main inspiration for the building is the Centraal Beheergebouw by Herman Hertzberger (1970). The building is still in Apeldoorn but it is unfortunately not in operation anymore.

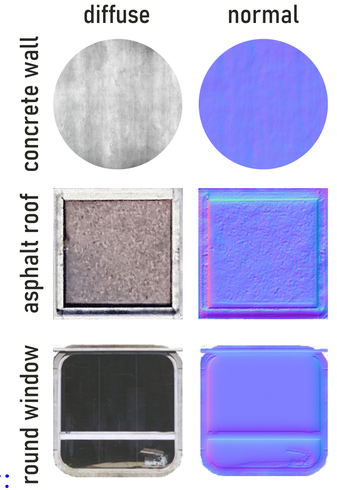


Textures

For the office building we chose for materials with a more industrial feel. We used concrete and glass blocks. We also used a round window that we borrowed from the Musée gallo-romain de Fourvière in Lyon (1975).

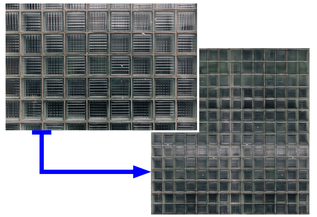
We used a photoshop feature to generate the normal maps for each texture except for the glass blocks from the office building. These normal maps give a visible depth to the textures.

We did not find a way to also make specular and shininess maps for the textures, so we used the brick maps as they would have a similar way of reflecting light as concrete. We did not use these textures for the glass blocks because they reclect light in a very different way.



We created the glass block wall textures by expanding an existing image of glass blocks with the generative-fill function included in the new Photoshop-AI.

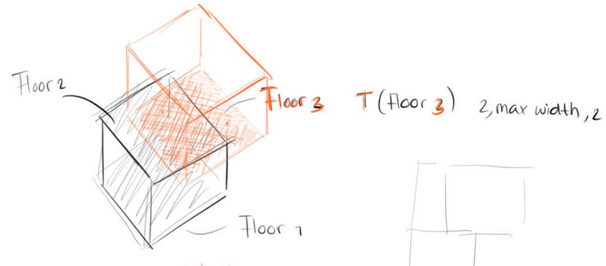
We also made it so that each piece of the building with 1 floor becomes glass-blocks. If a piece of the building has 2 or 3 floors, it will get the regular concrete textures with the round windows.

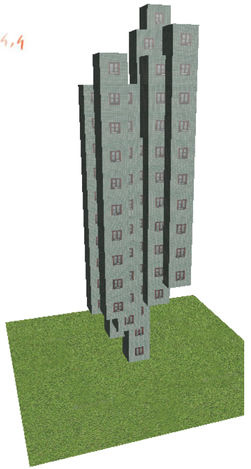


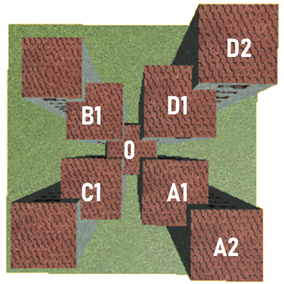
Highrise Building

Approach

The high rise building was created by playing around with copying and translating with a basic tower. After this the decision was made to make it a symmetrical building, seen from the top. But the towers around the centre start and end at different heights.



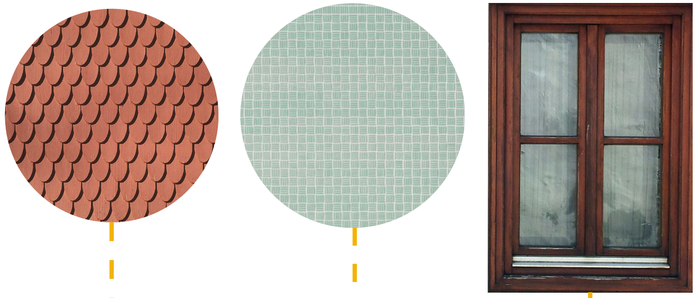




To program this there are multiple towers created in building.py. First a tower is made in the center if the building called 0. This tower is than copied eight times and translated to different positions. By defining add_walls_to_floor, walls were assigned to the floors to create a building. To add windows a WindowWall in components.py is created and assigned to wall 1 and wall 3, so the building as window on the front and on the backside.

The building consists of tower 0 (core), A1, A2, B1, B2, C1, C2, D2 & D3. The position of these towers can be found on the image next to this text.

Textures



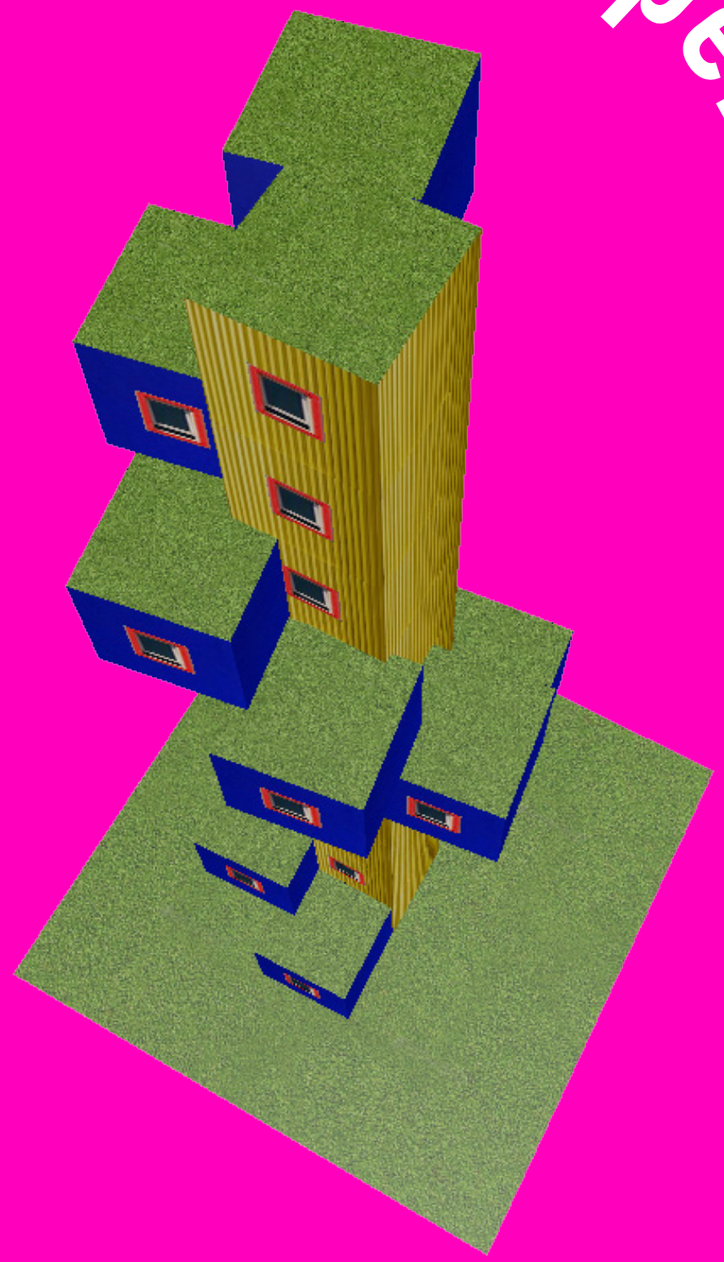
Roof tiles

Mosaic wall

Wooden windows

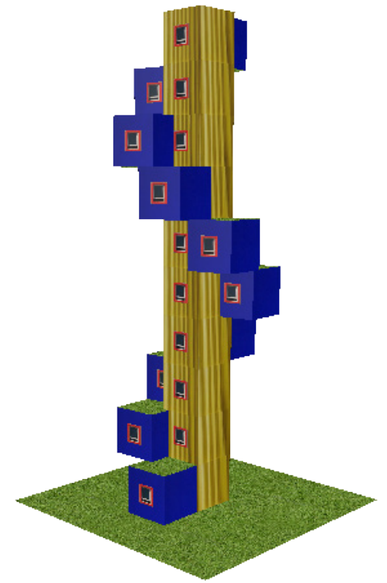
The highrise building uses images of mosaic wall tiles as wall-textures and basic ceramic rooftiles as roof-texture. It also uses an image of a wooden window as the window-texture.

Skyscraper

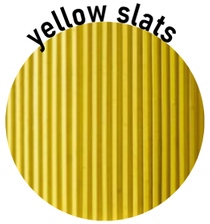


Approach

The idea of the skyscraper was having a spiralling element around a tower. Modeling this is pretty straight-forward. The first step of creating our skyscraper is making a long 20-story straight building. The second step is creating floors that rotate each iteration around the rigid core. The angle being defined by the following formula: angle = i + 2 * math.pi / 4.



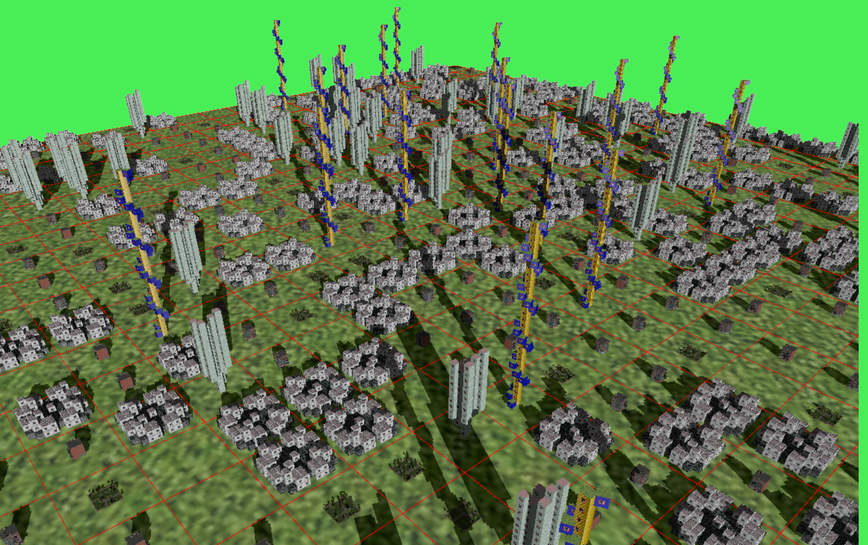
Textures



red window

When designing the skyscraper we wanted to use primal colors. We made the walls of the core tower yellow, by using an image of yellow wooden slats as the texture. The spiraling floors have walls made out of blue concrete. We introduce the color red by placing red windows on the walls of the spiraling floors and on the yellow core tower.

City

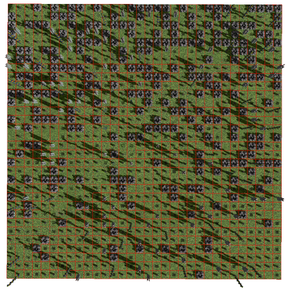


Approach

The first step of generating our city was increasing the grid size. The grids were too small for the office building, they need a minimum width and length of 33 units. This means that the other buildings would have a lot of green space around them. This may not be optimal for a city but since this problem did not have priority, we decided to leave it this way.

The second step of generating the city is updating the buildings.py and the components.py with our own classes and textures. We have made one big asset folder for generating the city. We placed this folder in the same directory as all the python files needed to generate the city, this way it is very easy to access a texture file: ./assets/TEXTURE_NAME.jpg.

The next step is updating the city.py, by replacing the code in the constructing_building function by adding the grid_config needed for generating the offices and adjusting the floors_nums for the other buildings:



```python
def construct_building(self, row: int, col: int, building_type: BuildingType):
    building = None

    if building_type is BuildingType.HOUSE:
        building = House(self._app)
    elif building_type is BuildingType.OFFICE:
        grid_config = [
        ['0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0'],
        ['0', '0', '1', '2', '2', '0', '2', '2', '1', '0', '0', '0'],
        ['0', '1', '3', '2', '2', '1', '2', '2', '3', '1', '0', '0'],
        ['0', '2', '2', '3', '1', '1', '1', '3', '2', '2', '0', '0'],
        ['0', '2', '2', '1', '1', '1', '1', '1', '2', '2', '0', '0'],
        ['0', '0', '1', '1', '1', '2', '1', '1', '1', '0', '0', '0'],
        ['0', '2', '2', '1', '1', '1', '1', '1', '2', '2', '0', '0'],
        ['0', '2', '2', '3', '1', '1', '1', '3', '2', '2', '0', '0'],
        ['0', '1', '3', '2', '2', '1', '2', '2', '3', '1', '0', '0'],
        ['0', '0', '1', '2', '2', '0', '2', '2', '1', '0', '0', '0'],
        ['0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0'],
        ]
        building = Office(self._app, 3, grid_config=grid_config)
        building.building.set_transform(Mat4.from_translation(Vec3(0, 0, 0)))
    elif building_type is BuildingType.HIGHRISE:
        building = Highrise(self._app, 10, 3)
    elif building_type is BuildingType.SKYSCRAPER:
        building = Skyscraper(self._app, 30, 3)
    elif building_type is BuildingType.PARK:
        building = Park(self._app)

    self._plots[row * self._plots_per_col + col] = building
```

The following step was making sure every building was generated the correct amount of times in the city. This is done with this piece of code in city.py:

```python
total_plots = self._plots_per_row * self._plots_per_col
skyscraper_percentage = 5
highrise_percentage = 8
office_percentage = 25
house_percentage = 37
park_percentage = 15

skyscraper_count = int(total_plots * skyscraper_percentage / 100)
highrise_count = int(total_plots * highrise_percentage / 100)
office_count = int(total_plots * office_percentage / 100)
house_count = int(total_plots * house_percentage / 100)
park_count = int(total_plots * park_percentage / 100)

building_types = (
    [BuildingType.SKYSCRAPER] * skyscraper_count +
    [BuildingType.HIGHRISE] * highrise_count +
    [BuildingType.OFFICE] * office_count +
    [BuildingType.HOUSE] * house_count +
    [BuildingType.PARK] * park_count +
    [BuildingType.EMPTY] * (total_plots - skyscraper_count - highrise_count - office_count - house_count - park_count)
)
```

Optimizer

Writing the optimizer.py was the trickiest part of this exercize. We made some rules that we wanted to adhere to:

1. In tiles next to skyscraper = no skyscraper
2. In tiles next to residential building = atleast one office
3. In tiles next to residential building = atleast 1 park

The first step is a counting function that counts how many skyscrapers have another skyscraper in the tiles next to them. This counts up to a score. In the next step the skyscrapers that have another skyscraper in their area will be swapped with a building that is not a skyscraper. This is a random process. When the swap is completed the code checks again if there are still neighbouring skyscrapers in the direct facinity. If not the code will be 0 and the skyscraper swapping will not continue. This usually happens after 1 or 2 steps so it is a very quickprocess.

We have started on implementing the second rule in the optimizer. But right now it will not complete the process and it keeps swapping offices with random buildings that are no offices or residential buildings (skyscrapers + highrise). So there is still room for improvement here.

We still want to improve our code by completing the optimizer and we also want to implement some random building generation mechanism in the office class. This can be done fairly easy because we already have the grid_config tool. All we have to do is to randomize each integer in the grid_config grid with a range of 0 to 3