A 3D Pacman Game Approach

By Team 8: Brief Specification, Design, and User Manual

COMP 371
Fall 2011

Table of Contents

Introduction

3D Pacman was created to full-fill the requirements of COMP 371 Computer Graphics, Concordia University, Fall 2011. The Objective of this project is to commemorate the 31 anniversary of the iconic arcade game Pac-Man, by Nameco, by implementing a 3D version in OpenGL with
C++. For the duration of the semester the project was developed iteratively, beginning with modelling, then lighting, camera controls, texture mapping and finally game play, levels and shadows. The project was developed in accordance with the specifications in COMP371: Computer Graphics TENATTIVE PROJECT DESCRIPTION (Revision 1.3), Serguei A. Mokhov

1. Architecture

1.1 Introduction

This project employs an Object Oriented architecture to encapsulate all the elements of the game play. The software architecture is based on an example provided by Dr. Grogono, Graphic Examples, Object Oriented Style: users.encs.concordia.ca/~grogono/Graphics/OGL/ooopengl.cpp. An inheritance model uses the Sprite class as base class for all Objects in the scene. This includes Wall, Tile, Pellet, Eight, Ghost and Pacman. All the objects in the scene are contained within a Map object (not to be confused with the native C++ data structure Map). The Map contains all the Sprites. The Map draws all the Sprites and moves them around the scene. The Map is also responsible for collision detection, scoring and level management. The EventListener object initializes the scene, sets all the OpenGL variables, creates the Map, draws the Map. It listens for key presses and mouse clicks, and tells the map what to do in response. This project will run in both Windows and Unix environments.

1.2 Files and Data Formats

The source of this project is contained in standard C++ header and class files. The texture maps are in BMP file format and are found in the data directory. The maze design is contained in a plain text files which can also be found in the data directory.

1.3 Code

1.3.1. Introduction

The basic idea behind the software architecture is to represent each object, or sprite, within the scene as it's own object within the code. The abstract base class Sprite contains the sprite's position in the x, y and z co-ordinates, the texture, a camera and the ability to move the sprite around the scene. Each sprite inherits from Sprite and extents the sprites capabilities. All the sprites are contained within a Map object. Event handling occurs in the EventListener class. The response to mouse behaviour is determined by the Mouse object.

1.3.2. Event Listener

The EventListener is responsible for handling all the input. It contains numerous boolean switches that control the various display options including camera selection, camera pitch, roll and yaw, shading, light and texture switches.

1.3.3. Map

The Map object acts as a container for all the Sprites in the game. It creates the maze, initializes all the Sprites and tracks the levels and scoring of the game. The function Map::draw() draws all the Sprites by calling each sprites draw function.

Collison mapping is handled by the Map object. Map::hasPellet() is manages collision detection between Pacman and Pellets. It looks up the x and z coordinates of Pacman in the Pellets array and check if there is a pellet. Map::pacmanGhostCollisionDetection() uses brute force to compare the location of each Ghost to that of Pacman. Map::whatDirectionsCanHeMove() checks the configuration of the Walls at a given coordinates. It returns a bit string that indicates which directions a Sprite can move from it's location. The sprite is only allowed to move in the allowed direction. This prevents collisions with Walls.

The motion of Ghosts and Pacman in autoplay mode is determined by a random selection of the 4 possible moves that one of these Sprites can make, namely move forward or back, turn right or turn left.

1.3.4. Maze

The maze is defined my a plain text file. The file is a series of chars. Each char represents the wall configuration around a single tile. Each char is mapped to a bit string of length 4. Each bit represents 1 side of the tile. If the bit is 1, there is a wall. If it is 0, no wall. The positive direction along x axis is consider north (N), the negative x direction, south(S), positive z is east (E) and negative z is west (W). The bit string maps like this: ENWS. The bit string 1000 would indicate 1 wall on the east edge of a tile. 1111 would indicate walls on all sides of a tile, and 0000 no walls.

1.3.5. Pacman

Pacman is formed from 2 half spheres. The half spheres are modelled using quadratic spheres with clipping plane. He also has 2 spheres as eyes. Pacman has 3 different textures that can be used. The texture is switched by pressing the 'x' key. The Jack-o-latern texture is automatically used when pacman is in power pellet mode.

Pacman can move forward and backward. He can also turn right and left. A tile is 2 unit x 2 units. Each step pacman takes is 1 unit. Thus pacman steps from the middle of a tile to the edge of a tile, and then the the middle of the next tile when moving forward or backward. Pacman can only move forward or backward when it is facing directly N,E,W or S and when he is in the middle of a tile. Pacman’s facing direction is traced in variable ‘rotation’. Pacman’s rotation is limited to the range 0-360.

Pacman has various states. He can be alive or dead. He can also be in power pellet mode. All the states are represented within the Pacman object.

The Pacman object also contains a camera. When this camera is the active camera the player has first person perspective from pacmen.

1.3.6. Ghost

There are 6 ghosts in this game. These Ghosts are stored in an array in the Map object. Each ghost is formed from a cone. The cone is modelled from a quadratic cylinder. Ghosts are also transparent. This is accomplished using alpha blending. The movement of a Ghosts is analogous to that of Pacman. Ghost can also be alive or dead. When a Ghost is dead it is not longer displayed and is not an active can element. The Ghost object also contains a camera. When this camera is the active camera the player has first person perspective from the ghost.

1.3.7. Pellet

There is a pellet on each tile. These pellets are stored in a multidimensional array, which is indexed according to the pellet's position in the maze. The first index indicates it's placement on the x axis, the second the position on the z axis. The pellet is modelled using a quadratic sphere. A pellet is either a standard pellet or a power pellet. Standard pellets are smaller and have a pink and blue texture. Power pellets are larger and have yellow texture. The Pellet object tracks whether the pellet has been eaten or not.

1.3.8. Streetlight

There is a streetlight in each corner of the maze. It is modelled using a rectangle created from GL_QUADS for the post and quadratic sphere for the bulb. Each streetlight has OpenGL light that is position at the top and shines on the maze at a 45 degree angle downward. Each light can be turn on and off independently. There is also master switch that turns all the lights on and off at o nce. Each streetlight contains a camera. When this camera is the active camera the player views the maze from the top of the lightpost at a 45 degree angle downwards.

1.3.9. Wall

Walls are modelled using GL_QUADS. Walls have a type that specifies the walls orientation in relation the tile it surrounds. The type specifies if the Wall is on the N,E,W or S edge of the tile. Like Pellets, Walls are stored in a multidimensional array in the Map object, which is indexed according to the Wall's position in the maze.

1.3.10. Tile

Tiles are modelled from GL_QUADS with 4 vertices. Each Tile also has an Eight object, which is the 8 that appears on it's face. This Eight is modelled from GL_QUADS and GL_TRIANGLES. Like Pellets, Tiles are stored in a multidimensional array in the Map object, which is indexed according to the Tile's position in the maze.

2. Operation

2.1 Game Play

The object of the game is for pacman to eat all the pellets in the maze.

The ghosts are the enemies and will damage pacman's health if he collides with them. Pacman's health is decreased by 20% with each collision.

There are 2 type of pellets: power pellets and standard pellets. Power pellets are the large yellow pellets. When pacman eats a power pellet he enters power mode. Power mode lasts for a small period of time. During power mode pacman score point for each standard pellet eaten. He can also kill ghosts during power mode. When power mode comes to an end a message is posted to the console.

There are two levels in this game. Once both levels are complete the game is finished. A player can choose to skip to level 2 if the wish, although this will have a significant impact on the final game score.

2.2 Game Controls

KEY	ACTION
GENERAL CONTROLS	
h	Help.
a	Autoplay mode.
5	Fullscreen on/off switch
6	Skip to level 2.
PACMAN CONTROLS	
↑	Move forward.
↓	Move backward.
→	Turn right.
←	Turn left.
SHADING CONTROLS	
w	Wireframe mode.
f	Fill mode.
s	Smooth Shading.
b	Flat Shading.
r	Reset view position.
CAMERA CONTROLS	
|	Camera pitch.
;	Camera roll.
“	Camera yaw.
1	Switch to Lightpost 1 camera.
2	Switch to Lightpost 2 camera.
3	Switch to Lightpost 3 camera.
4	Switch to Lightpost 4 camera.
9	Switch to Ghost camera.
0	Switch to Pacman camera.
,	Move camera left.
.	Move camera right.
m	Switch to turn on and off camera control by mouse.
[	Zoom in.
]	Zoom out.

LIGHT CONTROLS	
d	Daylight.
n	Nightlight.
y	All Streetlights on/off switch
u	Lightpost 1 on/off switch.
i	Lightpost 2 on/off switch.
o	Lightpost 3 on/off switch.
p	Lightpost 4 on/off switch.
TEXTURE CONTROLS	
z	Turn on/off all texture mapping.
x	Switch pacman's texture.
c	Switch between pellet texture modes.

3. Miscellanea / Appendices

3.1 Open Issues

- Key mapping usability – The current key mapping is neither intuitive or ergonomic. A more involved and refined key mapping and handling of input is necessary. It would be a good idea to implement input modes. For instance, if the user presses '1' it would activate Texture Mode, where within all the alpha keys are mapped to the various texturing options. Then if the user presses '2' the key would map to camera controls and so on.
- Mapping of the plain text source file to scene – The current mapping scheme is highly unintuitive. It is extremely cumbersome to work with to generate new mazes. A more intuitive mapping would utilize the standards of geographic mapping. First, the 4 bit string should correlate to the global coordinate system, starting with north and moving in a clockwise direction, ie: NESW. The plain text source should likewise map to the global coordinates whereby the first line defines the first line of tiles, the most northern, starting at the origin of the game space and moving along the x axis, towards the east.
- Camera controls - The camera movements need to be refined to give smoother camera control.

3.2 Code Sources

- Implementation of the GLFrame Class by Richard S. Wright Jr.
  These libraries are used for camera control.
  Includes: glFrame.h, math3d.h, math3d.cpp
  Source:
http://code.google.com/p/oglsuperbible5/source/browse/trunk/Src/GLTools/include/GLFrame.h
- Simple OpenGL Image Library
  This library is used for loading textures.
  Source: http://lonesock.net/soil.html

3.3 References

1. Peter Grogono. Graphics Examples, Object Oriented Style. [online], 2011. http://users.encs.concordia.ca/~grogono/Graphics/examples.html
2. Richard S. Wright Jr. Implementation of the GLFrame. [online], 2011. http://code.google.com/p/oglsuperbible5/source/browse/trunk/Src/GLTools/include/GLFrame.h
3. Simple OpenGL Image Library. [online], 2011. http://lonesock.net/soil.html