Converting 2-Dimensional Carbon Dioxide Modelling into 3-Dimensional Space

Last modified: July 16, 2020

Changes to the CO_2 Model's Code

The CO_2 model's C++ code did not require modification in the process of transitioning to three dimensions. Cadmium generates N-dimensional neighbourhoods which the CO_2 model uses. Each cell obtains its CO_2 concentration by averaging the concentration of its neighbours. Since the neighbourhood is N-dimensional, obtaining the neighbourhood is already handled. If the model is given a 3D environment, Cadmium will generate neighbourhoods in three dimensions.

Changes to the Environment Models

Previously, the files describing the environment represented a 2D space, making modelling the walls, floor, ceiling, and the heights of different elements unnecessary. By shifting the model to three dimensions, these things must be taken into consideration. Adding these components manually would take a significant amount of time and would be extremely prone to errors. Instead, there is now a Python script that is capable of making a rough conversion from 2D space to 3D space.

Python Scripts

There is one script that is necessary to convert an image to a JSON file ("convert.py"). The script is capable of creating 2-dimensional and 3-dimensional models. There is another script ("make3D.py") which can convert 2D models that are already in JSON format to 3D JSON format.

Image to JSON (convert.py)

Configuration

The configuration file ("config.json", by default) contains three sections: files, model, and image.

The *files* section determines where the input image is located and where the resulting JSON will be saved.

The *model* section provides details about how the model will be constructed. Importantly, much of the information in this section pertains to how the model will be converted from 2D to 3D. The *height* parameter sets how many cells tall the model will be. A height of 14 means that the model will be 14 cells tall. Since each cell is $25cm \times 25cm \times 25cm$, this corresponds to a total height of 3.5 metres. However, the room itself will be 12 cells (3 metres) tall as the floor and ceiling are included in the height value. **Setting this value to be 1 will cause the script to produce a 2D model.** The *walls_only* value, if set to true, will ignore all elements that are not of type -300 (IMPERMEABLE_STRUCTURE). Typically, this value should be false. The *heights* object sets where the specified elements will be placed on the new axis. For example, *door_top* sets highest cell that the door will extend to. Note that the highest possible value for these parameters is $height - 1$ since the first layer is at $z = 0$.

Within the *model* section, there is a subsection called *counter*. This area determines the *counter* value of each WORKSTATION (type -700) cell. The *counter* attribute of a cell determines when a WORKSTATION cell will become a CO2_SOURCE (type -200). The *seed* value is the seed for the random number

generator. The *minimum* and *maximum* fields specify the minimum and the maximum possible values for the *counter* value (both inclusive), respectively.

The *image* section in the configuration file contains information on how the script will convert different colours and the tolerance (how different a pixel in the image may be from a solid colour) that it will permit. The current default is as follows:

Colour	String	Cell Type	Type Number
white	255,255,255	AIR	-100
cyan	0,255,255	CO2_SOURCE	-200
black	0,0,0	IMPERMEABLE_STRUCTURE	-300
green	0,255,0	DOOR	-400
yellow	255,255,0	WINDOW	-500
blue	0,0,255	VENTILATION	-600
red	255,0,0	WORKSTATION	-700

Usage

The script uses a Python library module called Pillow (imported as "PIL" in the script). This module requires separate installation which can be done using pip. The module can be installed with the following command for Python 3:

```
pip3 install Pillow
```

Note Bash (Cygwin) can install the module, but may cause problems when getting Python 3 to find the module. However, using Windows Command Prompt, Python was able to use the module.

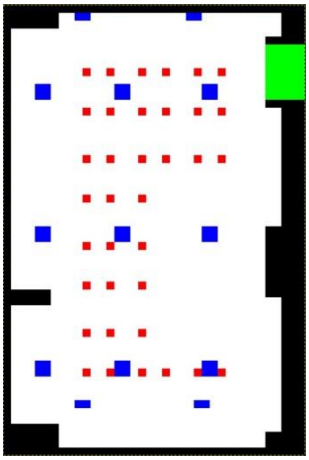
The script itself has the following usage:

```
Bash: python3 convert.py [<config file>]
CMD:  py convert.py [<config file>]
```

The command line argument is optional. It specifies the configuration file for the converter to use. If the argument is missing, the script will look in the same directory as itself for a file titled "config.json". The script will output a file containing the model in JSON format.

Creating Images

The script works on a per-pixel basis. This means that each pixel is considered to be a cell. For a model with the shape $25 \times 30 \times 14$, the image should be 25 pixels by 30 pixels. Using an image manipulation program such as GIMP allows for the necessary amount of control. Programs such as Microsoft Paint also work but have a limited zoom and have no ruler.



It is important to note that an uncompressed image format should be used (such as a bitmap). Using compressed formats (JPG, etc.) can result in pixels changing colours and blurring.

The picture to the right is an example of a model before being converted. It uses the default colour values.

2D JSON to 3D JSON (make3D.py)

Configuration

Configuration files from the other script can be used here as well. A notable difference is that the image section of the configuration file can be omitted (although there is no harm in leaving it in the file).

Usage

The usage of the “make3D.py” script is similar to “convert.py”. The difference is that the input file (specified in the configuration file) must be a 2D JSON model.

```
Bash: python3 make3D.py [<config file>]
CMD:  py make3D.py [<config file>]
```

As before, the script will check the directory that it is in for a file titled “config.json” if not provided a path to a configuration file via the command line.