# COMP 410/510, Computer Graphics, Spring 2023

**Due Date: Sunday, April 2, 2023**

## Problem Description and Learning Objectives:

In this assignment, you will design and implement a basic graphics application that animates a ball bouncing horizontally on the screen (as shown above). This will be an interactive program that allows the user to select an object to draw (from a choice of two objects), to choose various drawing attributes for the object (such as color and polygon mode). Each time the user selects one of the drawing options, the image on the screen will be redrawn with the new choice. To develop this program you will need to learn how to use both **shader-based OpenGL** and **GLFW** 3D graphics libraries. The **GLFW** library includes functions for implementing event-driven input and display handling operations. This assignment will be an introduction to event-driven graphics programming. Your program must be developed using **shader-based** OpenGL and C/C++.

## Problem Specification

Your program will simulate and display a bouncing ball (either cubical or spherical) under gravitational force.The ball will initially be placed on top leftmost of the display window and then will start its fall (accelerated with gravity). The initial (horizontal) velocity will be a parameter to set. The ball will move rightwards with proper acceleration until it hits the bottom of the window, and bounce back slowing down gradually with each hit until it goes out of the window or its velocity becomes zero. Bouncing will continue as such. You may reduce the velocity by a fixed factor after each bouncing. During execution, the user may change the object type, color and drawing mode.

You can refer to the following page for physics of a bouncing ball under gravity (Note that in this assignment we will be ignoring the deformation of the bouncing ball): bouncing ball

Your program must handle user input from keyboard and mouse, and set the drawing modes as specified below:

Object type and drawing mode (wireframe or solid) must be set by using mouse button callback:

- **Object type** -- set the current object to be drawn; switch between the following two choices, whenever right button is pressed:
  - cube
  - sphere
- **Drawing mode** -- set how the triangles are drawn; switch between the following two choices, whenever left button is pressed:
  - wireframe (i.e., as lines)
  - solid mode

Object initialization, color setting, termination and help functionalities should be controlled through keyboard callback as specified below:

- **i** -- initialize the pose (top left corner of the window)
- **c** -- switch between two colors (of your choice), which is used to draw lines or triangles.
- **h** -- help; print explanation of your input control (simply to the command line)
- **q** -- quit (exit) the program

Your program must also properly handle the **reshape event**; so you must define your own reshape callback function.

**Transformations** (i.e, vertex displacements) and **projection** have to be implemented in vertex shader. Since we haven't yet seen how to shade surfaces, when you display your objects in solid (filled polygon) mode, you'll be able to observe them only as silhouettes.

You can use the default orthographic projection (the viewing volume being the cube centered at the origin with sides of length 2). Note that since reshape callback function is invoked when the window is first opened at the beginning of program execution, it is a good place to put all projection-related settings (that means you won't then need to set projection in init() function).

**Bonus (+10pts):** Load and display the 3D Bunny model from an OFF file as a third object type (bunny.off). So in this case you will be able to set the current object to draw, as one of the three choices (cube, sphere or bunny). Note that you may initially need to adjust the orientation and scale of the bunny model using a transformation matrix.

You can preview a given off file in 3D using the following web link: (https://3dviewer.net/). The off file in our case includes a vertex list first and then a triangle list as shown below. Off files are actually ASCII files and you can see their content using any text editor. But for your application you need to write your own loader:

```
OFF
4922 9840 0
1.7325 -9.534 24.02
1.7 -9.7711 23.8073
2.216 -9.723 23.6798
2.285 -9.5198 23.923
1.0534 -9.8046 23.8978
...
...
3 0 1 2
3 0 2 3
3 4 1 0
3 4 0 5
3 6 1 4
3 6 4 7
...
```

Note that the first two numbers in the header indicate the number of vertices and the number of triangles in the list, respectively (the third number is the number of edges, which you can simply ignore in this assignment). Then follows a list of x y z coordinates, and then a list of indices. On each indices row, first the number of vertices specifying that polygon is given, which is always 3 in our case, followed by the three indices themselves (hence each row represents a triangle of the model).

## Program Requirements

1. You are expected to design the program as an event-driven application that responds to keyboard, mouse and reshape events. Thus, you should follow the main program and function module model given in your textbook and lecture notes.
2. Use shader-based OpenGL.
3. Your program files must have documentation comments.

   Your program will be graded for:

   - good programming **design**,
   - good programming **style**,
   - good program **documentation,** and
   - **correctness**.
4. You may be required to run your program and demonstrate that it works.

You are required to submit only the source files, i.e., the code files, ready to be compiled and run. Please try to comply with the announced due date. You will upload your submission to blackboard under the Homework 1 assignment.

## Implementation Hints

1. Detailed documentation for **OpenGL functions** are described in your textbook and OpenGL 4 Reference Pages. I strongly recommend you to read Chapters 1 and 2 from your textbook (E. Angel).
2. Use the **GLFW library** functions as described in lecture notes. See also https://www.glfw.org/docs/latest/ and/or https://learnopengl.com/.
3. Your program can be based on the example codes from your textbook. You can directly use the given InitShader.cpp code to load your shaders as well as the provided header files. You may use the spinCube example from lectures as a skeleton code to modify as well as the Appendix Example 7 from the textbook to draw a sphere in terms of triangles (read also Chapter 5.6, page 280 from the textbook for description of the subdivision method to approximate a sphere).
4. Note that the viewing volume (that you set by Ortho() function defined in mat.h) is eventually fit into the display window. So the bondaries of the window match the boundaries of the view volume. If you use the default viewing settings, the displacement of the ball should be along x and y axes.
5. To switch between different object types (cube or sphere or bunny), you can use vertex arrays. A cube is easier to implement so you can start with that.
6. You can use glPolygonMode() function to switch between wireframe and solid types of object rendering.
7. You can use glfwGetTime() in the main event loop to get animation as in lecture codes. With this function, it is possible to have direct control on the frame rate of the animation. You should also be able to adjust it by increasing or decreasing the initial speed of the ball and/or the amount of stepwise displacements, so that you can try to make it look as physically correct as possible.
8. Adjust the size and the initial speed of the ball properly (by playing with the state variables) so that animation looks nice. One would expect the ball to bounce at least 3 or 4 times before getting out of the window.
9. You will need **global variables** that hold state values for your program, e.g., a code number for the current object type to draw, a code value for the current draw mode (wire frame or solid), etc. The callback functions will simply update these global variables. The **display()** function will read the current state to determine what object to draw, how to draw it, etc. Note, we are forced to use global data even though that is not good software design because of the way the GLFW library software designers specified the input callback functions.
10. The various drawing mode control options listed above (type of object, draw mode, color, etc.) should be handled by using GLFW functions. You will definitely need the following:
    - glfwSetKeyCallback()
    - glfwSetMouseButtonCallback()
    - glfwSetFramebufferSizeCallback()

    - etc

    Each of these functions must be called one time at program initialization with a parameter that is the name of your callback function. Your actual callback functions must have the appropriate prototype.

    You are welcome to design your own user input technique for setting these state variable values with some other combination of key inputs or with mouse inputs. Make sure that you explain them in the help functionality (via "h" key).
11. You won't need any rotation and zooming in this assignment except maybe for the initial pose correction of the bunny model.
12. Projection will be implemented in terms of matrix multiplication operations in vertex shader. You can create the corresponding 4x4 transformation matrices by using the functions given in mat.h file. These matrices should be created in the application and then sent to shader as uniform variables. For example, you can make use of Ortho() function that returns a 4x4 orthographic projection matrix (the viewing volume is specified through the function arguments).
13. **Avoid using OpenGL functionalities deprecated and removed by OpenGL 3.1** (you can actually use any function that is used in the textbook examples, and possibly many others).
14. Be creative and inventive!!! **Extra credit** will be given for use of different OpenGL functions in creative ways and for a clever design. Additionally you may want to add other control options or design and implement a better user interface.