

# Interactive Graphics Homework 1

Michal Ostyk-Narbutt (1854051)

Prof. Marco Schaerf

May 1, 2020



**SAPIENZA**  
UNIVERSITÀ DI ROMA

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Documentation</b>	<b>2</b>
2.1	Creating an irregular object . . . . .	2
<b>3</b>	<b>Conclusions</b>	<b>3</b>
3.1	Advantages . . . . .	3
3.2	Disadvantages . . . . .	3

# 1 Introduction

This is a documentation report describing the techniques used in the First Homework for the Interactive Graphics course. Given a baseline file of a cube, the task was to modify it by

- expanding the number of vertices (20-30) with each having a normal and texture coordinates.
- adding a viewer position and a perspective projection
- computing the ModelView and Projection matrices in the Javascript application
- adding two lights sources
  1. Directional
  2. Spotlight
- assigning to the object a material with the relevant properties
- Implementing a per-fragment shading model
- adding a texture loaded from file, with the pixel color a combination of the color computed using the lighting model and the texture.

## 2 Documentation

### 2.1 Creating an irregular object

My idea was to create some octagon Hourglass as seen in Figure 1/. In order to accomplish that task I decided to create out of a series of triangles and quadrilaterals. However, unlike in Figure 1, mine would have a more visible inner part.



Figure 1: Octagon Hourglass [1]

I modeled the vertices based 4 faces of different widths, with the top and bottom sharing one value for the width, and the inner "neck" of the hour composed of two octagons also sharing one value. To accomplish this task I decided to run the following python code which allowed me obtain various initial sizes.

```

1 def getshapevalues(height_value, vertices=8, width=1):
2     coords = []
3     for ind, value in enumerate(range(vertices), start=1):
4         x = cos(2*value*pi/vertices) * width
5         y = sin(2*value*pi/vertices) * width
6         coords.append([x, y])
7     return coords

```

Listing 1: Getting values for the vertices

The code snippet in Listing 1 in Python is just a visual version of obtaining a polygon coordinates. These in reality would be checked for value and using strings formatted into JavaScript ready code. The result is show in Listing 2 below.

```

1 vec4(0.6, 0.0, -1.0, 1.0), // point 0
2 vec4(0.3*Math.sqrt(2), 0.3*Math.sqrt(2), -1.0, 1.0), // point 1
3 vec4(0.0, 0.6, -1.0, 1.0), // point 2
4 vec4(-0.3*Math.sqrt(2), 0.3*Math.sqrt(2), -1.0, 1.0), // point 3
5 vec4(-0.6, 0.0, -1.0, 1.0), // point 4
6 vec4(-0.3*Math.sqrt(2), -0.3*Math.sqrt(2), -1.0, 1.0), // point 5
7 vec4(0.0, -0.6, -1.0, 1.0), // point 6
8 vec4(0.3*Math.sqrt(2), -0.3*Math.sqrt(2), -1.0, 1.0), // point 7
9 vec4(0, 0, -1.0, 1.0), // point (centroid) 8

```

Listing 2: Bottom Octagon example vertex defintions

## 3 Conclusions

### 3.1 Advantages

### 3.2 Disadvantages

## References

[1] <http://evershinegift.com/Index.asp?Product420/742.html>