15/05/2017

3D TANK MAZE GAME

Computer Vision & Graphics Assignment

Eva Esteban

Abstract

This report describes the implementation of an interactive 3D computer game developed in OpenGL using the C programming language. It consists of a virtual tank which is moved along a maze by the user. The aim is to collect all the coins within the given time limit without falling off the edges. The different techniques used are outlined in this report, as well as the method applied, the testing performed and the obtained results.

1. Introduction

The aim of this project is to develop an interactive game which consists of navigating a tank using the keyboard and mouse with the objective to collect the coins that appear on screen within a set time. It incorporates a source code, which contains a set of functions alongside the main function, and both a vertex shader and a fragment shader. These are responsible for the different vertices and the colour of the pixels between them. A set of comments are incorporated to make debugging easier and guide the user through the code.

2. Programme description

Methodology

The proposed programme is compiled by executing the command **qmake** once in the terminal, followed by **make** every time changes are made to the code. It is run by typing **./TankAssignment < input.txt** where it reads the data from the file **input.txt**. Three input files **input1.txt**, **input2.txt** and **input3.txt** are provided to account for different difficulty levels of the game. The programme will display an error if the input file is empty or incorrect.

The game is implemented following the computer graphics pipeline shown on **Figure 1**. The transformations performed on different objects are performed by concatenation of 4x4 homogeneous translation, rotation and scaling matrices. Objects are projected on the image plane using perspective projection, and observed from two different camera perspectives: far from the tank and view from the tank.

Both specular and ambient lighting effects are included in the shaders to account for shadows and transparency, giving the game a more realistic effect. In addition, the option to change the brightness and background colour of the screen to different shades of blue is provided.

At the beginning of the game, an audio message is played. It explains the different controls for the tank as well as the aim of the game and how to access the menu and full screen mode. Moreover, heads up display includes the time left as well as the score.

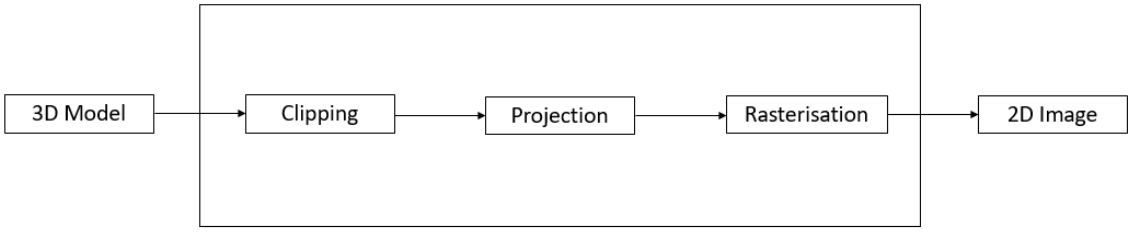

Figure 1: Computer graphics pipeline

Implementation

The following functions are included in the source code:

- **initGL:** Initialises the screen to the chosen dimensions, setting the appropriate parameters of the programme. It takes no parameters and returns nothing.
- **initShader:** Function to initialise the shader and apply the corresponding shading to the rendered objects.
- **InitTexture:** Reads the texture file provided as a parameter in .bmp format and applies it to the corresponding object.
- **display:** Function which is called continuously and outputs to screen the different objects with their associated textures at different positions, as well as the background colour, the time, the score and a set of messages for the user.
- **keyboard:** Detects when a key is pressed and executes the tasks associated with that specific key. It takes the key and coordinates as input parameters, returning nothing.
- **keyUp:** Detects when a key has been release, executing the lines of code associated with it.
- **handleKeys:** Allows for key buffering so that several keys can be pressed and the same time with their associated tasks being performed alongside each other. It does not take any input parameters and returns nothing. The keys **a**,**w**,**s**,**d** or **A,W,S,D** will determine the magnitude and direction of the movement of the tank. The keys **y**,**u,i** or **Y,U,I** are used to change the background colour and lighting of the scene. In addition, pressing **Esc** will exit the game, pressing **p** or **P** will make the game full-screen and **o** or **O** will reverse that action.
- **mouse:** Controls the mouse interaction. Pressing the right button will switch between camera perspectives far and heads up, while pressing the left button will make the tank fire a ball. Moreover, the middle button is linked to a menu that allows the user to pause or exit the game.
- **motion:** Function to control the turret movement by tracking the passive motion of the mouse on screen.
- **updateTank:** Sets the position of the tank on screen at every moment. The tank moves with an accelerated movement determined by the keys pressed by the user.
- **drawTank:** Function to render the different parts of the tank to their appropriate position on screen with their assigned textures.
- **checkTank:** Checks the position of the tank on screen at every point. If the tank is on an empty space in the maze, it will fall. If it is on a cube with a coin, it will collect it and the score will be incremented by one.

- **tankFall:** Sets the position of the tank at every moment whilst falling. The tank falls following an accelerated movement determined by gravity.
- **drawBall:** Function to render the ball at different positions on screen.
- **fireBall:** Fires a ball from the turret and resets the parameters so that another ball can be fired afterwards. The fired ball's movement is an accelerated one determined by gravity.
- **setBall:** Function to set the ball initial position at the far end of the turret.
- **ballCollision:** Tests if the ball collides with a coin. If it does, the coin is removed from the scene and the score is incremented by one.
- **render2dText:** It outputs the input string to the screen. It takes a string as a parameter, altogether with the values for the colours of the text i.e. red, green and blue and its position on screen.
- **createMenu:** Function to create a pop up menu when the middle button of the mouse is pressed that allows the user to pause or exit the game
- **menu:** Establishes the function of each of the menu options defined in **createMenu** and selects the appropriate one by receiving the option number as an input parameter.
- **main:** Main programme entry which reads the input file creating the maze accordingly, and loads the OBJ files and textures. A check is performed to ensure the file is not empty before opening it. Once the programme has finished, the file is successfully closed. This function is executed once and will output an error if the input file is empty or corrupted.

3. Testing and results

The programme functionalities have been tested using keyboard and mouse interaction. The following images show the rendered scene before and after some moves have been performed by the user.

Figure 2, shows the game immediately after it has been started. The score is initialised to zero and the time, shown in seconds, is initialised to maximum. The scene is presented with a medium blue colour and medium brightness.

Figure 2: Rendered scene at the beginning of the game

In **Figure 3**, the tank has collected the coin that appeared in front of it, increasing the score by one. The background colour has been changed to light blue by pressing **y** or **Y** on the keyboard, which has increased the brightness of the scene by a factor of 0.5. In addition, the user has moved the mouse on screen so the turret is pointing at where the mouse pointer is, with the camera following the turret. The time left has decreased and the pop up menu has appeared on screen due to the user pressing the middle mouse button.

Figure 3: Rendered scene in the middle of the game

A warning is printed on screen when there are less than four seconds left. This is shown on **Figure 4**, where the user has decreased the scene brightness by a factor of 1.0 relative to **Figure 3** by pressing **l** or **L**, and changed the background colour to dark purple. This figure shows the heads-up display provided when pressing the left mouse button. The score has increased to two after the user fired a ball in the direction the turret was pointing and collected a coin.

Figure 4: Rendered scene near the end of of the game

Once the tank falls or the time is over, a message will appear on screen indicating the end of the game and instructing the user on how to exit correctly, as shown on **Figure 5,** where the tank has just fallen from the maze and is rotating simulating a realistic fall.

Figure 5: Rendered scene when the tank is falling

If the user manages to collect all the coins before the time is up, a congratulations message will be printed on screen as shown on **Figure 6** and the game will end.

Figure 6: Rendered scene when the player wins

4. Conclusion

Implementing this project has helped me to improve my programming and debugging skills, whilst learning about different OpenGL functionalities and its syntax. It has provided an insight into the 3D computer graphics pipeline and how 4x4 matrices and shaders are used to render objects to screen in their appropriate position and with a realistic appearance. This assignment has also helped me to understand OBJ and BMP files and how these can be read and interpreted by different programmes.

5.References

- Hilton, A. (2016) 'Computer Vision & Graphics', University of Surrey.
- Schreiner D. Sellers G. Kessenich J. Licea-Cane B. 'OpenGL Programming Guide', The Khronos OpenGL ARB Workshop Group.
- http://learnopengl.com