PROJECT REPORT
“DUCKS IN SPACE”

Bachir Alhabbal	9334394
Pham Hoag Anhkhoa	5682304
Marc-André Faucher	9614729
Marc Leduc	9132007

A Project Report
submitted in partial fulfillment
of the requirements of
COMP 371

Concordia University

August 2011

TABLE OF CONTENTS

1. INTRODUCTION

This report accompanies our final project submission for the class COMP 371, Computer Graphics. The project consisted of programming a game in C++ using the OpenGL graphical library. Our team is composed of the following students:

- Bachir Alhabbal (9334394);
- Pham Hoag Anhkhoa (5682304);
- Marc-André Faucher (9614729);
- and Marc Leduc (9132007).

We decided to develop an arcade style game called “Ducks in Space”. In this report we will cover the game objective and interface, along with the OpenGL features we implemented in our project.

2. DESCRIPTION

The player's objective in “Ducks in Space” is to avoid obstacles coming towards him or her. The player controls a duck navigating between obstacles of multiple forms – such as spinning teapots, illuminating suns, flipping beetles and swaying cubes. The different obstacles move in his or her direction and are many times bigger than the duck to make them difficult to avoid.

The player starts with 3 extra lives, 3 health points and a score of 0. This information is displayed at the bottom left screen. The score increments as the player survives. If the player is hit by an obstacle, he loses one health. When the player is hit three times, he loses one life, and if he has no lives left to lose, it's game over.

The game world is set in space, however the player's movement delimited by a transparent walls on each sides. The game features three starter levels where the pace starts off slow, with low speed, low fog, and introducing the basic obstacles. Over the next two levels, the game adds one advanced object each level, increases the speed, increases the frequency of objects, and increases the fog value to lower visibility. After the third round no new obstacles are added, but it continues to generate as many increasingly difficult levels as needed until you run out of lives and reach what may end up being your highscore. In order to beat each level, you are required to survive for 60 seconds without losing a life. The level resets upon loss of a life, requiring you to start the 60 seconds over again.

What level can you reach?

3. USER-INTERFACE

The game begins with a title screen. The player navigates the menus intuitively using the left mouse button. Clicking the instructions button will show a summary of the game's controls. Clicking the credits button will show the names of the game designers. From any other menu screen (instruction, credits and game over) clicking anywhere will return you to the title screen. Finally, the “Play Game” button starts the game.

The player controls a flying rubber duck avoiding obstacles. To dodge the obstacles, the player can move to the left, right, up and down with the keyboard keys 'a', 'd', 'w', and 's', or even combinations of those allowing for diagonal mvement. In addition, the player can look around his surrounding with the help of the mouse, or zoom in and out with the +/- buttons even allowing first-person view. To quit the game, you simply press <ESC>.

We have included a testing mode to control parameters in the game. Simply press '~' to start testing mode. In testing mode, you can change levels manually by hitting <F1>, <F2> and <F3> for level 1, level 2 and level 3 respectively. Pressing <F4> will take you to the game over screen, where you can go back to the main menu. Several other options are available in testing mode:

- Pressing '1' will show the spheres we used to approximate the collision detection;
- Pressing '2' will hide the spheres again;
- Holding down '3' will change the dimensions of the playing field by moving the glass panels which block the player's movement.

4. FEATURES

We designed the game using mostly existing models because of lack of time and modeling skills. Most obstacles and duck all use existing models [1] [2] [3] [4] [5] [6]. The title screen, game over screen, mask texture and the accurate depiction of mars were found online [7] [8] [9][10]. However, apart from those, the rest of the textures, material files and buttons we created ourselves. We also use particles to draw the stars at random locations. Finally, we use bitmap fonts to display information on the game screen.

The game has up to five different lights. Initially, there is a weak ambient light and an orange, directed light coming from behind the player. The other three lights are point lights coming from the three possible sun obstacles. The material of each object is tweaked to appear more realistic, reflecting more or less diffuse, specular or ambient light. We also used black fog, which render the obstacles invisible until they come out of the fog. Both the fog and the lighting are enabled after the planets and stars are drawn.

We use the depth buffer so that the obstacles are drawn in the correct order. Using alpha blending, the glass panels are drawn over the game background with a low alpha value so that we still see the planet and stars.

To improve the rasterization, we use antialiasing by enabling GL_LINE_SMOOTH and mipmapping for the obstacle textures. The game will track the window size and adapt the display and input when the window is reshaped in every state, rescaling everything from button location to control of the view with the mouse.

We created a camera class which handles most of the view aspects. This includes rotating the camera around the player to look around, or moving it backwards and forwards to zoom in and out. The zoom feature also allows you the choice between third and first person views.

We implemented our own spherical collision detection based on manually set spheres emanating from each object, and matching the objects with their moves and scaling. Collision is efficiently calculated at all times between the duck and each object, as well as collision between the objects and the walls, pushing the object away from the wall if they collide. Our spheres are accurate for all objects except the teapot spout, as we found it would be unfair to have the collision sphere be that skewed from the long spout, but we love teapots too much to replace with an easier model.

Every object has its own form of animation, showcasing different aspects available to us in OpenGL. For example, the yellow suns pulse and emit a lightning effect, the skull translates along the X axis, the cone scales itself bigger and smaller, the teapot rotates along a random axis with a random color, the Vmask rocks back and forth along the Z axis, the cube combines everything together. Then we add extra objects in order to make things challenging, gloves that move along the Z axis (forward and back) and giant spinning swords that move in circles around the field. The planet in the background uses gluSphere and quadrics and a texture of mars.

The duck itself features several reactive effects such as spinning once on hit and blinking on death.

5. HIGHLIGHTS

We wish to highlight certain features which took considerable time to implement. Since the demonstration, we have also added several additional features which we want to touch upon. We coded the game to be flexible in case we wanted to tweak the game. This allowed us, for example, to dynamically add new levels if the player passes level 3, and the levels will continue to get harder and harder.

We added sound and music to the game, using free sounds effects found online [10]. We also added buttons to the main menu, as well as a credits and instruction screen.

We used a linked-list to implement a queue for the obstacles. This way, the game can generate endless streams of obstacles, and keep track simultaneously of hundreds of obstacles. Each obstacle is randomly generated with an x and y position, as well as one of 9 object types. There can only be three 'suns', each with their own light source. Each type of object is differently animated, many of those were added since the demonstration. We have limited the obstacle creation code so that certain objects only appear at the later levels.

We extended the glass perimeter so that it doesn't clip obstacles. The player's movement extends with the glass perimeter (this can be tested in test mode by pressing '3'). We added collision detection between the obstacles and the glass borders, and any obstacles which would exit the borders will be pushed back inside.

6. CONCLUSION

In implementing this game, we were able to put into practice many of the OpenGL features we learned about in class. In developing this project, we were able to execute our plan to build an arcade game, and integrate the knowledge we gained throughout the semesters. OpenGL, and similar libraries, are quite powerful and allow us to access hardware in order to draw 3D objects on the screen. It is a powerful skill for any programmer to have.

REFERENCES

[1] Game title image: http://www.budproduct.com/collections/80s-pop-culture/ducks-in-space-mini-duck-set-2/ducks-in-space-mini-duck-set/

[2] Game over image: http://oilersnation.com/2010/4/11/gdb-lxxxii-game-over

[3] Sun texture: http://www.turbosquid.com/FullPreview/Index.cfm/ID/557319

[4] Mask model and texture: http://www.3dvia.com/content/63BF85596B7D4F61

[5] Bug model and texture: http://www.oyonale.com/modeles.php?lang=en&page=55

[6] Glove model and texture: http://www.turbosquid.com/FullPreview/Index.cfm/ID/223158

[7] Sword model and texture: http://www.turbosquid.com/FullPreview/Index.cfm/ID/362147

[8] Skull model and texture: http://www.turbosquid.com/FullPreview/Index.cfm/ID/269706

[9] Rubber ducky model: http://www.oyonale.com/modeles.php?lang=en&page=53

[10] Music and sound effects: "http://www.freesound.org/”