**CS606: Computer Graphics / Term 2 (2021-22) / Programming Assignment A3+A4**
International Institute of Information Technology Bangalore

**Announcement Date: Mar 28, 2022**
**Submission Deadline (A3): 11:59 pm IST, Apr 17, 2022**
**Submission Deadline (A4): 11:59 pm IST, Apr 27, 2022**

---

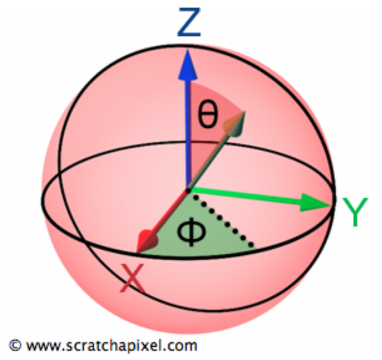
## Part 1 (Assignment 3)

**Summary:** 3D rendering with lighting and shading, and rotation using quaternions
Learning Objectives:

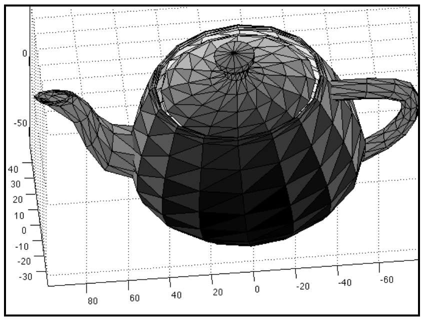
- Experimenting with lights and Blinn-Phong lighting model
- Experimenting with Gouraud and Phong shading models
- Using quaternions for rotation

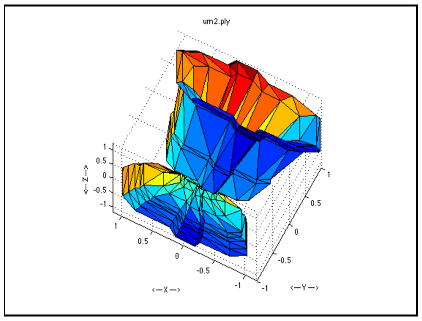
**Assignment: Illuminator**
**Assignment-3 Specifications**

1. Generate or import a sphere and a teapot/urn model that can be used here.
    a. Sphere mesh model can be created using polar coordinates by incrementally changing azimuth angle ϕ (longitude) from -π to +π, and elevation angle Θ (latitude) from -π/2 to +π/2. This can be used as parametric mapping for Part-2 for texture mapping.



    b. The teapot or the urn mesh models are available in .ply format at https://people.sc.fsu.edu/~jburkardt/data/ply/ply.html These are in ASCII format with the logic for file reading explained in the website.
    Pick one of the two meshes for Part 1.





2. Add a point light source corresponding to each mesh model, with different colors for ambient, diffuse, and specular components. Assign different material properties for each of the mesh objects, such as the three reflectance coefficients, $k_a$, $k_d$, $k_s$ for ambient, diffuse, and specular, respectively, choice of material color, and distance attenuation terms, i.e. $(a+bd+cd^2)$ where a, b, c are constant, linear, quadratic terms, respectively, and specular reflectance exponential value $\alpha$. Compute the local illumination model using all the light sources for all the surfaces in the scene.
    a. The light source corresponding to a mesh must be always present within a bounding box, 1.25 times that of the bounding box of the mesh, where the bounding boxes of the light source and of the mesh are centered at the centroid of the mesh.
3. Give single-digit numeric IDs other than “0”, “1”, “2” to meshes, to be used for selecting the mesh. Set the default selected-mesh-ID as “2”, implying none-of-the-meshes have been selected.
4. Set the **mesh-transformation mode** using key “m” to perform the affine transformations (translation, and rotation – excluding scaling) of the objects in the scene. When objects translate, their corresponding light source must translate using the same values.
    a. After activating this mode, select a mesh for affine transformation.
    b. Each affine transformation on the selected mesh can be done in the “m” mode using keys or mouse buttons, except for rotation, which must be done using the left mouse button. You are free to design the user interaction rules for translation and scaling. Commonly used patterns are the right mouse button for translation, and +/- keys or the middle mouse button (where usable) for zooming (which is not needed here). The idea is to use intuitive or commonly defined user interactions for these actions.
    c. The rotations must be done **using the trackball and quaternions**. This involves mapping the distances moved on the screen by pressing down the left mouse to angle and axis of rotation. If the rotation causes the violation of position of light source in 2a., then the light source can be reset to be placed at the centroid of the mesh.
    d. The rotation must be done about the centroid of the mesh object.

5. Set the **shading-model-choice mode** using the “s” key to toggle between the shading models, Gouraud, and Phong shading models.
    a. Gouraud shading is implemented using a vertex shader, and Phong using a fragment shader.
    b. Gouraud shading can be set as the default shader for all meshes.
    c. The toggling can be implemented only for the selected mesh.
6. Set the **illuminator mode** “l” key to be in the mode of moving light sources. Reserve “0”/”1” to turn off/on the *active* light source corresponding to the selected mesh.
    a. Light sources which are turned off must not be used in the local illumination model.
    b. Implement the local illumination model using the Blinn-Phong local illumination model.
    c. The default setting of all light sources is “on”. The light source corresponding to a selected mesh is considered the *active* light source.
    d. Use 6 keys, other than the ones used for mesh IDs and modes to manipulate translation of the active light source corresponding to selected mesh along +/- direction in the x, y, z directions. Specify this selection of keys in your README.txt
    e. The translation of the active light sources can be implemented only if it does not violate the position constraint of the light source with respect to the mesh, as specified in 2a.

__**Deliverables:**__

Submissions must be made on LMS.

1. The deliverables in a zipper folder must include a source code folder, a folder with screenshots, and a demo video not longer than 5 minutes. More details on the submission are given in the course handbook on the LMS course page.
2. If the deliverables are larger than the maximum allowable size on LMS, submit a text document with a repository URL (Google Drive, OneDrive, etc.). Care must be taken to not change this repository until grading is done.

Questions to be answered in the report:

1. What are your observations of the distance attenuation terms used for lighting on the sphere and teapot/urn models?
2. What are your observations about the change in the shading model on the two meshes?
3. What are your observations of the individual components of reflection in the Blinn-Phong model for each of the mesh models you have used?

## Part 2 (Assignment 4)

This part has a focus on animation, and is structured as a VR-like application. This part builds on Part 1 and adds the following features and objectives:

- Procedural animation of a scene
- Modeling a dynamic scene with a Scene Graph and relative transforms, and enabling animation through the scene graph
- Applying textures to mesh objects
- Managing lights and camera
- Handling collisions
- Basics of a VR application

**Assignment: Animation**
**Assignment-4 Specifications:**

In the following, the term “user” refers to the person running the program and manipulating controls such as keyboard and mouse, and hence triggering actions in the scene.
At the same time, as a VR-like application, we also have an *avatar* of the user as part of the scene. Since we do not have HMD’s etc, the user also controls the movements of the avatar (using keyboard or mouse), to simulate a VR setting.

**Scene and animation:**

We would like to do a simple simulation of a football game with 2 players - you and the opponent. You are represented by your avatar, and all actions of the avatar are controlled by the user with keyboard or mouse actions. The opponent is controlled by your program.

The scene consists of a large rectangular ground. The two opposite edges of the ground are the goal lines. The two players (your avatar and the opponent) are placed at arbitrary positions on this field. The sphere is now the football and can move around. The other objects from Part 1 (teapot, urn etc) are at fixed locations on the field and become obstacles. Set up at least 4 obstacles, replicating the objects from Part 1 if needed, and spread them out across the field.

The user chooses whether they are controlling the avatar or the opponent. Keyboard/mouse actions then control this player, such as with keyboard bindings that control the direction and speed of motion (e.g. turning left or right or continuing to go straight), and the actions kick/carry/dribble described below. The players can also turn their heads to the left or right, while they continue to move in a certain direction. (Note: lights or cameras attached to the player’s head would then need to move accordingly).

A player can handle the ball only when it is close (within a predefined distance). The player can then do one of 3 actions with the ball:

- "Carry" the ball. The ball moves with the player, fixed with respect to the player
- "Dribble" the ball. The ball moves on the ground a short distance in front of the player as the player moves around
- "Kick" the ball. The ball leaves the player and moves towards the goal line of that player.

When the ball is moving (dribble or kick), it follows these rules:

- the ball spins about its own axis at a slow rate
- if the ball comes near one of the fixed obstacles, it bounces off in a random direction and keeps moving in that direction.
- if it reaches the goal line, the game restarts
- Note: when a ball is being carried by a player, it does not spin or move relative to the player.

The mesh for the players/avatar can be any convenient object, and could be a single rigid object. You are not expected to animate the parts of the player objects. Thus, carry/dribble/kick are logical actions, and need not require movement of individual parts of the players' meshes.

This simulation can be implemented in one of two modes:

1. Single-player game: As described above, the user decides if they are controlling the user avatar or the opponent. All keyboard actions manipulate this player. The other player can move independently and does not affect the game
2. (Bonus marks) Two-player game: Here the user can manipulate either player, and players interact with each other.
    a. If the ball comes close to the other player (who would also be moving), it is "captured" by the other player, who can then manipulate the ball, with carry/dribble/kick.
    b. If the user is controlling the avatar, the opponent's actions can be triggered programmatically - for instance, the opponent always moving towards the ball, and kicking it once it is captured. You can try other approaches, including allowing the user to control both players simultaneously.

## Lighting

As in Part 1, the scene is illuminated by multiple lights. Modify the lights of Part 1 or use additional lights as follows:

A. One set of lights is fixed relative to the ground, and each light illuminates a small region around it (e.g. streetlights).
B. Another light is also at a fixed location relative to the ground, but tracks one of the moving objects (e.g. a searchlight).

C. Another light is attached to one of the moving objects (e.g attached to the ball or a player's head).

The user should be able to switch any of these sets of lights off or on. Adjust the intensity of the lights so that all the effects are clearly perceptible.

**Textures**

All the objects in the scene, including the ground, are to be rendered using textures.

A. The ball should be rendered using spherical maps for the texture. Please use texture images that will show the correctness of the texture mapping. Thus, use images such as photos of people/animals/natural scenes, world maps, checkerboards, etc, and avoid featureless textures like sand, wood, brick, fur etc.
B. Other objects can be rendered using either the default texture coordinates for the object (if it exists) or a simple planar map. Use any reasonable texture images for these objects.

**Camera**

The scene can be viewed through one of 2 cameras:

A. One camera is observing the whole scene from a certain height. This is the default view. This camera can zoom in and out, and also rotate about the scene. Use mouse movements and trackball to implement rotations of the camera about the scene. Note that all the scene objects, including the user avatar, should be visible in this view (if within the camera view volume)
B. Another camera is attached to the head of your avatar and simulates the avatar's view of the scene.

The application has only one viewing area, and the user can choose which camera feed should be shown in the view area. Use an appropriate keyboard control to toggle between the camera views.

**<u>Implementation Notes</u>**:

1. Use a Scene Graph to model the scene. This should capture the positional relationships between objects, if they exist. Each scene node contains information on the transforms needed to move the object relative to its parent in the scene graph. At a minimum, the ball should be modeled as a child of the player when in carry or dribble mode. All the scene objects should be part of the scene graph.
2. The process of animation involves recomputing the relative transforms from each successive frame that is to be rendered. Thus, you structure the code as a two-pass

process: the first pass of the scene graph is used to update the transforms, and the second pass is used to render the scene.

3. Use a simple collision detection mechanism, such as intersection of bounding boxes.
4. Use positional information from objects in the scene graph to control the position and orientation of lights and cameras.

**<u>Deliverables:</u>**

Submissions must be made on LMS and using the same structure as Assignment 3.

Questions to be answered in the report:

1. What kind of distortions do you notice with the texture maps you have used? What would be your approach to correcting them?
2. Provide a brief writeup (1-2 pages) on the design you used for:
    a. scene graph organization,
    b. how the position and orientation of lights and the avatar's camera are calculated,
    c. computation of animation including collision detection/avoidance.