Computer Games: ARGame

James Rogers, 100062949

School of Computing Sciences, University of East Anglia, UK

Abstract

Abstract.

1 Introduction

2 Requirements

2.1 Specification

2.2 MoSCoW

3 Design

3.1 IID

3.1.1 Version 0.1

3.2 Technologies

3.3 Inital Architecture

3.4 Final Architecture

3.5 Finite State Machine

4 Implementation

4.1 Graphics

This section will detail all graphical related modules. This is the method of drawing graphical information to the screen, the loading, storing and managing of geometric data and materials.

4.1.1 GLKit

Apple[1] provide the GLKit[2] framework which manages the OpenGL ES context and state information. It also provides additional functionality such as the `Effect` class which contains ans manages shader programs for easy drawing. It also contains math libraries such as vector and matrix operations.

4.1.2 Mesh

The `Mesh` class contains an array of vertex and material data to represent a virtual geometric construction. A vertex is represented via a `Vertex` structure which contains two 3D vectors, a position and a normal, and a 2D vector for the texture coordinate. The material consists of diffuse and specular textures which are used approximately model material characteristics. Instead of using every three contiguous elements in the vertex array to represent a triangle, `Model` contains an array of vertex indices to point to vertex elements. Every three contiguous elements in the indices array forms a triangle. This eliminates the need of storing multiple vertices, ultimately using less GPU memory, which is especially important on a mobile device where memory is limited. The OpenGL ES method, glDrawElements is used to draw the mesh data using the indices array.

4.1.3 Model

The `Model` class contains an array of meshes to represent a more complex object. For example, the model of a chair could contain two meshes; one mesh to represent the wooden frame, and another mesh for the cushion. This also allows models to be 'pulled apart' or attached to each other by simply combining the mesh array.

4.1.4 Model Loading

Model loading is performed using the `ModeLoader` class. It provides an interface of static methods which take a file path to the model file as their parameter, then return an instance of the `Model` class with the data loaded inside it.

The Assimp[3] model loading library was used to read 3D mesh and material data from Wavefront .obj formats. The Assimp library is written in C++, therefore a class, `AssimpModelLoader` was implemented in C++ to interact with the Assimp library, interpret and temporarily store it's output. The Swift `ModelLoader` class interacts with `AssimpModelLoader` through a bridging header; which consists of C functions that Swift can invoke to retrieve data from `AssimpModelLoader`.

Specifically, `ModelLoader` creates an instance of `AssimpModelLoader` using the initAssimpModelLoader C method which takes the file path of the model and returns a void pointer to the instance of `AssimpModelLoader` with the model data loaded inside it. Next, the void pointer is passed through other C methods in the bridging header to retrieve the model data, which is then stored in the `Model` class. Before returning the newly created model, the void pointer to the assimpModelLoader is destroyed, along with the model data it holds to avoid memory leaks.

4.1.5 Object

The `Object` class holds a `Model` and transformation data such as translation, rotation and scale matrices. This class is responsible for the positioning of the `Model` and controlling its movement in the environment.

[1] `https://developer.apple.com`

[2] `https://developer.apple.com/reference/glkit`

[3] `www.assimp.org`

4.1.6 Photogrametery

4.2 Collisions

4.2.1 Collision Detection

4.2.2 Collision Resolution

4.3 Augmented Reality

This section will detail the augmented reality aspects of the application. `ARHandler` is a wrapper class which contains ARToolKit functionality for the camera and marker tracking.

4.3.1 Camera

The `ARHandler` class controls the camera attached to the device. It uses the `CameraVideo` class which is a part of the ARToolKit[4] SDK. `CameraVideo` provides functionality of turning on the video from the camera. Each time a frame is captured from the camera video, a delegate method is invoked to handle the raw camera output. The raw output consists of two bit planes in the YUV colour space. These bit planes are stored in an OpenGL texture cache which avoids coping the buffers, resulting in increased performance. When the texture is drawn to the display, the YUV colour space is converted to RGB.

4.3.2 Marker Tracking

Marker tracking is also handled in the `ARHandler`, as it needs to work closely with the raw output of the camera.

4.3.3 Smoothing

4.3.4 Illumination Model

5 Testing

5.1 Unit Testing

5.2 Intergration Testing

6 Conclusion

[4] `https://artoolkit.org`