UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science
Faculty of Engineering, Built Environment & IT
University of Pretoria

COS344 - Computer Graphics

Practical 2 Specification: 2D Rendering

Release Date: 26-02-2024 at 06:00

Start By Date: 22-03-2024

Due Date: 04-04-2024 at 09:00

Total Marks: 60

Contents

1 General Instructions

- ***Read the entire assignment thoroughly before you start coding.***
- This assignment should be completed individually, no group effort is allowed.
- **To prevent plagiarism, every submission will be inspected with the help of dedicated software.**
- Be ready to upload your assignment well before the deadline, as **no extension will be granted.**
- If your code does not compile, you will be awarded a mark of 0. The rendering output of your program will be primarily considered for marks, although internal structure may also be tested (eg. the presence/absence of certain functions or classes).
- Failure of your program to successfully exit will result in a mark of 0.
- Note that plagiarism is considered a very serious offence. Plagiarism will not be tolerated, and disciplinary action will be taken against offending students. Please refer to the University of Pretoria's plagiarism page at http://www.ais.up.ac.za/plagiarism/index.htm.
- You are allowed to use any standard of C++.
- The usage of ChatGPT and other AI-Related software is strictly forbidden and will be considered as plagiarism.
- No pre-build objects and textures may be used. All objects and textures that you need to use must be created by yourself.
- You must use OpenGL version 3.3 for this practical.

2 Overview

For this practical, you will need to render a simple 2D object and apply a set of transformations to the object.

3 Your Task:

For this practical you will need to render a simple 2D depiction of a motor vehicle. An example of what you are expected to render is given in Figure 1. In the sections below, the different requirements are laid out. You are allowed to design any type of car as long as it fulfills the requirements.

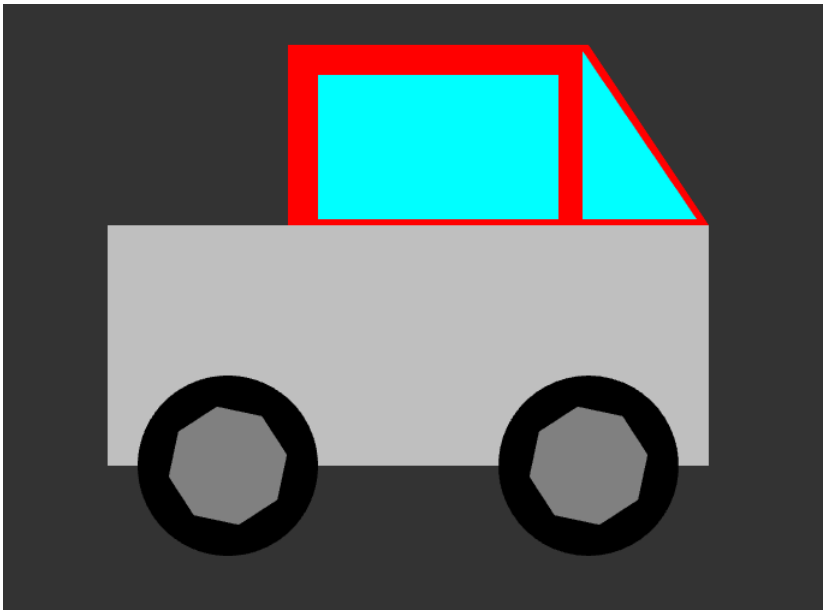

Figure 1: Example of expected render

3.1 Car requirements

The car should have the following visual components:

- A two-tone body.
- At least two separated windows of different shapes. In other words, there needs to be a clear spacial separation between the two windows.
- At least two wheels with rims.

Your car can contain more details if you wish to add it.

3.2 Shape requirements

Your car is required to at least contain two of the following distinct[1] shapes:

- Rectangle/Square
- Triangle
- High polygon circle
  - A circle that consists of at least 50 vertices
- Low polygon circle
  - A circle that consists of a vertice count of between 6 and 10.

Note you can use as many shapes as you like to build your car, as long as it complies with the above requirements.

[1]Distinct implies that the shape as a whole is counted and not the internal polygons used to create the shape.

3.3 Colour requirements

Your car needs to fulfil the following requirements in terms of colour.

- The body of the car needs to be a two-toned body.
  - Thus the body of the car needs to consist of two different solid colours.
- The wheels need to be black with the rims being a different distinctive colour.
- The windows need to also be a different colour from the rest of the car.
- The background colour has to also be a distinctive colour. *Hint: there is a opengl function that can assist with this.*

In conclusion. Your car needs to contain at least 5 different solid distinctive colours with the background also being a distinctive colour.

3.4 Transformation requirements

In this section are the transformation or animation requirements for your rendering. Note, keep the animation rates small such that it allows you to press the key multiple times before going out of camera view.

- When the `W key` is pressed, the car needs to move from its current position to the right of the screen. This mimics the car accelerating.
- When the `S key` is pressed, the car needs to move from its current position to the left of the screen, This mimics the car decelerating.
- When the `A key` is pressed, the car needs to be scaled down to a smaller size and move vertically up the screen. This mimics the car changing lanes away from the camera.
- When the `D key` is pressed, the car needs to be scaled up to a bigger size and move vertically down the screen. This mimics the car changing lanes towards the camera.
- The wheels and rims of the car needs to be constantly rotating during the execution of the program.

Below are examples of the result after multiple key presses assuming the car is at the original rendering position.

Note that when a key is pressed the car should **not** return the original position. Your program should be able to apply the transformation in any arbitrary order with the car at any position on the screen.

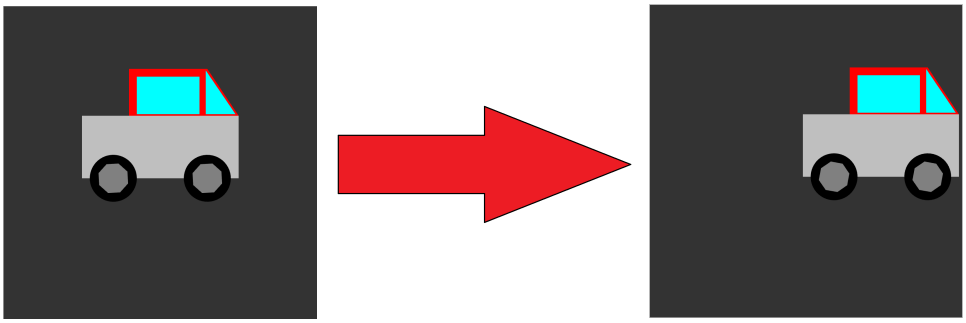

Figure 2: Original

Figure 3: After a series of `W key` presses

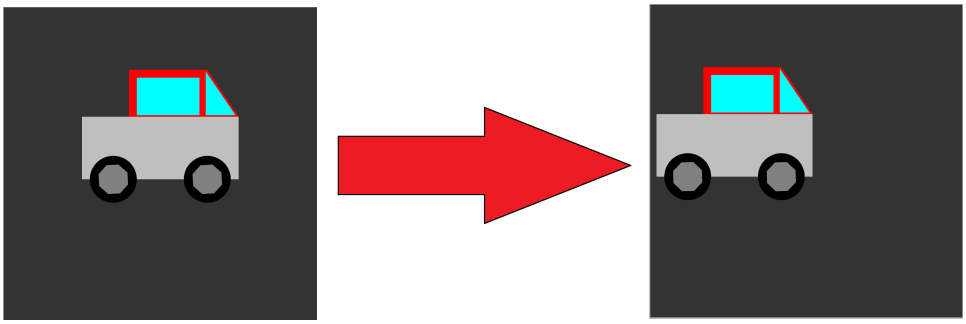

Figure 4: Original

Figure 5: After a series of `S key` presses

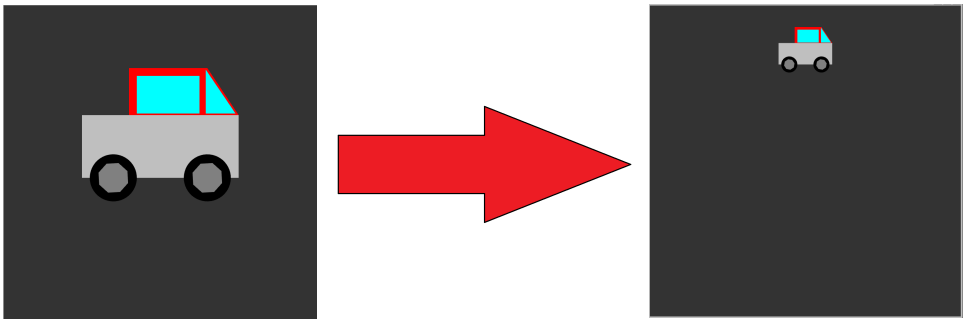

Figure 6: Original

Figure 7: After a series of `A key` presses

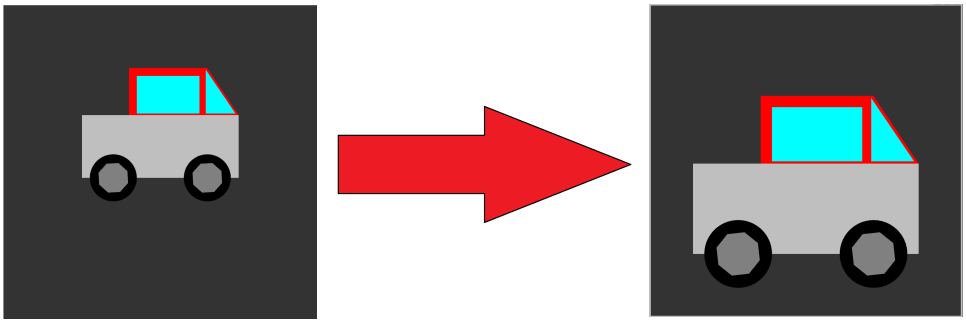

Figure 8: Original

Figure 9: After a series of `D key` presses

3.5 Wireframe

You are also required to implement a wireframe for your car. The wireframe should maintain the colour scheme of your car, such that the colours of the different shapes can be identified. The wireframe should also conform to all the transformations described earlier.

Your program should toggle between the wireframe and normal car when the `Enter key` is pressed. *Hint: you may need to implement a time delay between key presses for the wireframe toggling such that the expected behaviour is achieved.*

Please note that you **must** use the `GL_LINES` to implement your wireframe. Using the `glPolygonMode` function will result in the forfeiting of your wireframe marks.

Please see the figure below for an example of a wireframe rendering of the car.

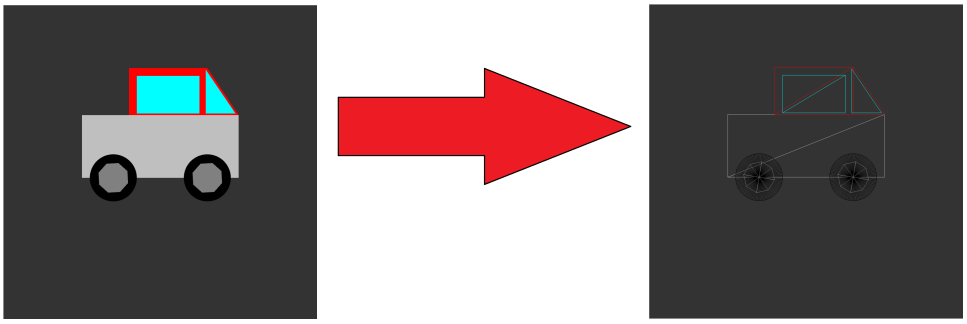

Figure 10: Original

Figure 11: Wireframe

4 Marking rubric

The following rubric will be used to mark your submitted assignment. Note you will be demoing the practical during the practical sessions on your own computer or a lab computer. Please see Table 1 for the rubric. Note: 1 mark will be subtracted for each transformation assessment criteria if the render is moved back to the center before a new transformation is applied

5 Bonus marks

There are up to 10 bonus marks available for bonus marks. Bonus marks are worth 2 marks for each extra that you render or implement.

- A proper background with trees or buildings built using primative polygons. Simply loading a texture does not count.
- Adding bound checking such that the car does not move off screen.
- Adding at least two extra details to your car such that it looks more realistic. These include shapes that represent lights, shapes that represent mirrors, etc.
- Enabling your car to flip around once it touches the border. (Note the `W, S, A, D` keys will also need to be swapped.)

- Adding additional animations to your car as the window being able to roll down.
- Let the wheels rotate quicker if the car speeds up and slower if the car slows down.

6 Implementation Details

- You need to use OpenGL version 3.3 for this practical.
- You may **not** use any of the built-in mathematical libraries within the `glm` package. This included matrix arithmetic. *Hint: You may use your practical 1 in this practical.*
- You may **not** use any of the built-in OpenGL functions to generate the shapes for you. You need to create each shape from first principles.
- You may **not** use any of the built-in OpenGL functions to perform the transformations of the shapes. You need to transform each shape from first principles, either explicitly or by using the matrix arithmetic techniques discussed in class.
- You may only use the following C++ and OpenGL libraries:
  - stdio.h
  - stdlib.h
  - iostream
  - iomanip
  - cmath
  - sstream
  - GL/glew.h
  - GLFW/glfw3.h
  - glm/glm.hpp

  You may also use the `shader.hpp` and `glad.c` files that assist with compiling and linking of shaders. Your code should be able to be compiled without the assistance of an IDE, i.e. the project needs to be able to be compiled from terminal.
- All your helper classes and files needs to be in the same directory of the main.cpp.
- Ensure that the title of the window of your program is your correct student number.

Assessment Criteria	**0**	**1**	**2**	**3**	**4**
Car requirements [10 marks]					
Two toned body	There is no body for the car	There is a single toned body for the car		There is a two toned body for the car	
At least different windows shapes	There are no windows	There is a single window	There are at least two windows but not different shapes	There are at least two windows with different shapes	
At least two different wheels with rims	There are no wheels with no rims	There is at least a single wheel but no rims	There are at least two wheels but no rims	There are at least two wheels with a single rim	There are at least two wheels with rims
Shape requirements [12 marks]					
At least two rectangles/ squares	There are no rectangles/squares	There is at least one rectangle/square	There are at least two rectangles/squares		
At least two triangles	There are no triangles	There is at least one triangle	There are at least two triangles		
At least two high polygon circle	There are no high polygon circles		There is at least one high polygon circle		There are at least two high polygon circles
At least two low polygon circle	There are no low polygon circles		There is at least one low polygon circle		There are at least two low polygon circles
Colour requirements [12 marks]					
Wheels of the car is the correct colour	Not correct colour			Correct colour	
Rims of the car is the correct colour	Not the correct colour or same as the wheels			Correct colour	

Assessment Criteria	**0**	**1**	**2**	**3**	**4**
Windows of the car is the correct colour	Not the correct colour or same as body of car			Correct colour	
Background colour is distinctive	Not distinctive colour or black			Correct colour	
Transformation requirements[16 marks]					
Horizontal translations	The car is not able to move left or right		The car is able to only translate in one direction		The car is able to translate in both directions
Scaling	The car is not able to be scaled		The car is able to only scale in one direction		The car is able to scale in both directions
Vertical translations	The car is not able to move up or down		The car is able to only translate in one direction		The car is able to translate in both directions
Wheel and rim rotations	The wheels and rims do not rotate	Wheels and rims only rotate when a key is pressed	Only one of the wheels and rims move		Both of the wheel and rims move
Wireframe[10 marks]					
Render	No wireframe		Partial wireframe		Correct wireframe
Colour	No wireframe		Partially coloured		Correctly coloured
Transformations	No wireframe	Partial transformations	Correct transformations		

Table 1: Marking rubric

7 Submission

You are required to submit on ClickUp under the appropriate submission link. In the archive that you submit, include a makefile and compiling instructions such that the program can be compiled and executed by the markers if needed. *Failure to upload to ClickUP will result in you forfeiting all marks for this practical.* No exceptions will be made on this matter.

8 Demo Instructions

1. You will first be required to download your submission from ClickUP.
2. You will then demo your practical to the tutor.
3. In the presence of the tutor, you will be required to upload the archive you downloaded from ClickUP to FitchFork. *Failure to upload to FitchFork will result in you forfeiting all marks for this practical.* No exceptions will be made on this matter.