Project 2: Transformation and Viewing

This document contains guidelines and instructions for required tasks and extra credit. Also, check out the Grading and **Submission Guideline** in the document. **Please carefully read and follow this guideline and refer to Lectures 6 – 9** to understand the necessary knowledge that you need to know to complete this project, as well as the terminology mentioned in this guideline.

1. Introduction

This project contains **two parts**: 1) you will build your first 3D character, animate their gait, and make them walk in the 3D scene using geometric transformations; 2) you will also implement dynamic viewing by changing camera parameters through keyboard input.

Objectives

1. Understand how to build 3D models using basic shapes (i.e., cylinders, spheres, cubes) through geometric transformations and OpenGL functions.
2. Learn how scene graphs and hierarchical modeling work and how to implement them to support the animation of complex structures.
3. Implement 3D viewing transformations using OpenGL functions.
4. Develop a user-controlled camera model and 3D viewing transformations.

Code Structure and Program Running

There are **two** Python files:

- **P2_transformView_main.py**: main function (drawing, viewing, keyboard input).
- **P2_transformView_model.py**: **2 classes**: **`Scarecrow`**: create the 3D model and animate its walking, containing all the parameters for limb and walking motions; **`Camera`**: contains all the viewing parameters needed for the camera control.

Notes:

- **Follow the provided code structure and function settings** for easy code review.
- **Use the designated keyboard input** for this project (see the Look-Up Table).
- This project is broken down into eight closely related tasks, which are built incrementally (subsequent tasks are built on top of the previous ones). Once you have completed all the tasks, **please go over EACH TASK and make sure each still works individually (with its designated key input) and together as a whole**.

2. Geometric Transformations

In this section, you will assemble a 3D character, Scarecrow, using basic 3D shapes, including cylinders, spheres, and cubes, by scaling, rotating, or translating them to proper positions and orientations in 3D. You will first build a basic Scarecrow. Then, you will upgrade your scarecrow by including more joints and limbs to support the subsequential tasks and allow it to walk in place and then walk freely in 3D.

2.1 Create a Scarecrow with a Rotatable Head

Task 1: Create a Basic Scarecrow

Create a “scarecrow” using the basic shapes as follows. The dimensions and desired positions of these shapes are illustrated in **Figure 1**.

- **Head**: Sphere, radius = 2.5, green;
- **Nose**: Cone - a cylinder with the base radius = 0.3, top radius = 0.0, and length = 1.8, red;
- **Torso**: Cylinder, radius = 2.5, length = 10, yellow;
- **Legs**: Cylinder, radius = 1.0, length = 12, red;
- **Arms**: Cylinder, radius = 1.0, length = 10, blue.

By default, when these geometries are created without any transformations, a sphere is centered at the origin, a cylinder/cone’s center line is aligned with the z-axis, and its base is in the x-y plane and centered at the origin.

<u>Where to write your code</u>: Write some code in `draw_Scarecrow()` within the class `Scarecrow` to transform these body parts, which have already been created in the function.

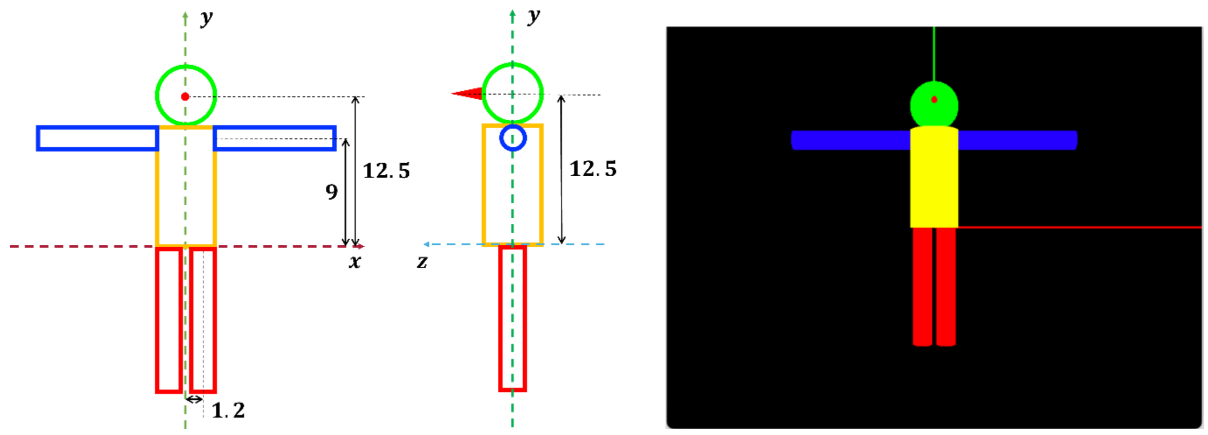

Figure 1: A basic Scarecrow.

Task 2: Rotate the Scarecrow's Head & Nose with Keyboard Input

Once the Scarecrow has been built in Task 1, you will **rotate its head (and nose)** so that it can look to its left and right **without** moving other parts of its body, **using the keyboard input**, specifically:

- Press the **key "i"** to rotate the head to **its left** (may not be your left), one degree at a time.
- Press the **key "o"** to rotate the head **to its right** (may not be your right), one degree at a time.
- Confine the angle to rotate the head (and nose), `head_angle`, with a range [-85, 85], i.e., the head **should not** be rotated further once it reaches -85 or 85 degrees from the center.
- Make sure the nose rotates with the head and looks like it attaches to the head the entire time.
- **Where to write your code**: for the transformation of the head and nose, add code into `draw_Scarecrow()`; your code for updating `head_angle` based on the key input should be written in a proper place in the main function.

2.2 Upgrade the Scarecrow with More Joints and Animate Walk-in-Place

Now, your Scarecrow wants to enjoy a bit more freedom and go to places to experience the world! Upgrade it with more joints and limbs as well as hands and feet. Then, you will create a *walk-in-place* animation by transforming its body parts into appropriate positions over time so it looks like it is walking without physically moving around.

Task 3: Upgrade the Scarecrow with More Joints

In **another function,** `draw_Scarecrow_Upgrade()`, based on your code in Task 1, make the following changes, **with the other body parts remaining in the same setup** as in Task 1. Refer to these new changes and dimensions in **Figure 2**.

- For each arm, instead of using one cylinder, use two cylinders (**upper arm and lower arm**) – the same radius and half-length, which are connected by a joint - a sphere of the same radius and the same color as the arm. **The total length of the arm should remain the same as the one in Task 1**. The initial angle between each arm and the y-axis is 15 degrees at the front view.
- For each leg, instead of using one cylinder, use two cylinders (**upper leg and lower leg**) – the same radius and half-length, which are connected by a joint - a sphere of the same radius and the same color as the leg. **The total length of the arm should remain the same as the one in Task 1**.

- Attach a pair of hands to the end of the arms. Each hand is a sphere with a radius of 1.1 and the same color as the head.
- Attach a pair of feet to the end of the legs. For each foot, use `glutSolidCube()` to create a cube of size 1.5, see **Figure 2**. Then, scale the cube to approximate a regular foot proportion, i.e., longer (x 1.8) along its longitude and flatter (x 0.8) along its height. You **must** use an appropriate OpenGL transformation function with proper parameters to scale the cube.
- Use the **key "u"** to switch between the basic and upgraded Scarecrow.
- **<u>Where to write your code</u>**:
    - Write your code for both creating and transforming the body parts in `draw_Scarecrow_Upgrade()` inside the class `Scarecrow`.
    - The code for switching the display of the basic and upgraded Scarecrow is already written in the main. If your code for both drawing functions works, you should see the switch between these two versions by pressing 'u'.

In your written report, build a scene graph to represent the upgraded Scarecrow's body parts in a hierarchical tree. Assume that all the body parts initially are at the origin, with spheres and cubes centered at the origin and cylinders aligned with the z-axis with their base touching the origin. The scene graph indicates the basic shape (sphere, cylinder, etc.) and dimensions of each body part, the transformations applied to each body part locally, and their parent-child relationship. In the scene graph for the upgraded Scarecrow, there are two nodes derived from the root (doesn't need to be a body part): "*upper body*" and "*lower body*." The upper body is made of "*trunk*" and two arms – "*left arm*" and "*right arm*". Each arm contains a child's "upper arm," which has a child's "*lower arm.*" The "*head*" is the child of the trunk, and the "*nose*" is the child of the head. The lower body is made of two legs – "*left leg*" and "*right leg,*" with each leg containing the "*upper leg*" and its child "*lower leg*"; each foot is the child of the lower leg. **Draw the scene graph and associate each node with appropriate geometric transformations that occur to that node (if any) in order to transform it to the intended position/orientation in 3D**.

For example, for the node "trunk," which is a part of the group "upper body," to transform the trunk to its current position in **Figure 2**, you need to rotate it along some axis by some angle. Therefore, the transformation node applied to the "trunk" could be expressed as "$R_{axis}(angle)$," of course, using a specific $axis$ (e.g., x-axis) and $angle$ in degrees. For its child node "head", there should be another transformation node (e.g., translation) applied to the head, say, "$T(x, y, z)$", where x, y, z are the coordinates of the translation vector. Note that the cumulative transformation matrix (CTM) for the node "head" should be $R_{axis}(angle) \cdot T(x, y, z) \cdot head$. **Your OpenGL code should be consistent with the transformations obtained from your scene graph for each node.**

Please refer to the "Scene Graph" in Lecture 6 "Transformations & Scene Graph", for more details. You may also find Lecture 7 Exercise 3, "Construct a Joint Structure," relevant and helpful for this task.

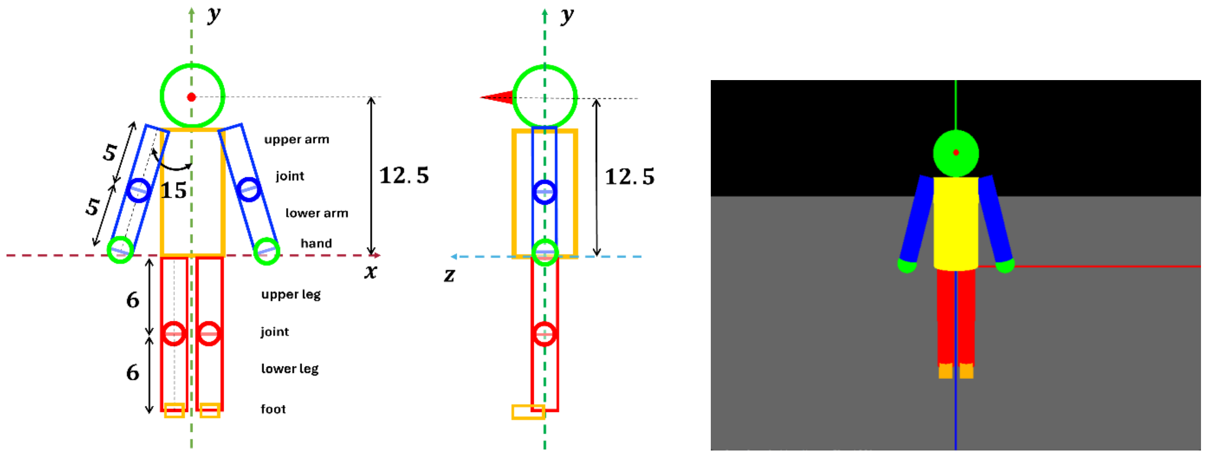

Figure 2: Scarecrow with more joints.

Task 4: "Walk in Place" Animation

To simulate walking, the Scarecrow's arms and legs should be swinging forward and backward with the arms and legs on the opposite sides moving in the same direction and speed to maintain the balance of the body during motion, see **Figure 3**. For instance, the right arm and left leg should be moving forward while the left arm and right leg are moving backward, and vice versa. Specifically,

- When moving the arm forward, the lower arm should be rotated more than the upper arm when the arm passes the torso in a forward motion. As the arm moves backward, the lower arm gradually rotates to be aligned with the upper arm, and the entire arm remains straight in the rest of the backward motion.
- Legs are moved in an opposite way as the arms. When moving the leg backward, the lower leg should be rotated more than the upper leg when the leg passes the torso in a backward motion. As the leg moves forward, once the lower leg rotates to be aligned with the upper leg, they remain straight.
- The swinging angles for upper arms, `arm_angle`, and upper legs, `leg_angle`, range [-30, 30]. Feel free to adjust the range as you see more appropriate. When the Scarecrow stands still, its initial position of the legs and arms is aligned with the middle line of the torso, with both angles 0, see **Figure 3**.

- Make sure to use `swing_speed`, i.e., the delta angle increases or decreases the angles of the arms and legs in each iteration, to update `leg_angle` and `arm_angle` so that the **swinging speed is adjustable**.
- Use the **key 'l'** to toggle the walk-in-place animation – press it once to start and press it again to pause. By default, the animation should be turned off so that **the Scarecrow stands still at its origin, just like in Figure 2**.
- **<u>Where to write your code</u>**: Write your code for transforming the arms, legs, joints, hands, and feet for the walk-in-place animation within the function `draw_Scarecrow_Upgrade()`. Your code for updating limb motion parameters (`arm_angle`,`leg_angle, etc.) based on key input should be written in a proper place in the main function.

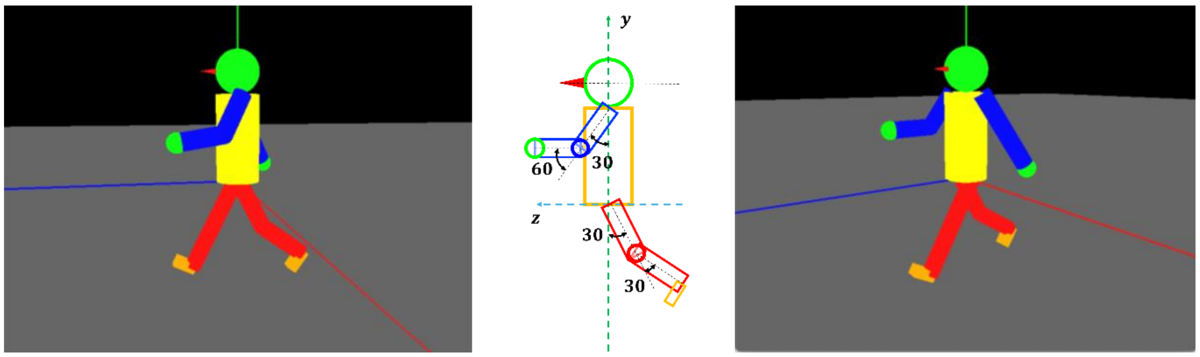

Figure 3: Transforming the Scarecrow's limbs during the walking animation. The middle figure only shows the moving angles of the left arm (forward) and left leg (backward), where the lower limbs rotate further based on the rotation of their upper limbs.

2.3 Scarecrow Freeform Walking with Keyboard Input Control

Once you feel good about the walk-in-place animation in Task 4, you may set your Scarecrow free to let it walk along a given direction, which should be adjustable by the keyboard inputs.

Task 5: Walk in a Straight Line

By default, let your Scarecrow walk along the positive direction of the z-axis, i.e., walk_direction = (0, 0, 1) at a constant speed, e.g., walk_speed = 0.5. Therefore, for each iteration, the **<u>entire</u>** Scarecrow is translated by `*walk_vector*`, where

$$walk\_vector \mathrel{+}= walk\_speed \times walk\_direction \qquad \textbf{(Eq.1)}$$

Specifically,

- You should set your `*walk_speed*` to a proper value so it looks like the Scarecrow walks at a speed that is synchronized with the speed of swinging its legs and arms in Task 4 to make the animation look more realistic.
- Use the **key 'r'** to toggle the freeform walking animation to let it walk toward the given direction at a constant speed. Press it again to pause the motion. **When 'r' is pressed, the "walk-in-place" animation from Task 4 should be automatically enabled**. By default, the walking animation should be turned off so that **the Scarecrow stands still at its origin, just like in Figure 2**.
- **Where to write your code**: Write your code for this task within the function `draw_Scarecrow_Upgrade()`. Your code for the key input detection and handling should be in the main function.

Task 6: Freeform Walk with Keyboard Control

By default, the Scarecrow faces the +z direction and walks along the direction, walk_direction = (0, 0, 1) at `walk_speed = 0.01`. To change its walking direction, use the keyboard key, ← and →, to rotate the `walk_direction` left and right around the y-axis by `walk_angle`, and then translate the entire Scarecrow with `walk_vector` in **Eq. 1**, see **Figure 4**. Moreover, the whole body of the Scarecrow should also be rotated by `walk_angle` accordingly. Specifically,

- Use the **key '←'** to update `walk_angle` (you need to figure out whether to increment or decrement it), and then respectively rotate the entire body of the Scarecrow to **its left** by that angle and translate its position by `walk_vector` calculated based on the transformed walk_direction.
- Use the **key '→'** to update `walk_angle` (you need to figure out whether to increment or decrement it), and then respectively rotate the entire body of the Scarecrow to **its right** by that angle and translate its position by `walk_vector` calculated based on the transformed `walk_direction`.
- By default, without pressing ← or →, the Scarecrow should walk along the default `walk_direction = (0, 0, 1)` (i.e., straight-line walking in Task 5) once the key 'r' has been toggled to start the walking animation.
- **Where to write your code:**
  - Complete the function `rotate_vector()`, which rotates the input vector by the input angle (in degrees) around the input rotation axis. You may assume that the rotation axis is either the x-axis, y-axis, or z-axis. You should always normalize `walk_direction` every time after the rotation to make sure it still is a unit vector (length of 1). To implement this task, establish a 3 x 3 rotation matrix based on the input rotation axis and angle and multiply it with the

input vector. **Hint**: use Numpy arrays for the matrices and vectors and Numpy's dot product for matrix multiplication.

- Complete the function `update_walk_vector()` within the Scarecrow class, which first rotates the current `walk_direction` by `walk_angle` using the function `rotate_vector()`, then update `walk_vector` based on **Eq. 1**.
- Then, in `draw_Scarecrow_Upgrade()`, rotate and translate the entire Scarecrow using `walk_angle` and `walk_vector` respectively.
- Your code for the key input detection and handling should be in the main function. When implementing the key event, ensure that you allow **holding the key to update the `walk_angle` continuously** instead of pressing the key every time to change `walk_angle`, which would be very tedious and tiresome when trying to make a turn. Once the update is updated, call the function `update_walk_vector()`.

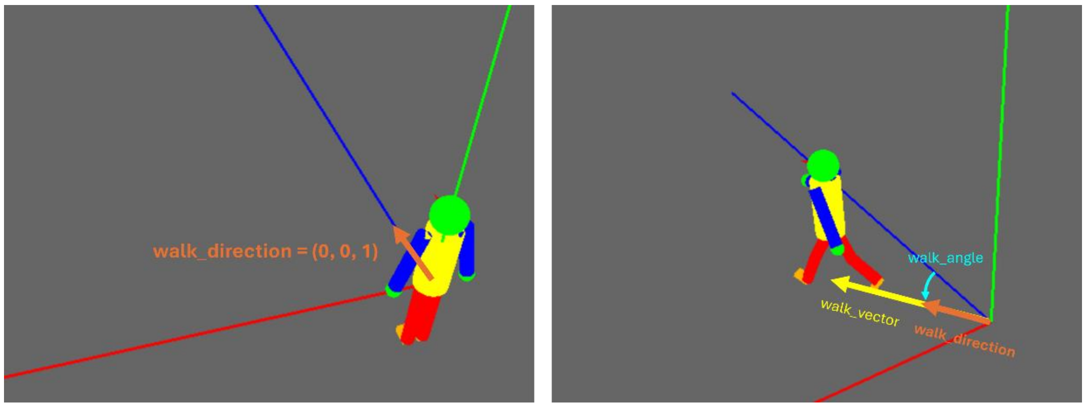

Figure 4: Freeform walking based on keyboard input.

3. Viewing Transformations

Besides animating the Scarecrow using geometric transformations, you will also implement a dynamic 3D viewing transformation, specifically developing a user-controlled camera model by adjusting **camera parameters** (i.e., eye position, look-at point) in real-time using keyboard control.

In this project, we use perspective projection the entire time, i.e., gluPerspective(). You will only need to adjust the camera parameters in gluLookAt(), to use different camera settings in different tasks.

Task 7: Switch Between Front, Side, and Back Views

Before implementing dynamic view transformation using keyboard input, you will first implement **three view modes**: **front view**, **side view**, and **back view**. The view-up vector remains (0, 1, 0) in these views. Specifically,

- **Front view**: The default view when running the program, where the camera is positioned on the positive z-axis and looks in the direction of the negative z-axis, and sees the front face of the Scarecrow, see **Figure 5: Left**. The front view has been implemented already. **You should keep it as your default view**.
- **Side view**: The camera is positioned at the positive x-axis and looks in the direction of the negative x-axis. Make sure you place the camera close enough that we can see its whole body clearly, and that it is not clipped; see **Figure 5: Middle**. This view can be used to view **Scarecrow's walk-in-place animation**.
- **Back view**: The camera is positioned in the back and looks at the Scarecrow's back in a diagonal direction. Since this view would be used for viewing the **straight-line and freeform walking animations**, ensure you place the camera in a position so that we can clearly see the walking animation for a longer period, something like **Figure 5: Right**.
- Use the **space bar** to switch between these three views. **The front view is the default view** when running the program. Press the space bar to switch to the side view, press it again to the back view, and press it one more time back to the front view. Of course, you may hold the mouse or trackpad to rotate the view during any of these views. But the space bar should always switch the view to the next pre-defined view based on the order: front view -> side view -> back view -> front view.
- <u>**Where to write your code**</u>:
    - Complete the function `switch_view()` inside the class `Camera` by writing some code to initialize camera parameters for `side` and `back`. Also, write code inside the function to support the switch between these view modes (see the description in the last bullet point above). The function is already called in the main function responding to the key input.

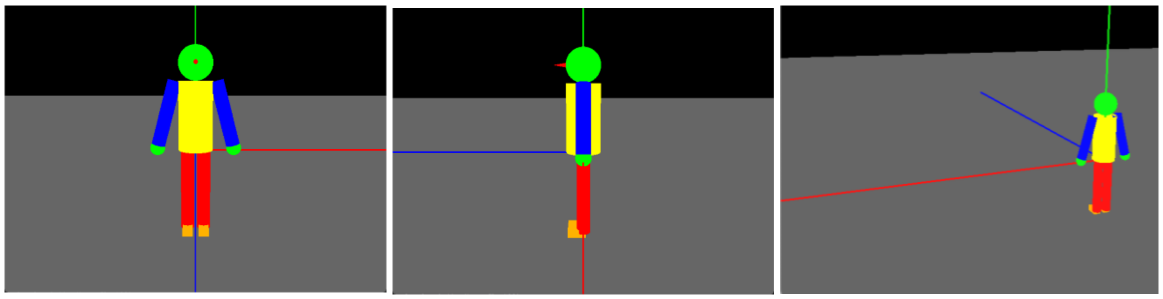

Figure 5: Front view (left), side view (middle), and back view (right).

Task 8: Dynamic Viewing with Keyboard Control

Finally, the camera position and look-at point should be updated based on key input in real-time. Specifically,

- Rotate the camera to its left or right (horizontally):
  - Hold the **key “a”** to continuously rotate the camera to look to its left.
  - Hold the **key “d”** to continuously rotate the camera to look to its right.
  - When the key “a” or “d” is pressed, increase or decrease `tilt_angle_horizontal` (you need to figure out which is which). Then, rotate the camera’s current **gaze vector** with that angle and use the updated gaze vector to calculate the updated look-at point. Use the function `rotate_vector()` for this. **The eye position remains the same.**
- Rotate the camera upward or downward (vertically):
  - Hold the **key “w”** to continuously rotate the camera to look upward.
  - Hold the **key “s”** to continuously rotate the camera to look downward.
  - When the key “w” or “s” is pressed, increase or decrease `tilt_angle_vertical` (you need to figure out which is which). Then, rotate the camera’s current **gaze vector** with that angle and use the updated gaze vector to calculate the updated look-at point. Use the function `rotate_vector()` for this. **The eye position remains the same.**
- Move the camera forward or backward along its gaze vector:
  - Hold the **key “q”** to continuously move the camera’s position forward along its gaze vector.
  - Hold the **key “e”** to continuously move the camera’s position backward along its gaze vector.
  - When the key “q” or “e” is pressed, increase or decrease `zoom_distance` (you need to figure out which is which). Then, update the camera’s position with a vector that has the direction of the current gaze vector (convert it into a unit vector first) and the length of `zoom_distance`.
- **Where to write your code**:
  - Write some code to update the camera parameters inside the function `update_view()` within the class `Scarecrow`. The function is already called inside the main function.
  - Your code for updating the viewing parameters (e.g., `title_angle_horizontal`, `zoom_distance`) based on the key input should be written in a proper place in the main function before `update_view()` is called. When implementing the key event, ensure that you allow **holding the keys to update the camera’s parameters continuously**.

4. Extra Credit

So far, we have been using a **third-person view** for the scarecrow, where the camera is located somewhere in the world space and is independent of the scarecrow's position. However, sometimes, the **first-person view** may be preferred by the user, especially in first-person shooting games. Can you transform the camera to the head of the Scarecrow so that we can view the scene from its point of view during the animation?

To calculate the camera position in real time, you will need to know the exact position of the Scarecrow's head in each frame during the walking animation. To do so, you will need to manually calculate its current head position by updating it with the `walk_vector` defined in **Eq.1**.

To calculate the camera's look-at point in real-time, you will need to know which direction the Scarecrow's head faces during the walking animation. Initially, the Scarecrow's facing direction is (0, 0, 1). During walking animation, we may rotate its entire body using keys "→" and "←" and rotate its head using the keys "i" and "o". You will need to manually rotate the facing vector by multiplying it first with a rotation matrix based on the body's rotation angle, `walk_angle` (Task 6), and then with a rotation matrix based on the head's rotation angle, `head_angle` (Task 2). The look-at point can then be calculated based on the updated facing direction. You may assume the camera's view-up vector remains the same during the walking animation.

Enrich your 3D scene – To clearly demonstrate your result for the first-person view and make it more fun, please create some 3D shapes (cubes, etc.) in the scene so that we may see changes in the views through the Scarecrow's eyes while it is walking.

<u>**Where to write your code**</u>:

- Create a new view mode named "**first_person**" inside function `update_view(),` and you can use the space bar to switch between viewing modes.
- When switching to the first-person view, the camera's parameters are adjusted according to the transformation of the Scarecrow's head; **the original camera's keyboard input (a/d/w/s/q/e) should be disabled**.

You are also welcome to implement your own 3D animation based on the Scarecrow and/or more dynamic control with the camera. Please check with the instructor to verify that it is sufficient.

Look-Up Table for Keyboard Input

Please ensure you use exactly the following keys for all eight tasks and that each key works individually and as a whole.

- **"i"** and **"o"**: Rotate Scarecrow's head to its left and right
- **"l"**: Start and pause the limb motion for a walking-in-place effect
- **"r"**: Start and pause the freeform walking based on given direction and speed; limb motion enabled automatically; walking along the z-axis with speed 0.1 by default.
- "← " and "→": Turn Scarecrow to the left and right during the freeform walking.
- **"a"** and **"d"**: Tilt the camera to look at its left and right.
- **"w"** and **"s"**: Tilt the camera to look at its up and down.
- **"q"** and **"e"**: Move the camera forward and backward along its gaze vector.
- "**u**": Switch the version of Scarecrow: "basic" and "upgraded".
- **Space bar**: Switch the camera between different views: front view (default), side view, back view, and first-person view (if any).
- **"0"**: to reset the view **(implemented already).**

Grading:

- Create a Scarecrow with a Rotatable Head (**15 pts**)
- Upgraded Scarecrow and Walk-in-Place Animation (**25 pts**)
- Freeform Walking with Keyboard Control (**20 pts**)
- Dynamic Viewing with Keyboard Control (**15 pts**)
- Software engineering, efficiency, & stability (**5 pts**)
- Written report: clarity, formatting, & comprehensiveness (**20 pts**)
- Extra credit: First-Person Dynamic View and/or your own choice (**up to 20 pts each**)

Project 2 Submission Guideline

- (80%) TWO Python files, "P2_transformView_model.py" and "P2_transformView_main.py," contain your complete code for all the required tasks in this document, including extra credit if any; make sure your code is up and running. Modify the window title to include your name. Search for "CPSC515: Transformation & Viewing - YOUR NAME" in the main file.
- (20%) Write-up (**PDF**) **and** **1 x Demo video (MP4)**:
  - **Task 1: Create a Basic Scarecrow**

    - **Method**: Briefly explain how you transform (rotation/translation/scale, by what parameters) each body part into the appropriate positions to construct the Scarecrow, in what order.
    - **Result**: Part of the result of Task 7.
- **Task 2: Rotate the Scarecrow's Head & Nose with Keyboard Input**
    - **Method**: Briefly explain how you only rotate the head and nose, what OpenGL functions you use, how you incorporate them into the previous code for Task 1, and how you incorporate keyboard input into the transformation.
    - **Result**: **Screen-recording video** that clearly shows the Scarecrow's head and nose are rotated and controlled by you (you don't need to record your key pressing action, just the outcome).
- **Task 3: Upgrade the Scarecrow with More Joints**
    - **Method**: First, include the scene graph (by hand or drawing tool) of the upgraded Scarecrow. **Please refer to Task 3 for detailed instructions** on the scene graph. Second, briefly describe how you transform each body part into the upgraded Scarecrow using OpenGL and how your code is related to the scene graph.
    - **Result**: Part of the result for Task 7.
- **Task 4: Walk-in-Place Animation**
    - **Method**: 1) Briefly explain how you transform (translate/rotate/scale, by what parameters) an arm (upper and lower) and leg (upper and lower) during the forward and backward movement, respectively; 2) How and when to make the lower limbs transform more than the upper limbs; 3) What angle ranges are used for the arms and legs (upper and lower); 4) How and when to keep the arms and the legs straight; 5) How to transform a joint, for both arm and leg; 6) How to transform hands; 7) How to create **and** transform feet, which initially are cubes; 8) How you adjust the swinging speed of the arms and legs?
    - **Result**: **Screen-recording video** that **clearly** shows the Scarecrow's walk-in-place animation **from different angles**. In the video, you should demonstrate both the start and pause of the animation by pressing a designated key (you don't need to record your key pressing action, just the outcome).
- **Task 5: Walk in a Straight Line**
    - **Method**: Briefly explain 1) How to transform the entire Scarecrow along the given direction and speed; 2) How you implement it on top

        of the previous code for Task 4; and 3) How you synchronize the walking speed with the arm swinging speed to make the walking look realistic?
    - **Result**: **Screen-recording video** that **clearly** shows the Scarecrow's straight-line walking animation. In the video, you should demonstrate both the start and pause of the animation by pressing a designated key (you don't need to record your key pressing action, just the outcome).
- **Task 6: Freeform Walk with Keyboard Control**
    - **Method**: Briefly explain 1) How you rotate `walk_direction` based on the rotation angle, `walk_angle`, mathematically, what your formula looks like, what rotation matrix and parameters you used; 2) How you calculate and update the `walk_vector` and how you use it to update the position of the Scarecrow, mathematically; 3) Explain your OpenGL implementation for freeform walking including both the transformation of the Scarecrow and the keyboard input to support continuous parameter update.
    - **Result**: **Screen-recording video** that **clearly** shows the Scarecrow's freeform walking animation. In the video, you should first show Scarecrow walking in a straight line for a bit and then turning to the left, walking for a while, and then turning to the right walking for a while, and repeat.
- **Task 7: Switch Between Front, Side, and Back Views**
    - **Method**: Briefly explain how you set the camera parameters for the side view and back view and why.
    - **Result**: **Screen-recording video** that clearly shows the switch between front, side, and back views in real-time. To demonstrate the robustness of your view-switching method, use the mouse to rotate the view a bit and then switch to the next view by pressing the designated key (you don't need to record your key pressing action, just the outcome).
- **Tasks 8**: **Dynamic Viewing with Keyboard Control**
    - **Method**: Briefly explain 1) how you rotate the camera's gaze vector by an angle, mathematically, say what formulas you use, how you construct the matrix, how you calculate the new look-at point based on that; 2) how you update the camera's position to make it move forward and backward along the gaze vector based on a distance, mathematically, say what formula you use; 3) Explain your OpenGL

        implementation for both camera tilting and zooming and the keyboard input to support continuous camera motion.
        - **Result**: **Screen-recording video** that **clearly** shows the camera control for 1) tilt left, 2) tilt right, 3) tilt up, 4) tilt down, 5) zoom-in: move forward, and 6) zoom-out move backward. **Ensure the Scarecrow is in view the entire time to best showcase your result**.
    - **Frist-Person Dynamic View (Extra Credit)**:
        - **Method**: Clearly explain your algorithm and OpenGL implementation: how you calculate the head position and facing direction mathematically and update these parameters to update the camera model during Scarecrow's dynamic walking (walking and turning around and looking around).
        - **Result**: **Screen-recording video** that **clearly** shows the first-person view during Scarecrow's dynamic walking (walking and turning around and looking around); enrich your scene to create more 3D shapes to help demonstrate the first-person view.
    - **Formatting**
        - The report should be well-organized and clearly written in a structured format.
        - Use headings, bullets, and concise explanations where appropriate.
        - Ensure the screenshots are clearly labeled.
        - **Submit your demo video as ONE MP4 file**: You may either shoot one video to demonstrate all the tasks in one run or combine all the video clips for individual tasks into one. You **don't need to narrate** your video, but please keep it **brief and effective**. **In the video, please clearly annotate which task is being demonstrated to avoid confusion.** Both MacOS (Screenshot) and Windows (Snipping Tool) have their own built-in screen recording and video editing apps. If your default video format is not MP4, please find online tools to convert your format into MP4 for submission.
        - **In the video, you must show the entire window of the pygame program (not your desktop), with your name on the window title**.

Using AI/Internet Resources in Assignments: AI tools and online resources can support your learning and provide inspiration, but your final work must demonstrate your own understanding and implementation. Refer to the course syllabus for specific guidelines on AI use in assignments. Be sure to properly cite and acknowledge any AI or internet sources you use, explaining where and how they contributed to your work in your written report.