Re-Engineering 3D Engine With Linear Algebra

# What is this?

This is a project guide that assist you to re-engineer a first-generation 3D rendering engine with basic linear algebra concepts. You will be surprised by how **straightforward** and **simple** it is – it does not contain a lot of code, or a tons of math concepts; with a little bit of knowledge about **mathematic modelling** and **linear algebra**, you’ll be able to construct the whole rendering engine from pieces!

# Few words about history……

Do you know that the first 3D video game ever was produced as early as 70s?

The first 3D video game ever was *Maze* – It looks like this:

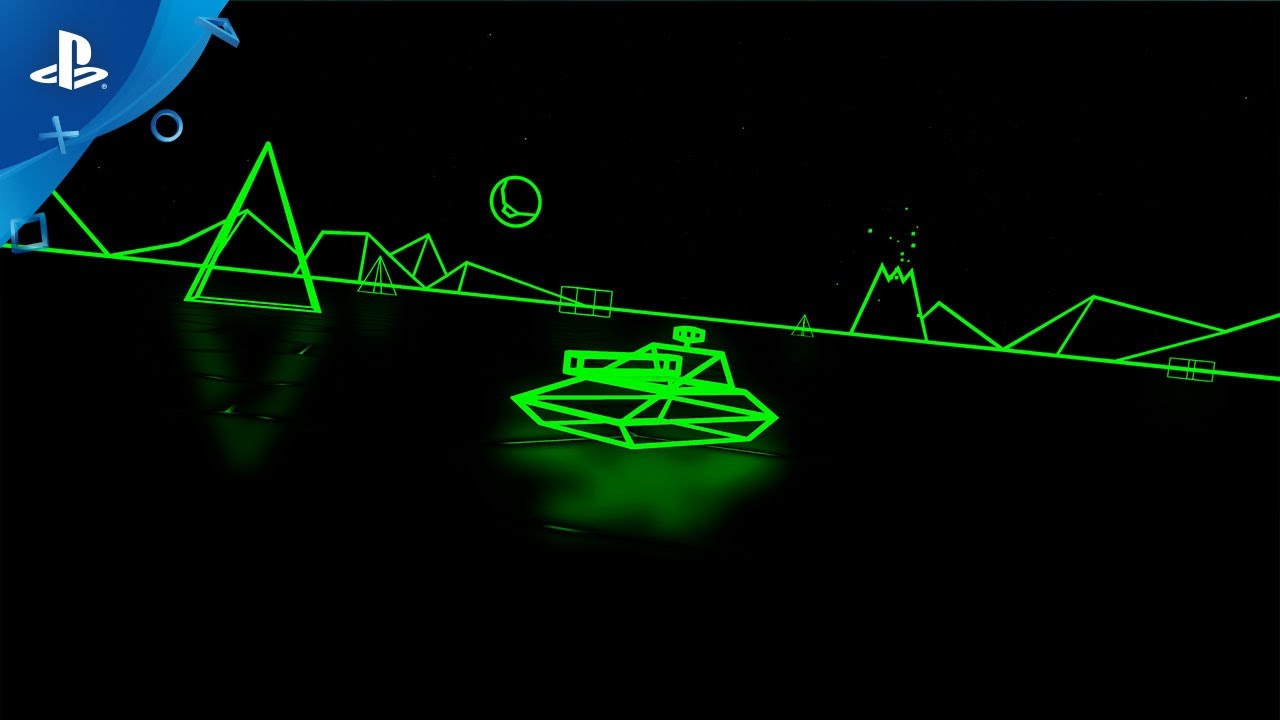


↑MIT version of *Maze* on an Imlac PDS-1D at the [Computer History Museum](https://en.wikipedia.org/wiki/Computer_History_Museum), from [Image Source: upload.wikimedia.org](https://upload.wikimedia.org/wikipedia/commons/f/f4/Maze_war.jpg)

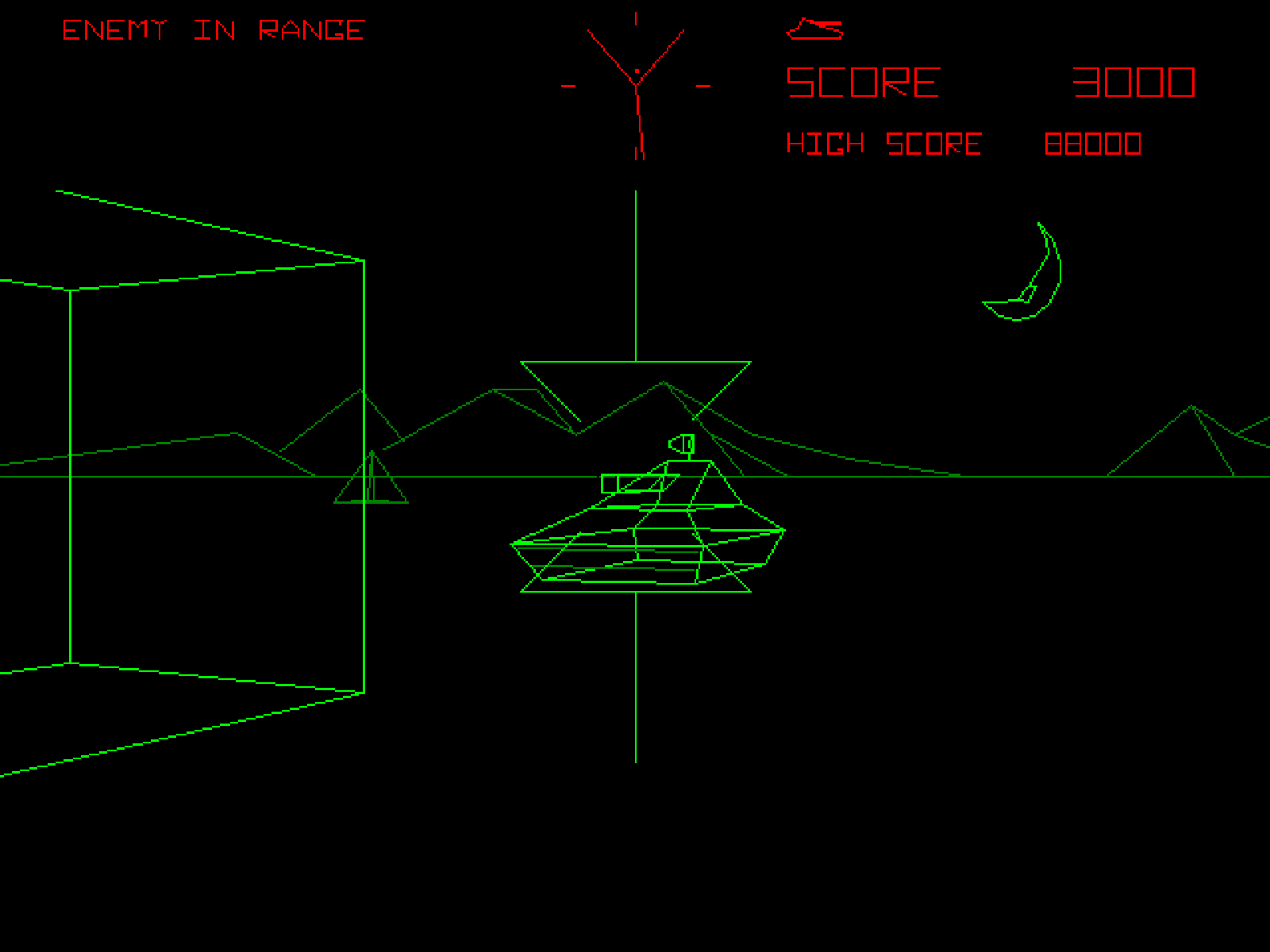
It was made in 1973 and is a first-person-shooter-game, which players will be shooting each other (yup, it supports multiplayer) in a 3D maze.

It is extremely primitive – there are only series of grids that represents walls, not texture or even image was included; the player could only turn 90-degree angle, etc. Though they didn’t stop it from being a popular game. You can find out more things about it [here](https://en.wikipedia.org/wiki/Maze_(1973_video_game)#Legacy) at Wikipedia.

Series of commercial 3D games were then released in the 80s, like [*Battlezone*](https://en.wikipedia.org/wiki/Battlezone_(1980_video_game)), which sold 15,000 copies (which means we could also be millionaires if back to 80s with this project done!)



*Battlezone* Classical mode remastered at PS VR, from [PlayStation Blog](https://blog.playstation.com/2016/12/15/battlezone-gets-classic-mode-update-on-dec-20/)



A screenshot from original *Battlezone*, from [Time](https://techland.time.com/2012/11/15/all-time-100-video-games/slide/battlezone-1980/)

# What do I need for making this 3D engine?

We’ll need some knowledge from Linear Algebra – since this is an applied math project. You need to know:

1. What is a **vector**
2. What is a **matrix**
3. What is **matrix-vector multiplication**

And, of course we’ll need knowledge from basic algebra, like trigonometry.

Programming wise, since this is intended to be an applied math project instead of a coding lab, I’ll be offering as much instructions as I can to make the coding part easier, and external links to potentially helpful third-party resources as well.

Hardware wise, you’ll need a PC (or something else that can run code) with OS that supports **Java** and its IDE (the integrated Development Environment – the nerdy name for a fancy text editor). We’ll make the engine with Java since it’s the first programming language of a good number of high school / college students, though the mathematical concepts potentially work for any other computer language as well. Feel free to try them out later.

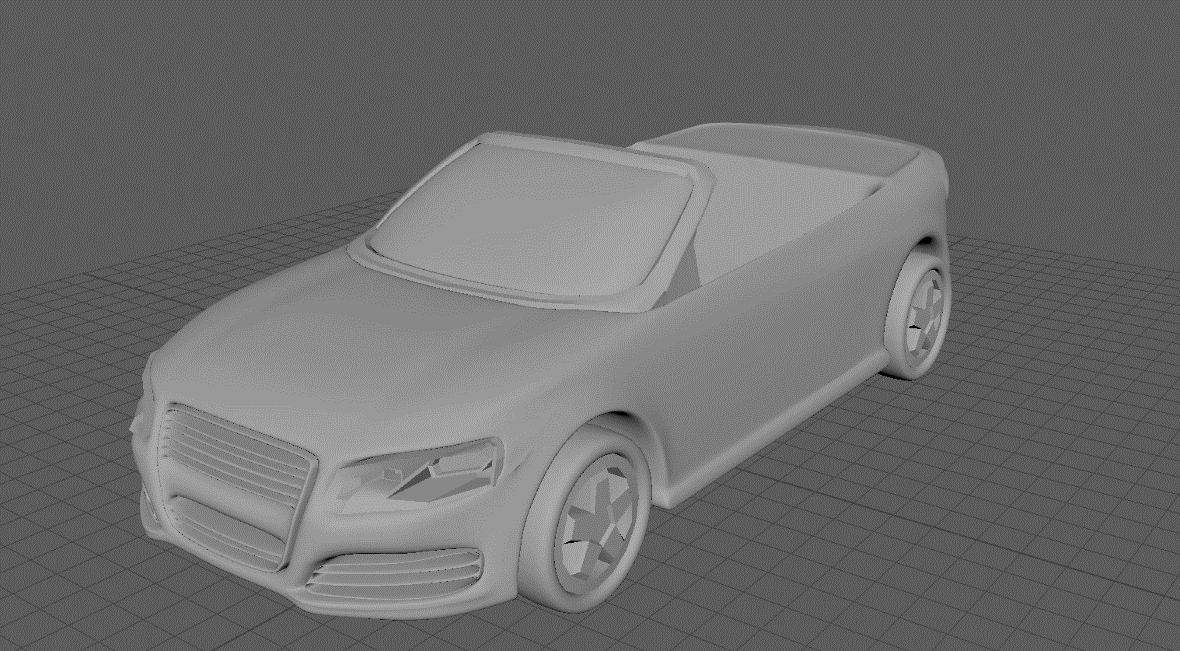
When you are ready, flip this page and we’ll begin!

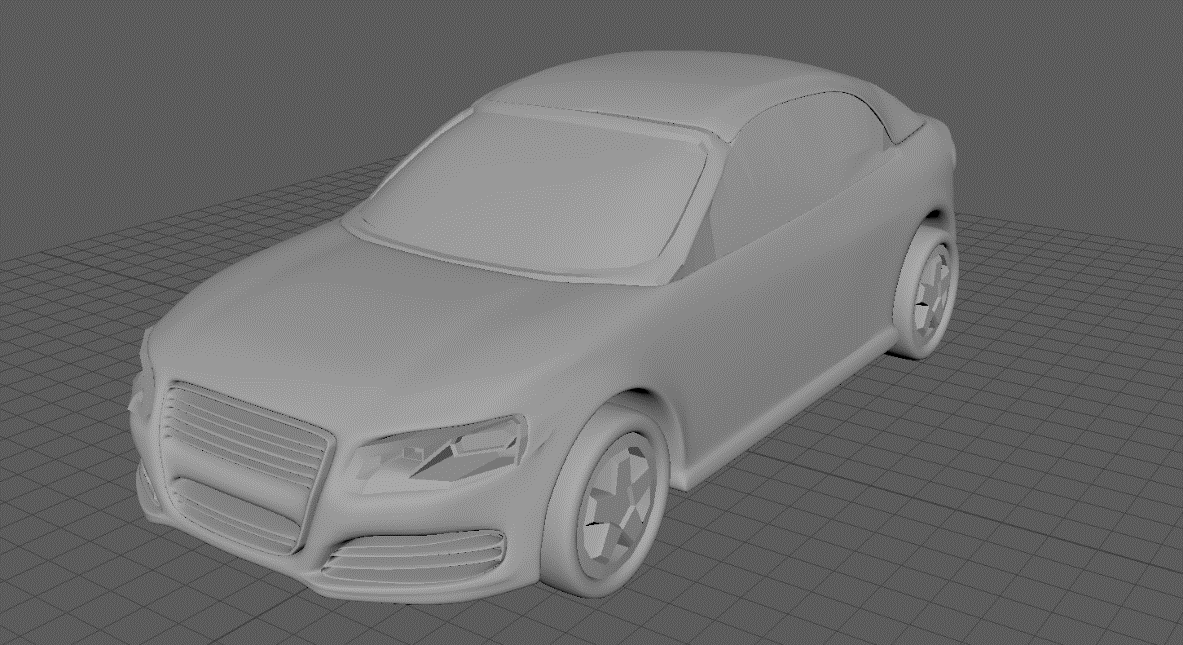
# How do I store a 3D shape?

Or in other words: How do we use number to represent a 3D shape?

In case you have taken a multivariable calculus class, of course we can describe a shape with 3-variable polynomials; that will yield perfect curves and save your disks.

However, won’t it be a bit too hard for artists who want to produce complex shapes? What about rigged models (which change shape according to code or “skeleton”) made for animations? What about the real-time compute power consumed by calculating a whole system of polynomials?





↑Think about this car – how much time is it going to take to use a system of polynomials to describe it, and how much compute power will it take when rendering?

The Vertices

What about starting with a lower dimension? Let’s say, a dot (or point).

It’s easy to use a 3-variable vector to represent a point in 3D space:

The position of this point is probably x unit displacements on direction, y unit displacements on direction, and z unit displacements on direction from origin.

Then, what about several points within the same 3d space? You probably have got it already; it’ll be a set of 3D vectors.