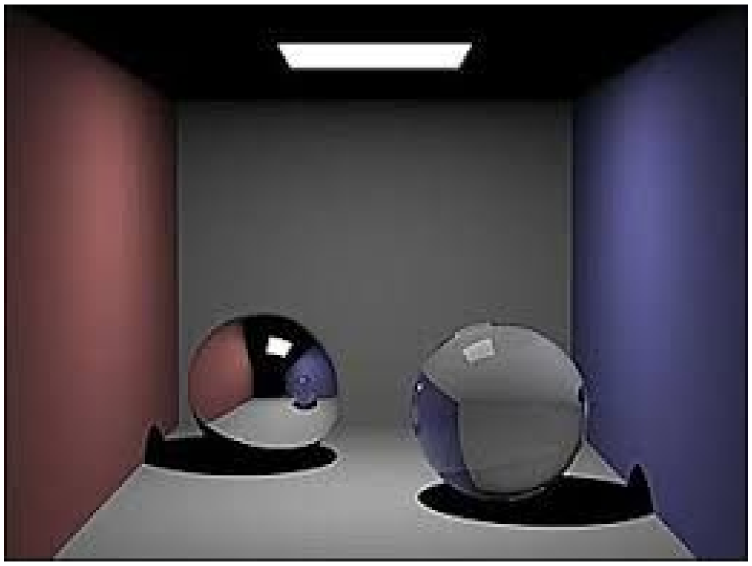
COMPUTER GRAPHICS REPORT

Creating a Moving 3D Model with OpenGL both from Mac OS and Windows OS



Members:

425514 ALVIN JANUAR RAMADAN (IUP)

425516 FAIZ KHANSA ADRIKA (IUP)

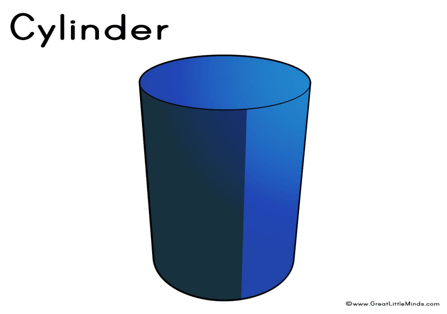
423119 TEGAR TAUFIK RAHMAN (IUP)

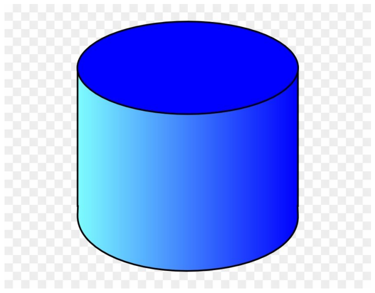
423121 ZHAFIRA ELHAM FAWNIA (IUP)

INTRODUCTION

In this report we would like to emphasis on our 3D model project assigned by Mr. M. Edi Wibowo from Computer Graphics class. It is in our concern that our team members have different operating systems, half of us have a Windows OS, while the other have a Mac OS. So we have decided to create ***two projects*** using OpenGL Windows, and using OpenGL ES from Mac OS.

Our inspiration for the OpenGL Mac project:





[It is literally just a cylinder, but there will be a **rotation**, **texture** and also effects on it]

MATERIALS & PROCESS

1. **For the 3D Model created in OpenGL ES Macbook**

   We use XCode

   We are using Swift to create an OpenGL application for macOS, iOS, and tvOS (But who even owns a tvOS). It features a rotating, normal mapped cylinder that is illuminated by three moving point lights of different colors Aside from the vertex and fragment shaders written in GLSL, all of the application code is written in Swift :)

BUILD REQUIREMENTS:

- Mac version: Mac OS X 10.6 or later, Xcode 6.1 or later
- iOS version: iOS SDK 4.0 or later

RUNTIME REQUIREMENTS:

- Mac version: Mac OS X 10.6 or later
- Mac version: Mac OS X 10.7 or later to use OpenGL 3.2
- iOS version: iOS 4.0 or later (with OpenGL ES 2.0 support)

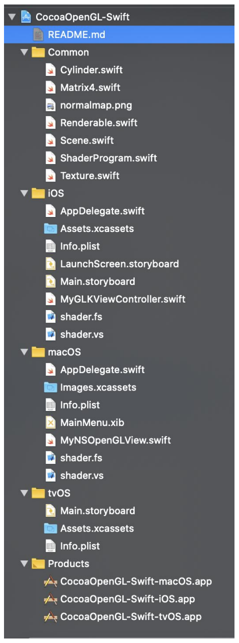
REVISION HISTORY:

First week:

Deciding whether we make both in the Windows platform or OS platform, due to our shortage and difference of OS between our members, we have decided to create for both platforms.

Also decided on what to create, searching for examples and references on the internet (for mac users, since it is required to have an XCODE and be able to operate OpenGL ES)

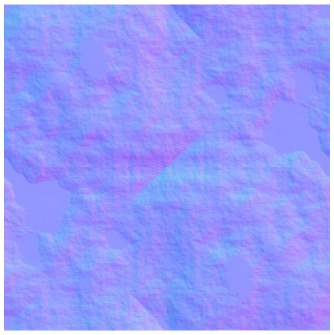
We figured out the distribution of the files:
The distribution itself is defined by four categories:



```
▼ CocoaOpenGL-Swift
    README.md
  ▼ Common
      Cylinder.swift
      Matrix4.swift
      normalmap.png
      Renderable.swift
      Scene.swift
      ShaderProgram.swift
      Texture.swift
  ▼ iOS
      AppDelegate.swift
      Assets.xcassets
      Info.plist
      LaunchScreen.storyboard
      Main.storyboard
      MyGLKViewController.swift
      shader.fs
      shader.vs
  ▼ macOS
      AppDelegate.swift
      Images.xcassets
      Info.plist
      MainMenu.xib
      MyNSOpenGLView.swift
      shader.fs
      shader.vs
  ▼ tvOS
      Main.storyboard
      Assets.xcassets
      Info.plist
  ▼ Products
      CocoaOpenGL-Swift-macOS.app
      CocoaOpenGL-Swift-iOS.app
      CocoaOpenGL-Swift-tvOS.app
```

1. **Common**: This one is for the commonly used files which would be accessed from both the iOS emulator, macOS emulator, and the tvOS emulator. The main files are all within this folder.
2. **iOS**: As usual with iOS app development, the difference here is that the main file would be it's AppDelegate, which would control it's files and where goes where. It is also added with shader.fs and shader.vs to enhance it on the iOS emulator
3. **macOS**: It is almost the same with the iOS one, the difference is that it doesn't run on an iOS emulator, instead it goes straight as an app if you open it from your mac.
4. **tvOS**: There is very little files here due to the fact that the creator has no idea how to develop a tvOS app that runs on OpenGL animation, perhaps some other time.
5. **Products**: These are the apps that are emulated from the codes above, this files shows the targets for the output.

24/10/2019



- Figured out the texture.png for the cylinder
- Tried multiple colors in the form of a 3x3 matrix, but maybe it is possible to operate with a 4x4 matrix, but who knows
- Tried to operate the camera's view and creating the matrices

25/10/2019

- Regenerated project using Xcode 6 and above, due to the fact that the creator's macbook was just updated to catalina so the Xcode have to settle as well.
- Also sets and fixes the colors of the lights by the 3x3 matrix
- Now uses single .xcodeproj for both iOS and OS X targets
- Now uses storyboard, for both targets.

01/11/2019

- Now uses ARC.
- Fixed global vars intended to be ivars. So it's accessible from the main files and app delegates.
- Now we customize the View Controller

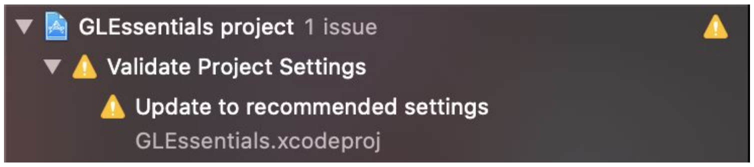
12/11/2019

- Removed code for compatibility with pre iOS 3.1 (i.e we can assume CADisplayLink is available, I think)
- Changed from GLEssentials → CocoaProjectSwift [Since we are now using import Cocoa to establish the OpenGL]

19/11/2019

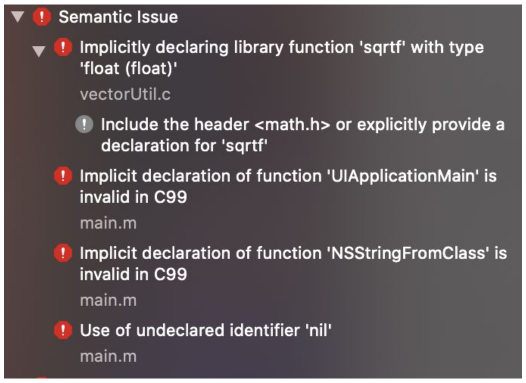
- Some bug fixes and trial and error-s

ISSUES:

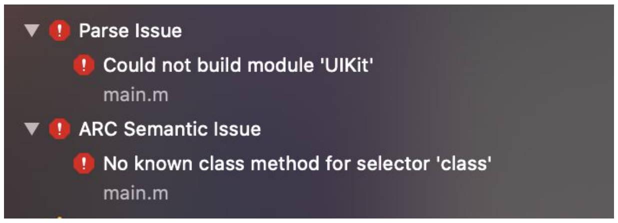


GLEssentials project 1 issue
Validate Project Settings
Update to recommended settings
GLEssentials.xcodeproj

The user's macbook has not updated to recommended settings for the XCode project.

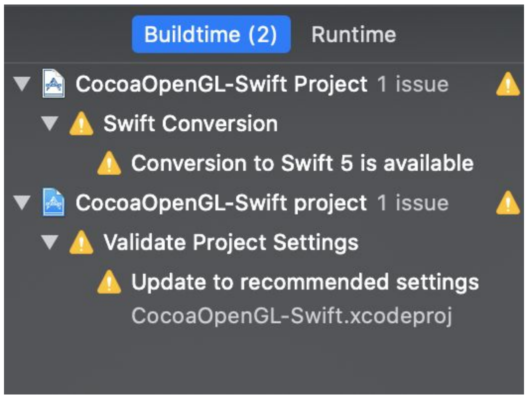


Semantic Issue
Implicitly declaring library function 'sqrtf' with type 'float (float)'
vectorUtil.c
Include the header <math.h> or explicitly provide a declaration for 'sqrtf'
Implicit declaration of function 'UIApplicationMain' is invalid in C99
main.m
Implicit declaration of function 'NSStringFromClass' is invalid in C99
main.m
Use of undeclared identifier 'nil'
main.m



Parse Issue
Could not build module 'UIKit'
main.m
ARC Semantic Issue
No known class method for selector 'class'
main.m

Surprisingly after months of trying to tackle it, this is the only warnings we have right now:



Buildtime (2) Runtime
CocoaOpenGL-Swift Project 1 issue
Swift Conversion
Conversion to Swift 5 is available
CocoaOpenGL-Swift project 1 issue
Validate Project Settings
Update to recommended settings
CocoaOpenGL-Swift.xcodeproj

It works anyway, so yeah.

PROCEDURE

1. Decide on the design
2. Start with the basis of the project, creating the project in XCode
3. Separate the project into three inner folders:
    a. Common
        i. Cylinder.swift: This is where the vertex data is loaded into the project, thus we have:
            1. Vertex positions
            2. Vertex Texture Coordinates
            3. Vertex Tangents
            4. Vertex Bitangents
            5. Vertex Normals

            Afterwards we would also have the buffers and the vertex array, the buffers consists of:

            1. The main buffers is called bufferIds
            2. Position buffer
            3. Attribute array for it’s position
            4. Texture coordinates buffer
            5. Texture coordinates attribute array
            6. Tangent Attribute Array
            7. Create bitangent buffer
            8. Create bitangent attribute array
            9. Create normal buffer
            10. Create normal attribute array
        ii. Matrix4.swift: This file is to establish matrixes, the matrixes would be to define the location of the item and its' aspects.
        iii. The .png file for texture mapping
        iv. Renderable.swift:

```
 /* enable rendering */
protocol Renderable {
   func render()
}
```

        v. Scene.swift: Color alterations, shape and size of the object, camera position and point of view

   vi. ShaderProgram.swift: Compile and create the shader’s use
   vii. Texture.swift: Generate and bind texture
b. iOS/macOS/tvOS
   i. AppDelegate.swift for macOS and iOS: It is to open the UIKit, UiResponder and UIApplicationDelegate
   ii. Shader.fs and Shader.vs:

      This shader file is to create the light effect. Supposedly, it's rotate-able throughout the object

      for Shader.vs: To adjust the camera, tangent space, and light vectors from the previous shader.fs file of the project.

   iii. MyGLKViewController.swift: GL setup for iOS
   iv. MyNSOpenGLView: Using cocoa to set up OpenGL and viewport
   v. Mainstoryboard: Ui when app first running

4. Rotation
   a. Reflection
   b. Position

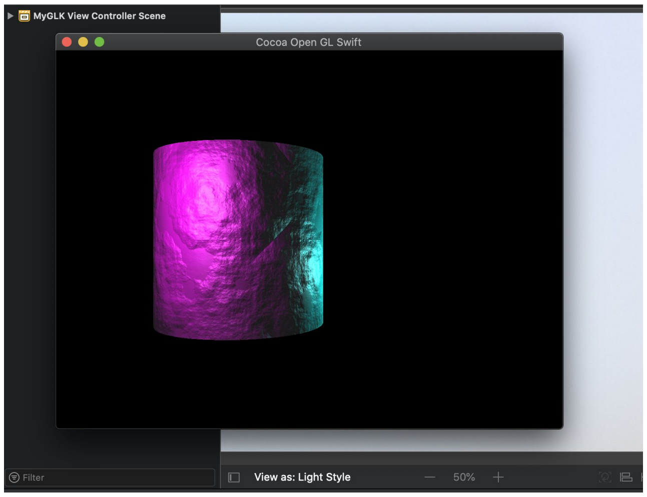
It also sets the functions for doing vector math, but it sets for more matrix math than vector math.

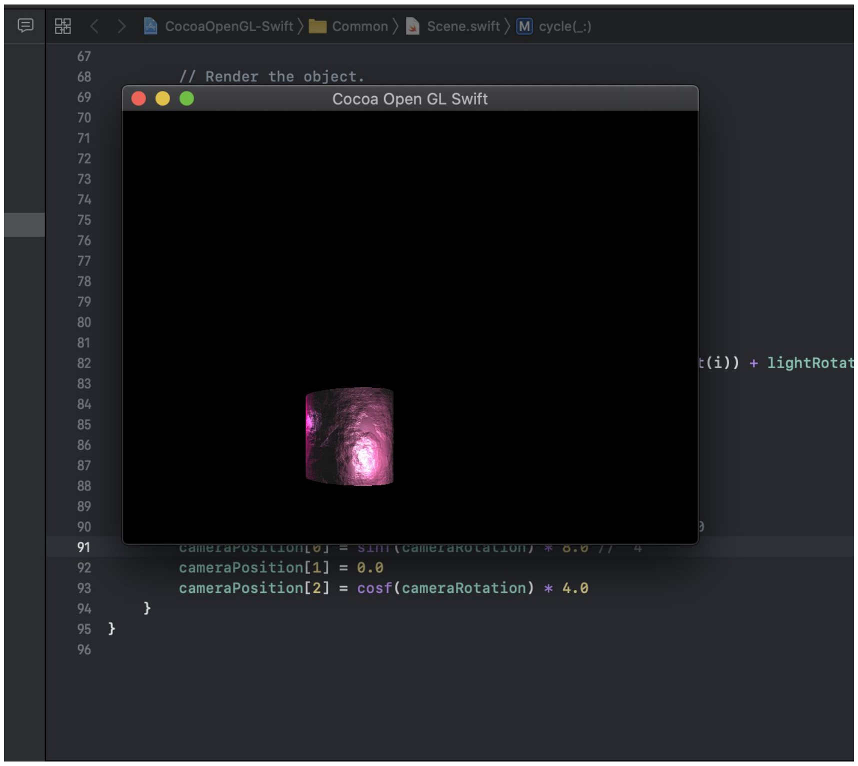
5. modelUtil.c and modelUtil.h, it sets functions for loading a model file for vertex arrays. The model file is used as a simple binary sample, invented for the purpose of the sample code given.
6. Create the application feature, it is supposedly to be downloadable through iOS platforms, but we really ***haven’t*** checked.

RESULTS

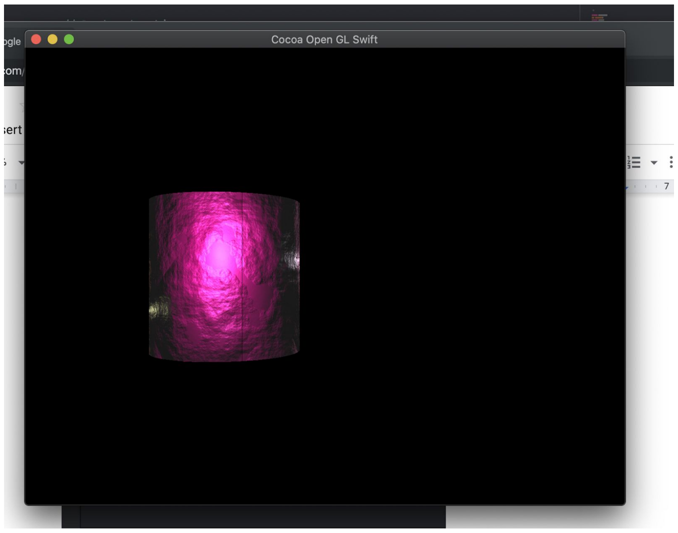
This will rotate around with lights shining upon it, three lights with three different colours.

Multiple testing:





```
        // Render the object.
t(i)) + lightRotat
        cameraPosition[0] = sinf(cameraRotation) * 8.0 //  4
        cameraPosition[1] = 0.0
        cameraPosition[2] = cosf(cameraRotation) * 4.0
    }
}
```



The video will be provided as well.

CONCLUSION

The Open Graphics Library (OpenGL) is used for visualizing 2D and 3D data. It is a multipurpose open-standard graphics library that supports applications for 2D and 3D digital content creation, mechanical and architectural design, virtual prototyping, flight simulation, video games, and more. You use OpenGL to configure a 3D graphics pipeline and submit data to it. Vertices are transformed and lit, assembled into primitives, and rasterized to create a 2D image. OpenGL is designed to translate function calls into graphics commands that can be sent to underlying graphics hardware. Because this underlying hardware is dedicated to processing graphics commands, OpenGL drawing is typically very fast. OpenGL for Embedded Systems (OpenGL ES) is a simplified version of OpenGL that eliminates redundant functionality to provide a library that is both easier to learn and easier to implement in mobile graphics hardware. Thus making it easier for us to create a 3D object model with Open GL, regardless of how it requires more utillities, and more separate files--this feature of Open GL ES from XCode is helpful.

REFERENCES

1. https://developer.apple.com/documentation/opengles & https://developer.apple.com/documentation/glkit
2. http://www.khronos.org/registry/OpenGL/index_es.php
3. https://openframeworks.cc/ → for slight references only
4. https://openframeworks.cc/learning/ → tutorials on how to get started in 3D

modelling using frameworks

5. http://www.swiftgl.org/ → learning openGL for Swift (using XCODE mac)