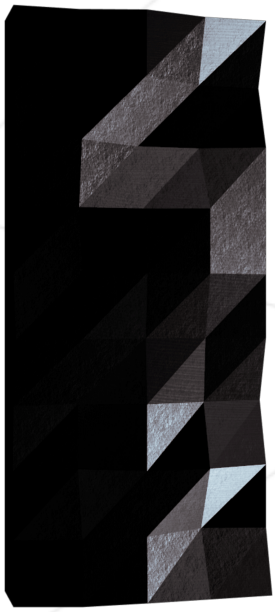


# Ft_vox

## Pimp my blocks

*Summary: This project is the first step to the creation of your very own Voxel World!*

# Contents

# Chapter I

# Preamble

By Mike Acton on March 14, 2008 9:44 PM

One of the things we talked about this year at GDC was what we called the "Three Big Lies of Software Development." How much programmers buy into these "lies" has a pretty profound effect on the design (and performance!) of an engine, or any high-performance embedded system for that matter.

(LIE 1) SOFTWARE IS A PLATFORM

I blame the universities for this one. Academics like to remove as many variables from a problem as possible and try to solve things under "ideal" or completely general conditions. It's like old physicist jokes that go "We have made several simplifying assumptions... first, let each horse be a perfect rolling sphere..."

The reality is software is not a platform. You can't idealize the hardware. And the constants in the "Big-O notation" that are so often ignored, are often the parts that actually matter in reality (for example, memory performance.) You can't judge code in a vacuum. Hardware impacts data design. Data design impacts code choices. If you forget that, you have something that might work, but you aren't going to know if it's going to work well on the platform you're working with, with the data you actually have.

(LIE 2) CODE SHOULD BE DESIGNED AROUND A MODEL OF THE WORLD

There is no value in code being some kind of model or map of an imaginary world. I don't know why this one is so compelling for some programmers, but it is extremely popular. If there's a rocket in the game, rest assured that there is a "Rocket" class (Assuming the code is C++) which contains data for exactly one rocket and does rocketty stuff. With no regard at all for what data transformation is really being done, or for the layout of the data. Or for that matter, without the basic understanding that where there's one thing, there's probably more than one.

Though there are a lot of performance penalties for this kind of design, the most significant one is that it doesn't scale. At all. One hundred rockets costs one hundred times as much as one rocket. And it's extremely likely it costs even more than that! Even to a non-programmer, that shouldn't make any sense. Economy of scale. If you have more

of something, it should get cheaper, not more expensive. And the way to do that is to design the data properly and group things by similar transformations.

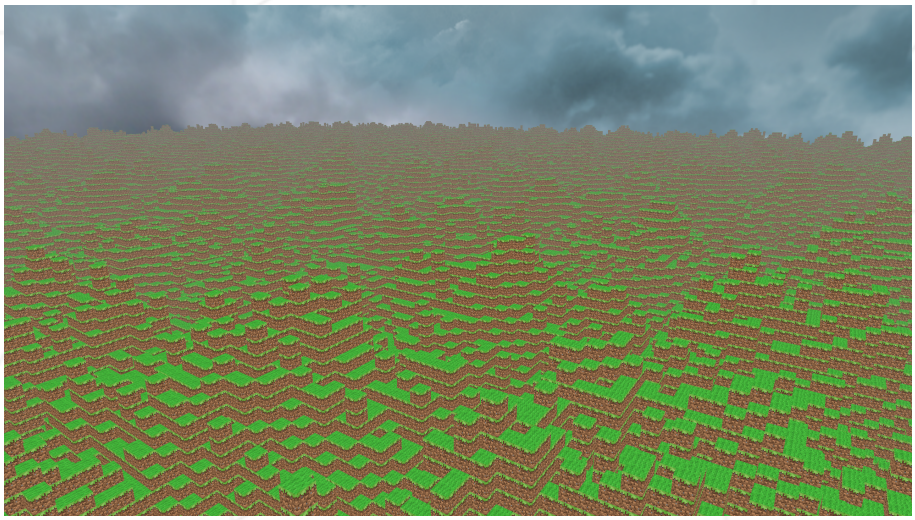
(LIE 3) CODE IS MORE IMPORTANT THAN DATA

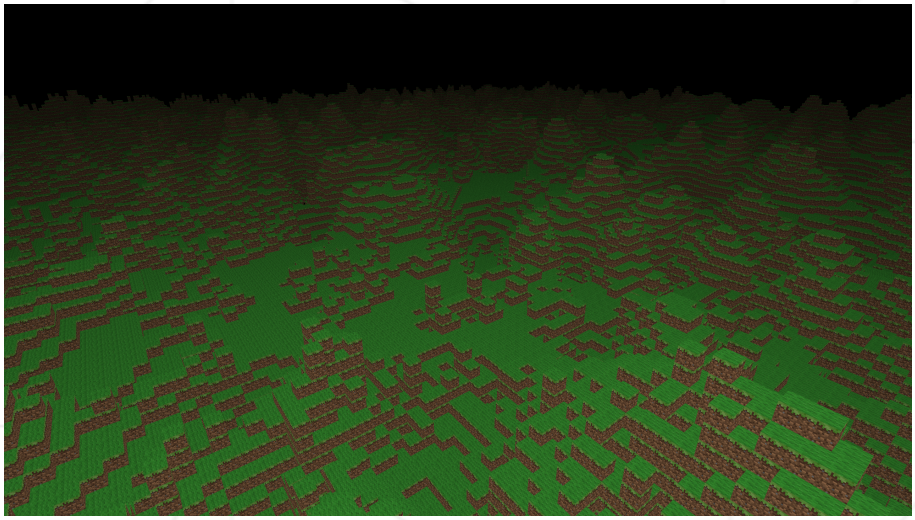
This is the biggest lie of all. Programmers have spent untold billions of man-years writing about code, how to write it faster, better, prettier, etc. and at the end of the day, it's not that significant. Code is ephemeral and has no real intrinsic value. The algorithms certainly do, sure. But the code itself isn't worth all this time (and shelf space! - have you seen how many books there are on UML diagrams?). The code, the performance and the features hinge on one thing - the data. Bad data equals slow and crappy application. Writing a good engine means first and foremost, understanding the data.

# Chapter II

# Introduction

Welcome to the beautiful world of voxels, where you will use all the benefits of the abstraction "the whole world is a 3 dimensions grid" in order to display a daunting number of things on a screen, travel in a gigantic procedural universe and a completely malleable playground. What are voxels?





# Chapter III

# Objectives

This project aims to confront you to a graphic project that will be extremely demanding in terms of optimization. You will have to study the characteristics of the voxel worlds, and use them along your infographics knowledge to display a lot of elements on screen. Thus, you will have to study different algo/opti to obtain a SMOOTH render (there are many of them). You will also have to manage your memory and data structures properly to be able to travel in a very, very large universe. Once you have achieved that, you will be ready to get to the next level with the project ft_minecraft, which will be even more demanding.

# Chapter IV

# General instructions

- You're free to use you language, but keep an eye on its performances. (If you can't choose, c/c++/rust are suggested).
- You will use openGL or openCL for your GPU calculations. You can also use Vulkan to replace OpenGL/OpenCL. You cannot use a library to work for you.
- You can use a library to load 3D objects and pictures, a windowing library and a mathematics library for your matrix/quaternions/vectors calculations. You must not push them in your repo. Instead, you must write your own download/install scripts.
- The render should always be SMOOTH. This means if your assessor considers your game offers an unpleasant visual experience, he can give you a 0.
- Any crash (Uncaught exception, segfault, abort ...) will disqualify you.
- Your program must be able to run for hours without eating the whole memory or slowing down. Manage your RAM as well as VRAM very carefully.
- Your program will have to run in full screen mode. Reduce the default frame buffer is prohibited.

# Chapter V

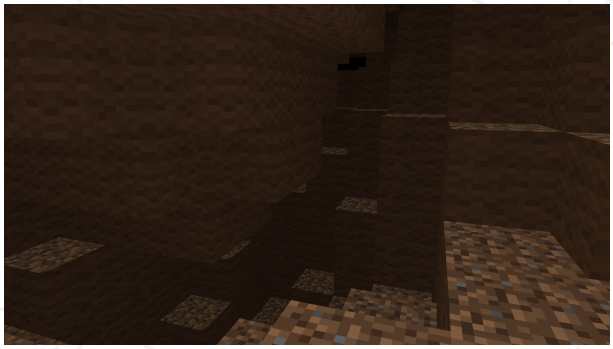
# Mandatory part

## V.1 The world

You must be able to create a very large procedural world. For this project, user should be able to visit at least 16384*256*16384 cubes (256 is the height).

`The ft_minecraft project will be WAY bigger !`

Some cubes may be empty, others can have different types, like grass, ground, sand, etc...
Except for the empty cubes, they will all be opaque but will have their own textures.
You must implement a method to generate terrains such as hills, mountains or caves when the user goes underground. This generation has to be determinist, which means the same seed will spawn the exact same map.
The terrain should have a remotely natural topography, with hills and/or mountains, caves and so on... A simple rand() will not be tolerated.
Each visited piece of terrain must be saved in the memory up to some limit you will set yourself and after which you can start deleting cubes from the memory.



## V.2 Graphic rendering

Cubes must also be displayed on screen. In the open, minimal distance render will be 160 cubes. So you have an idea, the distances is 320 in the first picture of the introduction. Of course, you don't have to show the cubes if they're hidden.
Each cube mus be textured, and you must have at least 2 different textures and 2 different types of cubes. Make sure the FoV is always filled with various elements. The user should never feel lost or confused when scanning the ground level.
Again, render must be smooth. Avoid the freeze frames at all costs.
FoV must be 80 degrees.
To make it a little nicer, you will set up a skybox. Don't leave any artefact on its junctions.

`If you want your render to run smooth, you should manage the workload so it is equally shared between the CPU and the GPU.`

## V.3 The camera

If we can travel in your game, this would make for a nice touch. You must configure a nice little camera. The mouse must be able to control it on 2 axis at least and you will set 4 keys that will make it go forth, back, right and left in according to the camera rotation. Of course, the user must be able to keep going if he keeps pressing a key. The camera speed should be set around 1 cube/second, but for the evaluation, you will create a key that will multiply this speed by 20.

## V.4 To sum it up:

- A gigantic terrain made of textured cubes of different types.
- A luxurious FOV.
- An advanced procedural generation offering a natural face to the terrain (hills, mountains, caves, lakes...).
- An intuitive camera.
- A skybox.

# Chapter VI

# Bonus part

I bet you're dying to add physics, a player, green explosive monsters that will ruin hours of painstaking work. But this will come with the ft_minecraft project. Right now, you will only focus on the technical side.

- Have a render distance always higher than 14 and always have a smooth display.
- A fps counter is displayed.
- Render is smooth and doesn't freeze, event at x20 speed.
- Being able to delete blocks with the mouse.
- Having a lot of different biomes.

# Chapter VII

# Turn-in and peer-evaluation

As usual, turn in your work on your repo `GiT`. Only the work included on your repo will be reviewed during the evaluation.