(B) Programming Part (68%):

Description:

3D object representation and construction (i.e., geometric modeling) is one of the major tasks in computer graphics. We can model the complex graphical object by using a number of simple models (e.g., primitive models) through a series of affine transformations in a hierarchical structure. In this assignment, you are required to create geometric models as an implementation of the 3D object modeling.

The goal of this assignment is to enhance the concept of 3D viewing, and create the special effects using the basic geometric transformations. The realization of 3D viewing environment through your implementation will make you better understand some of the most important features of the graphics applications in 3D scenes display and rendering. You are expected to make use of some OpenGL commands to achieve the task.

There are two parts in this assignment:

Part A: 3D viewing with an active virtual camera. You are to create a 3D viewing environment with a perspective view using an active virtual camera. The camera's position, orientation and field of view are adjustable. You will implement several basic functions of the virtual camera including Roll, Pitch, Yaw and Zoom (translate the virtual camera along its viewing direction). The scene that you are going to create is a simplified version of solar system. The scene should have at least four planets: Sun, Mars, Earth, and Moon.

Part B: Construct a graphical model by composing a number of primitive models:
You are required to implement a series of geometric transformation to construct a "lever" object using the primitive models "cylinder" and "sphere", and perform the rotation of the "lever" in a specified orientation.

Your implementation:

i. (a) Create a menu (or define keys) for camera functions:
roll+, roll-, pitch+, pitch-, yaw+, yaw-, slide+, slide-.
Note: "+" and "-" define the opposite orientations.
(b) Create menu for one application instance: lever objects rotation ("levers rotation")

ii. Define four viewports (V1, V2, V3 and V4) on the screen:

V3	V4
V2	V1

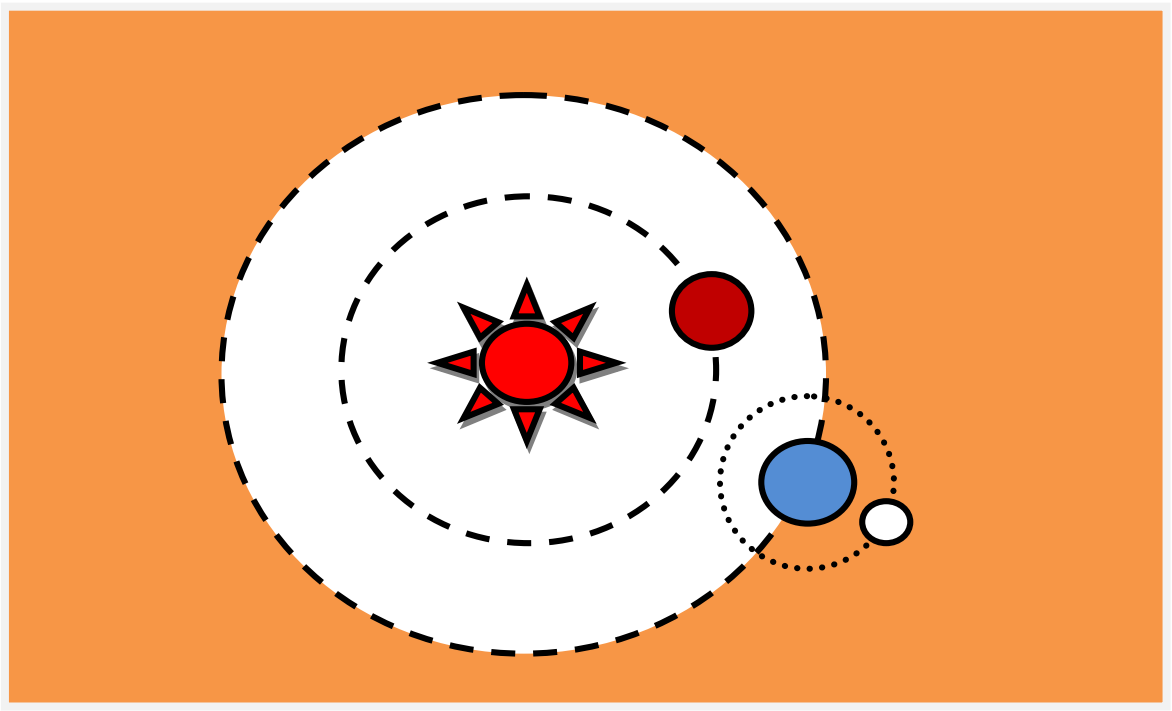
Lower-right (V1): the active virtual camera view
Lower-left (V2) / upper-left (V3) / upper-right (V4): the static virtual camera view at three locations to view the constructed object (e.g., lever model).
V2: Top view (looking toward the origin from +y position along the y axis).
V3: Side view (looking toward the origin from +x position along the x axis)
V4: Front view (looking toward the origin from +z position along the z axis).

Note: all views in V1- V4 are defined as perspective projections.

1. Part A: 3D visualization of a Solar system with an active virtual camera

(1) For V1:

a. Place the virtual camera in the initial position: (x, y, z) = (0, 0, 40), with the up-vector (upx, upy, upz) = (0, 1, 0) and looking at the origin point (0, 0, 0) by using the command gluLookAt(…).
b. Set the perspective view using gluPerspective(…)
c. Draw four planets with four wireframe spheres using GLU commands gluSphere(). The system includes a center Sphere (**Sun**), surrounded by two spheres **Mars** and **Earth**. Both spheres are also self-rotated individually. The smallest sphere (**Moon**) is self-rotated meanwhile it rotates around the Earth. (Shading and texturing are not required at this stage).



d. Perform the camera operations accordingly by adjusting the parameters of gluLookAt(…):

Roll: rotate camera around n axis
Pitch: rotate camera around u axis
Yaw: rotate camera around v axis

In each rotation (Roll or Pitch or Yaw), the step angle is: 10 degree

e. Translate the camera along its own viewing axis (i.e., n axis).
So that the camera can slide-in and slide-out (e.g., move
Forward & backward) to achieve the zooming effect.
The navigation of a virtual camera is expected to be implemented in this part.

2. Part B: hierarchical modeling

(1) Set three static cameras (i.e., viewpoints) at three locations using gluLookAt(…) in order to view an object (e.g., lever model) from three view points, and display in three viewports of the screen:

V4: Front view (looking toward the origin from +z position along the z axis).
Viewpoint (x, y, z) = (0, 0, 40), view-up vector is along the positive Y direction.

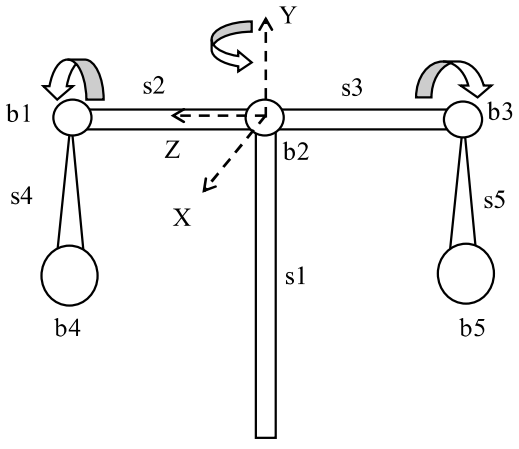
V3: Side view (looking toward the origin from +x position along the x axis)
Viewpoint (x, y, z) = (40, 0, 0), view-up vector is along the positive Y direction

V2: Top view (looking toward the origin from +y position along the y axis).
Viewpoint (x, y, z) = (0, 40, 0), view-up vector is along the positive Z direction.

(2) For V4, V3, and V2:

(1) Create a "lever" object which is constructed by four cylinders (s1, s2, s3, s4 and s5) and five spheres (b1, b2, b3, b4 and b5) using the translation and rotation operations (glTranslatef(…), glRotatef(…)). The vertical cylinder s1 is a support of the horizontal cylinder s2-s3.

The length of s1 is 20; the length for s2, s3, s4 and s5 is 10; the radius of sphere (b4 and b5) has twice size of b1, b2 and b3; the origin can be set at point b2.



(2) Perform the object motion by clicking the menu ("Levers rotation"). Each click will perform one-step motion for s2 – s5 simultaneously. s1 is fixed without any motion.

(3) In each step, the motion of s2 – s5 is defined as follows:
Rotate s2-s3 (10 degree) around the center "b2" in the XZ plane.
Rotate s4 (10 degree) around point b1.
Rotate s5 (-10 degree) around point b3.

In general, the constructed model performs the following motion: s4 and s5 perform the local rotation in opposite directions around the two end points (b1 and b3) in the vertical plane. Meanwhile s2 and s3 perform a global rotation around the center "b2" in the horizontal plane.

Note: You can take s2-s3 as one cylinder.

(4) The motion of the constructed object must be shown in the three views V4, V3 and V2 simultaneously.

(5) The ground plane must be displayed in V4, V3 and V2. The ground plane is fixed without any motion.

Hand-in

- **Code package:** Your program package and the assignment report must be compressed in a ZIP file, and submit it to the digital drop-box of the blackboard.
- **Write-up (report)**: You should document the way in which you solved the problem. This documentation should describe your solution so that the reader understands the problem that you are solving and then understands the code that you hand in. It should include
  (1) problem statement
  (2) algorithm design
  (3) major codes of your own implementation
  (4) sample images

Mark Distribution

Part A (28%):
(1) Display the simplified solar system in V1. (18%)
(2) Demo virtual camera functions: (10%)
Roll, Pitch, Yaw, Camera moving forward and backward

Part B (32%):
(1) Display the constructed object and ground plane in three viewports (perspective view from top, side and front) (16%)
(2) Perform the object motion: One horizontal rotation and two vertical rotations are running simultaneously. (16%)

Write-up (8%):
(1) Clearly explain your code design, and the solution to the problem you solved (8%)

Tips on how to complete the assignment:

- Read textbook Ch.7 for the coordinate system transformation
  (Especially the (u, v, n) camera coordinates system generation).

world-coordinate → view-coordinate (camera coordinate) → Projection coordinate

- Read OpenGL red book (Ch.3) for the commands:
  glTranslatef(..), glRotatef(..), glLookAt(…), glPerspective(..)

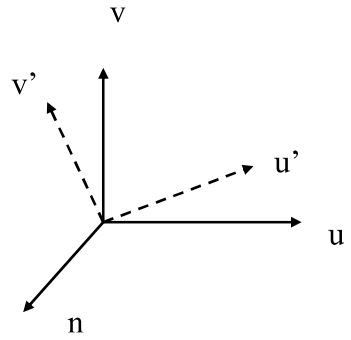
- In order to perform the roll, pitch and yaw operations, you need to apply 3D rotation matrix to calculate the up-vector values and the look-at point (centerx, centery, centerz) for calling gluLookAt(..).

  The up-vector and the look-at point can be derived by the rotation of vectors *u, v, n*.
  The initial vectors to define the view coordinate system are set as:
  u= (1, 0, 0), v = (0, 1, 0), n = (0, 0, -1);
  We need to track all these vectors when the camera is rotated.

  Roll, Pitch and Yaw operations are defined as follows:
  Roll: calculate the new u, v, n by rotating these vectors around n axis
  Pitch: calculate the new u, v, n by rotating these vectors around u axis
  Yaw: calculate the new u, v, n by rotating these vectors around v axis

  The new u, v, n vectors are obtained after the rotation calculation.

For example, rolling "alpha" degree of (u, v) vectors to (u', v') vectors:



u' = cos(alpha)*u – sin(alpha)*v
v' = sin(alpha)*u + cos(alpha)*v
n' = n

Now the parameters for gluLookAt(..) are ready: up-vector is the new v-vector; look-at point is located at the point by translating the eye position by *n*-vector in the viewing direction.

- In order to translate the camera along the viewing direction (n axis), you can calculate the new position *eye = eye + n*. "*n*" is the unit vector along the viewing direction (*n* axis).

- In order to construct an object from three cylinders and the motion of the object, you need to read OpenGL red book p107 – 109 carefully to make sure that you understand the order of the transformations operated in OpenGL:

For example:
If the following commands are called:

```
glLoadIdentity();        // Matrix I
glRotatef(..);           //Matrix R
glTranslatef(..);        // Matrix T

Draw_object();
```

This code implements drawing an object by translation first, rotation second, then multiply the identity matrix I, which is INVERSE order of the program execution!!

- Use glPushMatrix() and glPopMatrix() to save the defined matrix. See examples in page 144 and page 147-148 of the OpenGL red book.
- Cylinder and sphere objects can be easily created by OpenGL tool gluCylinder() and gluSphere().

- Read OpenGL red-book about the glViewport(..).

Good luck!