# Digital Systems

EEE4084F

[30 marks]

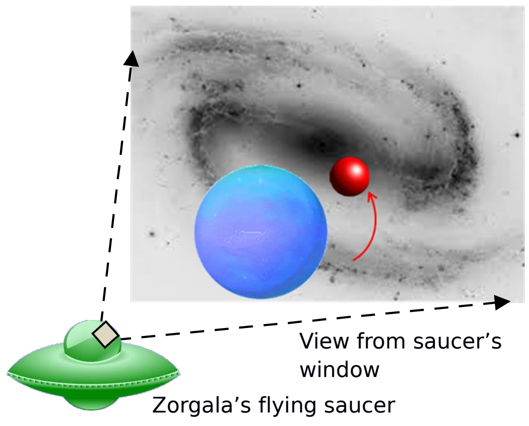
# Practical 3: Simulation of Planet Vogela with its Moon and Vogel Spiral Star Formation using OpenGL, OpenMP, and MPI

## Introduction

The objective of this assignment is to simulate an alien planet and starscape scene implemented using OpenGL for the graphics and MPICH to prove a parallelized solution to get a speed-up, using multiple processors beyond that which is obtained with a single GPU on one processor. The description 'An Alien Scene' below is what you are expected to simulate, with the objective of providing an animation with a moon orbiting a planet.

## An Alien Scene

Zorgela is a legal citizen, indeed one of many Zogels, on the planet Vogela. Annoyingly enough, Vogela has a big red moon, Rogela, that often gets in the way when Zorgela and his special squishy friend Gorgelis is on the planet looking up at the pretty stars (or, as Zogels call them, 'Stargelas') of the neighboring galaxy Shogela. But this works out fine for Zorgela because it gives him an excuse to take his cool green saucer out for a spin with Gorgelis so they see the show from some way off, i.e. something like a drive-in that Zogels invented centuries ago but alas forgot about.



## Objectives and Assumptions:

So the aim of this prac is to simulate what the view will look like from Zorgela's saucer some way off from the planet Vogela, with a possible fake addition that the moon Rogela spins round the planet at rather high speed – the faster you get the moon spinning round the planet the better!

Assume Vogela (planet V) is stationary in relation to the starscape Shogela, and the stargelas are simulated by spheres. Rogela (moon R) follows a simple circular orbit of constant speed. The galaxy Shogela has its stargelas arranged in the form of a Vogel Spiral (a Vogel spiral is in reality an effective approximation of how stars in many real galaxies are actually arranged).

The pseudo-code on the next page specifies how to positions spheres into the shape of a Vogel spiral.

*Pseudo code for Vogel Star formation algorithm for positioning spheres into a Vogel spiral*

```
Set integers i and r parameters to zero
Set the golden ratio for Vogel Spiral phi to (1 + sqrt(5)) / 2.0
Set a double theta value to 0
Declare variables/coordinates x and y as double types
Set OpenGL 3D Color to a light color
Loop from j=0 to number of stars
    Set i to j + 1
    Set r to sqrt(i)
    Set theta to (i * 2 * PI  / (phi*phi))
    Set x = (cos(theta)*r) ;
    Set y = (sin(theta)*r);
    Create OpenGL PushMatrix
        Position the star origin at coordinate (x/10, y/10, -10.0)
        Create a solid sphere of radius 0.01 to represent the star
    Create OpenGL glPopMatrix
End Loop
```

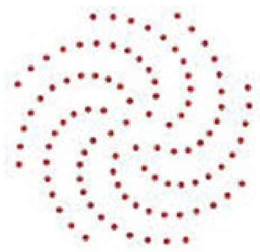
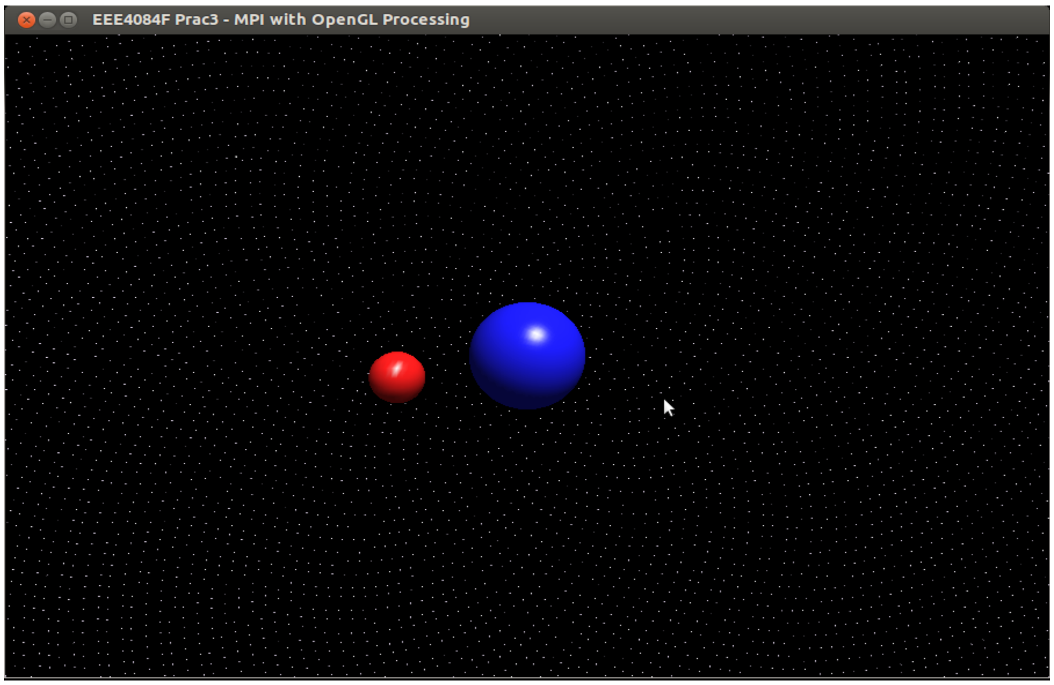


Illustration of how just a few points arranged into a Vogel spiral looks

## Things To Do and the Software Tools to Use

This practical aims at using OpenGL (Open Graphics Library) for simulating the planet with its revolving moon and stars in the background. The Vogel Spiral Star formation algorithm is computationally expensive for simulation of one thousand or more stars (where each star is a sphere that happens to have a reflective surface that makes processing more demanding). You are therefore required to implement the parallel version of this Vogel spiral Star formation using OpenMP, and compare it with the sequential and also an MPI parallel version of the same code.



Screen capture of the sample solution for the starscape simulation

Although MPI can be used together with OpenMP, we will use each on its own in this practical. OpenMP[1] provides compiler directives, library routines, and environment variables that allow users to create and manage parallel programs while permitting portability. These compiler directives extend the standard C, C++, and Fortran compilers to provide Single Program Multiple Data (SPMD) constructs, where the same program can simultaneously run on multiple platforms. These OpenMP directives allow for tasking constructs, device constructs, work sharing constructs, and synchronization constructs. These constructs provide support for sharing and privatizing data. MPI, on the other hand, is a standardized and portable message-passing system designed to provide language-independent communications protocol used to program parallel computers. MPI programs always work with processes, referred to as processors in the code documentation and usermanual. Typically, for maximum performance, each CPU (or core in a multi-core machine) will be assigned just a single process. For our simulation of the Shogela starscape, we will divide the scene rendering into 4 processors to speed-up the time to generate the visualizations.

## Tasks and Submissions:

Implement tasks 1 to 3 below and submit your source code file(s), which should be arranged nicely in a directory that is compressed into a zip file for easy online submission using the appropriate Vula assignment. The PRAC03_OGL_OMP_MPI folder (that accompanies this assignment description) contains the following source code files:

| **File** | **Description** |
|---|---|
| eee4084f_ogl_simple.cpp | Simple OpenGL application to simulate the planet V and its moon B |
| eee4084f_ogl_omp.cpp | Simple OpenGL application to simulate planet V and its moon B to be parallelized using OpenMP parallel for loop. Contains "TODO" pointers showing which sections to implement in parallel. |
| eee4084f_ogl_mpi.cpp | Simple OpenGL application to simulate planet V and its moon B to be parallelized using MPI. Contains "TODO" pointers showing which sections to implement in parallel. |

A makefile is provided that compiles the applications. A single "make" will clean and compile all the three sources specified above. To run the application, the "./" command and application name will be used. For instance, to run the eee4084_ogl_simple application the command "./eee4084_ogl_simple" can be used. Study the makefile and note the libraries to be linked and the command for compiling MPI code – the "mpicc" compiler is used instead of "g++" or "gcc" as with OpenMP or OpenGL only applications, but libraries (glut, gl, gomp, etc.) are still linked in the usual way.

**Part 1: Starting Out in OpenGL: A simple scene rendering program**

Follow the sub-parts #1 to #3 below…

**#1** Study the OpenGL Code Provided. Five main functions must be defined in the implementation of a simple planet V and its revolving moon simulation. These functions are as follows:

- Display() : In this function, one first clears the rendering area using glClear([mask]). Parameter [mask] will enable color writing and set the depth buffer. Next one sets the color of the objects to

[1] http://openmp.org/wp/

be drawn after clearing the screen. Color is set using glColor3d(r,g,b) for 0≤ r, g, and b ≤ 1. The next step is to draw objects on the rendering area, there are several objects including spheres (wire and solid), cubes, lines, wires, etc. To draw objects, the best way is to push then into a stack and pop them for efficient processing by OpenGL library. In this practical we first draw planet V and its moon R as follows:

*Drawing planet V and its moon R*

```
glPushMatrix();
        glTranslated(x,y,z); //Position cursor at this position
        glRotatef(angle, x,y,z) // Rotation of the planet
        glutSolidSphere(r, slices, stacks) //r=radius, slices=lines of longitude, stacks=lines of latitude

        // Change Color to red here  -- you need to figure out the simple way to do this one ☺
        // Position the curser using
        glRotatef(angle, x,y,z) // Rotation here becomes revolution of the moon around planet V
        glutSolidSphere(r, slices, stacks) //r=radius, slices=lines of longitude, stacks=lines of latitude
glPopMatrix();
```

- Resize(): This functions resizes the screen and provides viewport to set the frame area on the display screen for viewing the planet V, moon R, and the spiral stars. We use glViewport(x, y, width, height). If you're finding OpenGL interesting, then try finding out about the other functions that are called from in this function.
- Mouse(): Defines the behavior of the OpenGL application when a certain mouse button is clicked once. For instance, for our starscape simulation, left clicking the mouse starts the revolving of the R moon in an anti-clockwise motion, while right clicking start the revolving in a clockwise motion and center click reset the position of the moon R to its initial position.
- Keyboard(): This function is similar to mouse() function but define certain keys as an indication to the direction of the revolution. Key 'f' starts moon R revolution anti-clockwise while key 'b' start moon R revolution clockwise, and key 'r' resets the position of the moon R to its initial position. These keyboard() and mouse() functions are pretty easy to follow. Consult the tutor if you don't understand how they work.
- Main(): This is where command line parameters are captured and the OpenGL window is initialized, callback functions[2] for rendering, resizing, mouse, and keyboard are installed, and OpenGL main loop is called. Check the structure of the main function. Node that to provide clear 3D visualization of the planet, its moon, and stars, some glut lighting feature is instantiated.

[2] So what is a callback function? It is simply a function you implement in your code, which you don't necessarily call anywhere in your own code, but it is called from somewhere else such as a library linked to your program.

**#2** Using openmp timer (omp_get_wtime()), record the time it took for this simple planet V and revolving Moon B simulation. This is before addition of the stars. Observe how fast the moon revolves around the planet V.

**#3** In the display() function, just after drawing planet V and moon R, implement the for-loop for the Vogel Spiral model for star formation as described earlier. This should draw planet V and the stargelas of the galaxy Shogela. The Vogel Spiral model will assist in calculating the x- and y- axis values while the z-axis values are all set to constant value "-10". Set the number of stars to 1000 initially, and then vary the number to 10 000, 100 000, and 1 000 000. Measure the time it takes to execute this spiral model sequentially and observe how slow the moon revolves now that the stars are there. (PS: If you want to be fancy, without scoring extra marks, create a copy of your file with a different filename and apply z transforms as well in a spiral shapes so see what kind of weird arrangement the stargelas would look like if they weren't all neatly arranged in a plane; you probably want to put limits of about -5 to -10 on z and use only a few hundred stargelas otherwise the screen would get too full to see the patterns.)

**Part 2:** ***Using OpenMP to Parallelize Scene Rendering and Measuring Speed-ups***

Follow the numbered sub-points below to get your code into a form for applying OpenMP extensions…

**#1** Implement the parallel OpemMP for-loop for the Vogel Spiral star formation, i.e. drawing the Stargelas.

**#2** Measure execution time varying number of stars as above: 1 000, 10 000, 100 000, and 1 000 000.

**Part 3:** ***Using MPI to Parallelize Scene Rendering and Measuring Speed-ups***

**#1** Implement the MPI parallel version of the Vogel Star formation. Using OpenMPI to partition the rendering into four processors (i.e. Run each partition on the 4 cores of the i5 Machines in the Blue Lab, but you are welcome to try it on your Laptop/PC.). The structure of the MPI section is provided to you in the eee4084f_ogl_mpi_starform.cpp file.

**#2** Rewrite the mouse() function to allow the user to play around with different views/scenes (different angles of viewing the planet V scene) when the right click is performed. First right click provides different view angle, second provides another that has not been seen, and third does the same until it goes back to the original scene.

**#3** Measure the execution times using varying number of stars as follows: 1000, 10000, 100000, and 1000000. Notice the 4x speed-up.

In order to document your efforts, you need to provide a short report as described on the next page.

**Part 4: Short *Report***

Your report needs to document the main performance results you achieved. The report need only be a few pages, the length largely depending on how much code you need to put in to explain things clearly enough. Your short report needs to covers the following:

1. *Introduction:* Explain your implementation of the planet V simulation using the three programming models presented, sequential, parallel OMP, and parallel MPI. Well commented source code will make this easier as you could then simply copy and paste you code into this section, with just a bit of text at the start providing an outline of the how your code is divided up. Indicate relevant file names. (If you choose just to paste in your commented code, but it is lengthy, then just extract the important parts and put text such as "… does x …" in the parts that you leave out.)

2. *Results:* Provide sample runs of your program (screenshots) and provide a graph of speed-ups for each implementation and execution times of the planet V simulation using stars of 1 000, 10 000, 100 000, and 1 000 000.

3. *Conclusion:* Discuss your finding when comparing the OpenMP and MPI parallel version. Do you think there would be merit in providing a CUDA version of this parallel implementation; would such a thing be noticeably faster than the fastest of your OMP or MPI implementations? Explain you answer.

---

End of Practical Assignment