Project Documentation. 20.4.2019

3D Engine

716051 Tietotekniikka

Teemu Saravirta vuosikurssilta 2018

Abstract

The project is a from-scratch 3D engine running on Scala FX, written in Scala. Scala FX is mostly a re-skin for Java FX and the two are both functional inside the app. I have taken some functionality from Java FX, regarding the parts in which Scala FX documentation was sparse. The engine is fully scalable with different 3D objects, as long as the data is convertible to the format of three 3d points on a CSV row, each consisting of (x,z,y)-coordinates forming a triangle in a clock-wise order. The project was in my opinion executed on hard-difficulty as the scalability is virtually limitless, and the engine is able to handle all kinds of different shapes.

Differences from the plans

Instead of 2 to 3 coordinates per row, I used 9 to form a single triangle per CSV row. AD-keys are used to move sideways instead of turn, which is entirely handled by the mouse.

The class structure is vastly different as in plans to accommodate the different problems risen during development. More on this later, in part *Program Structure.*

Minimap was scrapped as non-integral part of the program, and mostly redundant, as information is attained with more clarity from the camera.

I’ve decided to also scrape all dependencies regarding calculations and the project runs entirely with my self-defined vector and matrix math. Only real dependency is Scala FX, which requires careful installation, but a ready-made jar file is included.

FOVSelect was a great start to the problem of having objects behind camera field of view, but it required a lot of expansion to get working.

Sources

<https://stackoverflow.com/questions/5666222/3d-line-plane-intersection>

<https://stackoverflow.com/questions/328107/how-can-you-determine-a-point-is-between-two-other-points-on-a-line-segment>

<http://www.ambrsoft.com/TrigoCalc/Plan3D/PlaneLineIntersection_.htm>

<http://www.songho.ca/math/line/line.html#intersect_lineline>

<https://stackoverflow.com/questions/5666222/3d-line-plane-intersection>

<http://eguruchela.com/math/Calculator/shortest-distance-between-point-plane>

<https://stackoverflow.com/questions/6615002/given-an-rgb-value-how-do-i-create-a-tint-or-shade>

<https://www.youtube.com/channel/UC-yuWVUplUJZvieEligKBkA>

4. Algorithms

The key algorithms of the project are the matrix calculations made in VectorVer and Matrix classes, the 3D planar projection, as well as intersection math, and the sorting, filtering and converting of the drawable shapes.

Algorithms under following classes:

Camera

Camera handles the rotation and the moving of the player in the world. The rotation is achieved with Tait-Bryan angles which allow as to look left-right , up-down as well as lean to the sides. As well all the other algorithms my version uses (x,z,y) sequence with behore-mentioned order of rotation.

Located under Front is the amount of turning, which I’ve decided on to be fractions of 2 times Pi to achieve regularity, and of course times sensitivity to allow user customization without touching the code.

Camera also handles moving in conjuction with Front, and to achieve precision the moving is inversely proportional to the tickrate of the AnimationTimer, because if the distance moved was always the same, the player would move more slowly when the tickrate is lower. Now the algorithm adjust the player to move more when the framerate is lower and compensate by lowering the speed when it is higher.

Front

A lot of required functionality is achieved with the help of the ScalaFX and JavaFX toolboxes, but some aspects of the program were not easily handled by them, and I’ll mention some of them here. First of all the program can be run in full-screen mode, or in any customisable aspect ratio the user wishes. I’ll tell about the re-scaling and re-sizing of content in [--------------------], but the way I’ve achieved the aspect ratio is as follows. I’ll get the size of the screen, and if the base