REPORT

Computer Graphics Mini Project

manVcat

Submitted By

Srinag Rao S
16IT141
IV Semester B.Tech IT

Department of Information Technology
National Institute of Technology
Karnataka

May 2018

Certificate

This is to certify that the mini-project titled **“manVcat”** has been presented by Srinag Rao S, a student of second year B.Tech (IT), Department of Information Technology, National Institute of Technology Karnataka, Surathkal, on 3rd May 2018, during the even semester of the academic year 2017 - 2018, in partial fulfilment of the requirements for the award of the degree of Bachelor of Technology in Information Technology at NITK Surathkal.

Place: Surathkal

Date: 3rd May 2018 (Signature of the Examiner)

Contents

Abstract

Your cat was bit by a rabid dog and is now wandering mindlessly in your backyard. The vet is on his way but you, along with your brother Astar, have to contain your cat till then. Don't get too close to it though! Remember, it's rabid. Use your remote controlled tank to neutralize you cat. 9 direct hits and he will go to sleep.

manVcat is a single player 2D shooter where the target is a constantly moving cat. The player controls a remote controlled tank capable of firing 30 tranquilizer shots. The player has a top view of the backyard from his room. The cat has 9 lives and the player has 7. The backyard is also filled with thorny bushes. If these obstacles touch the tank, the tank loses one life per contact. The player gains one point for shooting a bush, 4 points for shooting the cat and lose 5 points when hit.

The tank body can be controlled using WASD keys. The barrel can be rotated using J and L keys and K key fires a shot. To help the you concentrate on shooting, you can ask your virtual brother Astar to control the tank movement for you; simply click where you want the tank to be and let Astar navigate for you, while you help your cat. Press Q to quit the game and reveal you score.

Introduction

This project involves development of graphics tools to draw lines and polygons, design tools to develop models and characters quickly and efficiently, collision detection system and a path finding system.

OpenGl library and platform is used for rendering graphics. The only primitive shape used is the point. All other shapes are drawn using custom algorithms. Lines are drawn using the **drawLineBresenham** function. Lines are drawn using a generalized Bresenham's line drawing algorithm. **drawLineBresenham** is further optimized for case of vertical lines, which improves performance of polygon drawing algorithm. Polygons are drawn using **fillPoly**, a custom implementation of Scan Line polygon fill algorithm.
The function **circle** draws a circle using the Midpoint algorithm for generation of circles. The circle can also be filled if the appropriate flag is marked. A suite of other functions are provided to perform required transformations for example **myPushMatrix**, **myPopMatrix**, **myTranslatef** and **myRotated**.

A modelling system has been developed for efficient design of characters or models. A model is a collection of elements, ie shapes, which intern are a collection of vertices. Creating a model using the model class automatically adds it to the render queue(which manages rendering on display), adds it to the update queue(manages physics of characters), manages it's hitboxes(for collision detection), manages

memory when dead etc. This allows programmer to focus purely on design. To make a character the designer just mentions the vertices. Display, interactions, collision detection etc is taken care of.

The collision detection system works by detecting hitbox collisions. The modelling system handles the creating and modification of hitboxes. When a model logs into the collision detection system it is assigned an ID. The **onHit** method of the model is called when it is involved in a collision.

The path-finding system divides the world into a 10x10 grid of cells and uses the A-star algorithm to find the optimal path from a cell to another cell. The collision detection system helps path-finder to detect obstacles and verify validity of a target.

Requirements

Hardware Requirements

The project was developed and tested on a PC powered by Intel Core i5 with a AMD R5 M430.
Here are some hardware requirements.

- Color video output
- Keyboard
- Mouse
- 5.4MB on Disk
- 2 GB RAM
- Internet connection to receive updates.

Software Requirements

The project was developed and tested on Ubuntu 16.04/18.04. Opengl libraries provided by mesa and free-glut. Cmake was used to build.
Here are some software requirements.

- Recent OS which supports OpenGL
- OpenGL and GLUT
- C++11 compiler
- Linker and loader

Implementation

drawLineBreshenham: line draw function

This function takes 4 parameter ie 2 x,y coordinates and draws a 1 pixel wide line across them. It uses Bresenham's line drawing algorithm to generate a array of coordinates representing the line. The array is then sent to OpenGL for display. Vertical lines are draw by simply loading the range of x values into array.

fillPoly: polygon draw function

This function takes a vertex array and draws the polygon described by it using a customized scanline polygon fill algorithm. The custom implementation removed need for special cases for vertex intersections and the additional data structures needed for it. The runtime is debatable.

myPushMatrix and myPopMatrix: implementing glPushMatrix, glPopMatrix

myPushMatrix stores the current global transformation matrix and loads a copy of it in its place. myPopMatrix restores the previous global transformation matrix and discards the current. This is convenient function that helps in storing/restoring a previous matrix after transforming it. This is convenient function that helps in storing/restoring a previous matrix after transforming it. For example instead of restoring global matrix with reverse translation and reverse rotation after a translation and rotation, the myPushMatrix can be used to save the

current matrix, modify it, then restore it using myPopMatrix so that other draw operations are not affected by these transformations.

myTranslatef: implementing glTranslatef

Creates the translation matrix using given (x,y) and multiplies itself to the global transformation matrix. Draw operations after this are affected by this transformation.

myRotated: implementing glRotated

Creates the rotation matrix using given angle and multiplies itself to the global transformation matrix. Draw operations after this are affected by this transformation.

CoolMan: the collision manager

A grid representing the world is initialized to -1. When a model is **push**ed into the grid, it's **hCode** is added to every cell within it's hitbox. Once all models are pushed, we check the **dynamicModels**'s hitboxes for overlap by comparing the hitbox region with **hCode** and call **onHit** of overlapping models. If any collision is detected in previous step, we check **staticModels**'s hitboxes for overlap and call onHit appropriately. To update position of a dyamic Model, we first pop it(subtract hCode), update position and push it. The collision manager also maintains a 10x10 grid which, through the **unsafe** method, tells if a particular cell is occupied by a model.

AutoPilot: the path-finder

Using A* path finding algorithm and **unsafe** method, the method determines a short yet safe path of traversal. It is used by the **Tank.autoPilot** and the **Cat.update**. The cat's movement is generated by choosing a safe point at random and asking the autoPilot to generate a convincing intermediate path.

Results

The game delivers on features mentioned in abstract. Only the point primitive was used for constructing the game. Custom drawers had good accuracy and performance. Transformation helpers were developed successfully. Transformation helpers had good accuracy and performance.

Bounding rectangle hitboxes were convincing enough. A* algorithm performs good enough to not notice any unresponsiveness on click. Game performance was not adversely affected by use of custom drawers.
The custom scanline polyfill and Bresenham’s algorithm are able to draw corresponding shapes in good time.

Conclusions

The mini project was a reasonably successful. Features mentioned were implemented to reasonable satisfaction. The project also gave plenty of insight into the working of OpenGL, necessity of efficient algorithms in computer graphics etc.

Future works

- More models
- Multiplayer option
- Interactive UI with text on screen
- Health indicators
- File based models
- Fix memory leaks
- Optimize scanline filling

References

- OpenGL documentation: https://www.opengl.org/documentation/
- ScanLine polyFill concept: https://hackernoon.com/computer-graphics-scan-line-polygon-fill-algorithm-3cb47283df6?gi=6b9408b53f30
- Computer graphics with OpenGL - Book by Donald Hearn and M. Pauline Baker
- Theory: https://www.wikipedia.org/
- STL reference: http://www.cplusplus.com/reference/stl/
- A* algorithm: https://en.wikipedia.org/wiki/A*_search_algorithm