How blocks are stored

Romain PEREIRA

3rd November 2017

Summary

Preamble

This file is part of the Voxel Engine project. It is here to document the project, and to explain how the program was conceived. It is not a formal scientific paper, but we should try to stick to this style, to make it more readable and interesting. Thank you for your attention, have a nice reading.

1 Abstract

In order to store our voxels, we need a data structure. As the software should be dealing with million of blocks, this data structure has to be 'memory-friendly', or the whole program won't be able to run. That is what 'Terrain' are: a data structure.
The approach assumes that we need to store in RAM informations about each loaded blocks, not less, not more. These informatons are numerous: block behaviour (transparency, liquid, density, light emitter...), block rendering (lighting, mesh
Final results were definetly acceptable: the programs need 4 bytes per block. On a 1GB virtual machine, this would mean being able to fastly access more than 250 000 000 blocks. (which is the number of water drop on a $20m^3$ container.)

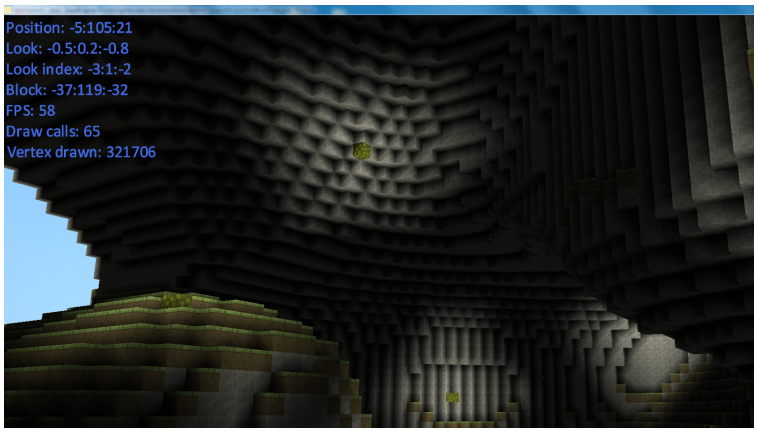

Figure 1: *Illustration of the final result*

2 Introduction

3 Materials and methods

3.1 Memory representation

3.1.1 Naive approach

Each blocks need attributes: textures, densities, light value, materials, amount of water...
If we store each blocks in it own instance with these attributes, the amount of memory per block (in bytes) would be:

$$S_b = pointer + textures + density + lightvalue + ... >= 40 \text{ bytes}$$

where:

- pointer: pointer to the block memory address (8 bytes)
- textures: textures integer id for each block faces (6 faces * 4 bytes per faces)
- density: float 4 bytes
- light value: float 4 bytes

... and many other imagineable attributes.

However, many blocks shares constants attributes. (each 'stone' block have the same density for example)
Moreover, Most of the block ($> 80\%$ of them) will be dirt, stone, leaves... These blocks have many shared constants attributes. So, instancing is the key.

3.1.2 Instancing

Let us seperate blocks in two categories: the one that don't need their own instance A), and the one who need it own instances B).

In category A, we find for instance: dirt, stone, leaves, logs, plants...

Each of these blocks need: 1 integer (which will be codded on 2 bytes in practice), representing the In category B, we find liquids (each blocks has it own amount of liquids), or any dynamic blocks that need it own 'per block' attributes.
NB: B inherits A

3.1.3 Terrain

A 'Terrain' simply is a $u * v * w$ 4 bytes-array, which hold blocks. Moreover, it contains a 3 integers 'world-relative' index (ix, iy, z), so we can change from 'Terrain' basis coordinate system to global 'World' basis.
TODO: 2D terrain scheme

The memory usage of a terrain (which can hold $u * v * w$ block from category A)) is:

$$S_t = 4 * u * v * w + o(u * v * w)$$

where:

- $o(u * v * w)$ is the size of every per-Terrain attributes

and so, per block:

$$S_b = 4 + o(1) \simeq 4.1 \text{ bytes (in practice)}$$

To optimize memory access (reading and writting), the 3 dimensionals array was flattened, see: https://stackoverflow.com/questions/2512082/java-multi-dimensional-array-vs-one-dimensional

3.2 sub section 2

4 General conclusion

References

[1] PEREIRA Romain, *Source code*,
https://github.com/rpereira-dev/VoxelEngine.git,
The main repository of the project.