SWS3005: Real-Time Graphics Rendering (2024)

Assignment #1 (Individual Work)

Release Date: 4 July 2024, Thursday

Submission Deadline: 9 July 2024, Tuesday, 11:59 PM

LEARNING OBJECTIVES

OpenGL viewing, OpenGL transformations, hierarchical modeling, and animation. After completing the programming assignment, you should have learned

- how to set up the view transformation (camera position and orientation) in OpenGL,
- how to set up perspective viewing in OpenGL,
- how to use the OpenGL transformations for modeling, and
- how to use the OpenGL transformations for animation.

TASKS

Please download the ZIP file **`sws3005_2024_assign1_todo_(*).zip`** from the **`Canvas > SWS3005 > Files > Assignments`** folder.

You are provided with an **incomplete** C++ application program **`main.cpp`**, and your job is to complete it according to the requirements.

You may try the **completed executables: `main_done.exe`** (for Windows), and **`main_done`** (for macOS) found in the same ZIP file. (On macOS, you may need to use the command **`sudo chmod +x main_done`** to give the file execute permission before running it.)

Please read the instructions shown in the console/terminal window to learn how to operate the program. When you run the program, you should see a spherical planet at the center of the window. There are a dozen cars moving on the planet surface, and each car moves in a different great circle (you can search the web to find out what a ***great circle*** is), and they have different speeds and colors.

You may try the followings on the program:

- resize the window and see what happens,
- press the Up, Down, Left or Right arrow key to change the camera's position,
- press the Page Up or Page Down key to change the camera's distance from the planet,
- press the 'P' key to pause/resume the animation of the cars.

You will notice that the camera is always looking at the center of the planet. With respect to the planet, the camera's position can be expressed as latitude and longitude, and its distance from the planet's center. When the Left or Right arrow key is pressed, the camera's longitude decreases or increases, respectively; and when the Down or Up arrow key is pressed, the camera's latitude decreases or increases, respectively. Note that the camera's up-vector is always pointing north.

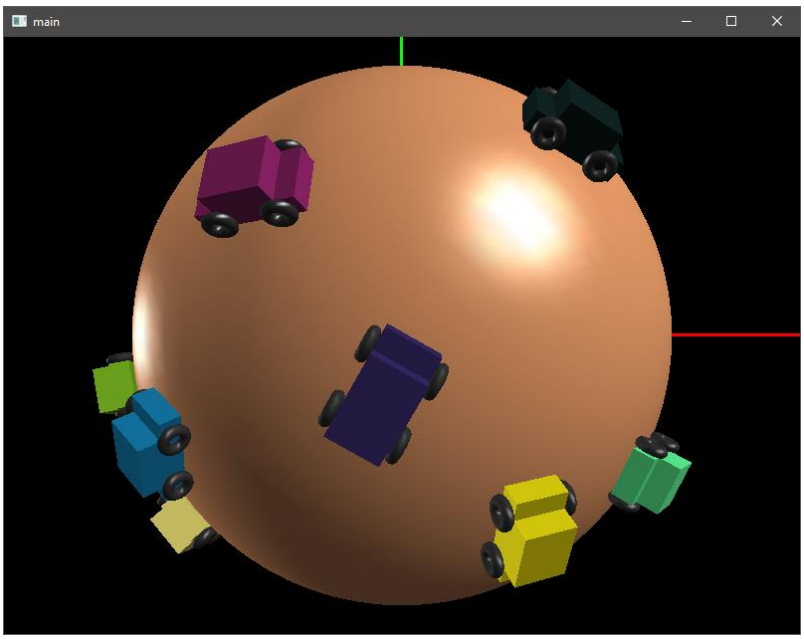

Figure 1

Follow the following instructions to complete **`main.cpp`** as required. You must not add or change any other files.

1) Study the source program very carefully.

2) Complete the **`DrawOneCar()`** function. The function must draw the car using only GLUT functions such as **`glutSolidCube()`**, **`glutSolidTorus()`**, **`glutSolidCone()`**, and **`glutSolidSphere()`**. **You should not directly use any OpenGL geometric primitive**. You should make use of the OpenGL 3D transformation functions to help you resize, orientate and position the parts. The functions **`glPushMatrix()`** and **`glPopMatrix()`** are very helpful for you to save and restore the current transformation before and after drawing each part. You can design your cars any way you like as long as they look like cars. More details about the GLUT functions can be found at https://www.opengl.org/resources/libraries/glut/spec3/spec3.html.

3) Complete the **`DrawAllCars()`** function. This function draws each car at the correct position on its great circle. Note that any great circle on a sphere centered at the origin can be defined as follows. Let C be the great circle in the $y = 0$ plane, and let v be a vector in the x-z plane, and let θ be an angle. If we rotate C about v by θ, we get another great circle of the sphere. All great circles of the sphere can be obtained by varying v and θ.

4) Set up the correct perspective viewing volume in the **`MyDisplay()`** function. You should use the **`gluPerspective()`** function. The near and far planes should be set near the planet's surface, yet still do not clip off any part of the planet and cars. The near and far planes should vary with the eye's distance from the planet's center. You should make use of the value of the predefined constant **`CLIP_PLANE_DIST`** to position your near and far planes.

5) Set up the correct view transformation (camera's position and orientation) in the **`MyDisplay()`** function. You may use the **`gluLookAt()`** function, or you can use an alternative method.

6) Complete the **`MyTimer()`** function. You should use the GLUT timer callback to control the speed of the animation by maintaining a constant frame rate (**`DESIRED_FPS`**). Refer to https://www.opengl.org/resources/libraries/glut/spec3/node64.html to find out more about the GLUT function **`glutTimerFunc()`**.

More detailed instructions can be found in the unfinished program code. You may add additional constants, global variables and functions to the given program.

DO NOT HARD-CODE VALUES. You should write your code in such a way that when the values of the named constants (defined in the beginning of the program) are changed to other valid values, your program should function accordingly. For example, if the car's size is changed, the tyre size should vary proportionally.

A Visual Studio 2017 solution **`main.sln`** (or Xcode project **`main.xcodeproj`** on macOS) is provided for you to build the executable program. In this assignment, **you are not required and must not change any other C/C++ source files** besides **`main.cpp`**.

Besides GLUT (or FreeGLUT), you should not use any other third-party libraries. Your code must compile with either the MSVC++ 2017 (or newer) compiler on Windows, or Clang on macOS.

GRADING

The maximum marks for this programming assignment is **100**, and it constitutes **20%** of your total marks for the course. The marks are allocated as follows:

- **30 marks** — drawing of a car in the **`DrawOneCar()`** function.
- **25 marks** — putting each car at its correct position on its great circle in the **`DrawAllCars()`** function.
- **10 marks** — setting up the correct **perspective viewing volume** with tight and varying near and far planes.
- **25 marks** — setting up the correct **view transformation** (camera's position and orientation).
- **10 marks** — correct **`MyTimer()`** function.

Note that marks will be deducted for bad coding style. If your program cannot be compiled and linked, you get 0 (zero) mark.

Good coding style. Comment your code adequately, use meaningful names for functions and variables, and indent your code properly. You must fill in your **name**, and **NUS User ID** in the **header comment**.

SUBMISSION

For this assignment, you need to **submit only** your completed **`main.cpp`**.

You must put it/them in a ZIP file and name your ZIP file ***nus-user-id*`_A1.zip`**. For example, if your NUS User ID is **`t0912345`**, you should name your file **`t0912345_A1.zip`**.

Note that you will be penalized for submitting non-required files.

Submit your ZIP file to **`Canvas > SWS3005 > Assignments > Assignment #1`**. Before the submission deadline, you may upload your ZIP file as many times as you want. **We will take only your latest submission**.

DEADLINE

Late submissions will NOT be accepted. The submission page will automatically close at the deadline.

——— End of Document ———