COSC363 Computer Graphics
Lab05: Vector Math

Aim:

This lab demostrates the application of vector operations in transforming and rendering objects.

I. FlightPath.cpp:

The program displays a simple scene containing a rocket model and a polygonal path (flight path) defined in three-dimensional space (Fig. 1a). Pressing key '2' switches the view to a close-up view of the model (Fig. 1b). The model is positioned at the origin and is oriented along the positive *x*-axis. Pressing key '1' switches the view back to the original room view. Use left and right arrow keys to rotate the scene to get a clear view of the shape of the flight path. The flight path contains 70 vertices whose coordinates are stored in the file `FlightPath.txt`.

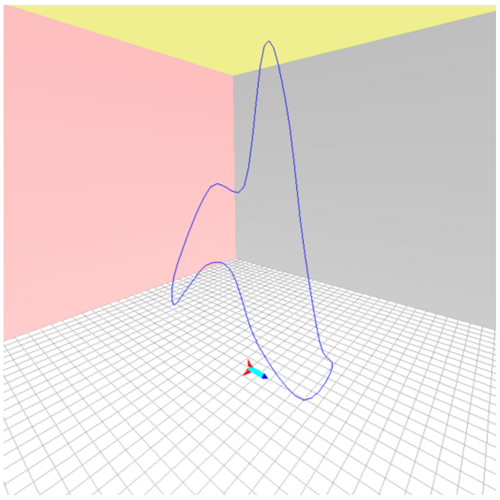
Fig. 1a

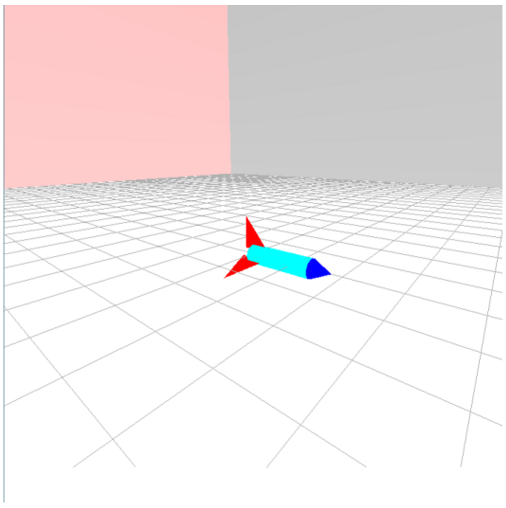
Fig. 1b

The objective of this lab exercise is to define transformations for the rocket model so that it moves along the flight path, continuously changing its orientation based on the local tangent direction at the current position.

Define a timer function that increments a global integer variable `indx` which we will use to index into the array of points on the flight path. This index should vary from 0 to NPTS-1, and get reset to 0 when it reaches the value NPTS. (NPTS is the total number of points on the flight path).

Let P = (ptx[indx], pty[indx], ptz[indx]) be the current point on the path, and Q the next point (refer to Slide [6]-26). The vector $\boldsymbol{v} = Q\text{-}P$ denotes the direction at the current position. Compute this vector inside the `display()` function and normalize it to a unit vector. The initial direction of the rocket model (Fig. 2a) is $\boldsymbol{u} = (1, 0, 0)$. Using the dot-product of the two vectors, compute the turn angle $\theta = \cos^{-1}(\boldsymbol{u}.\boldsymbol{v})$. Convert this angle from radians to degrees.

The axis of rotation $\boldsymbol{w}$ (Fig. 2a) is obtained as the vector cross-product $\boldsymbol{w} = \boldsymbol{u} \times \boldsymbol{v}$. The cross-product of two vectors (u_x, u_y, u_z) and (v_x, v_y, v_z) is given by the vector $(u_y v_z - u_z v_y,\ u_z v_x - u_x v_z,\ u_x v_y - u_y v_x)$. Apply the following transformations to the rocket model in the given order:

(a) First, rotate the model by an angle θ about the axis $\boldsymbol{w}$.

(b) Then, translate the model to the current point P on the path.

The above sequence of operations, if implemented correctly, should generate the display of the motion of the model along the flight path (Fig. 2b).

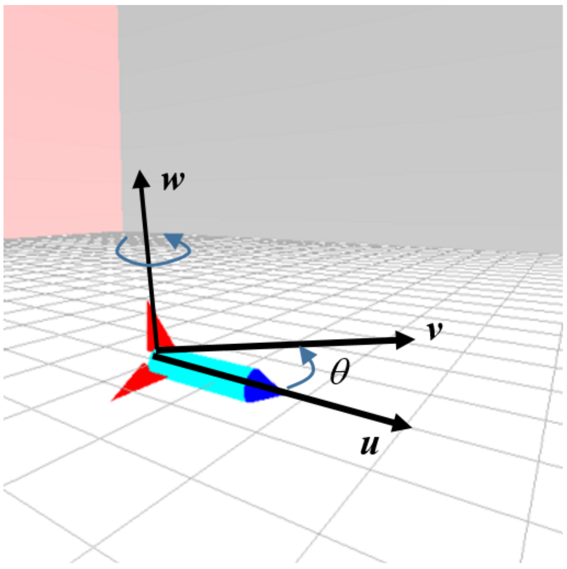

Fig. 2a

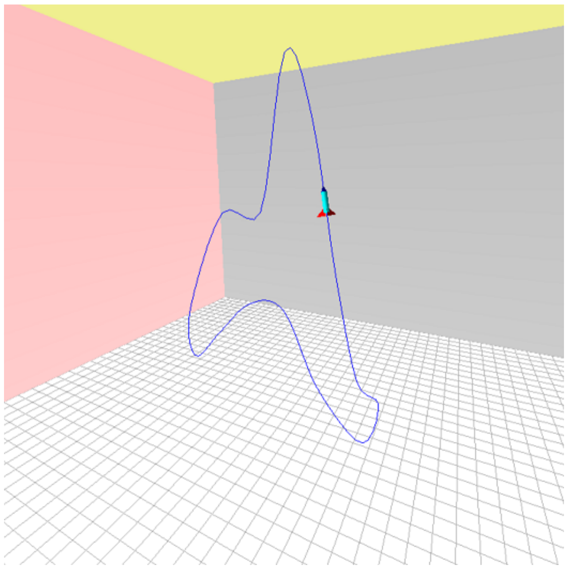
Fig. 2b

II. Shadow.cpp:

The program displays a floor plane and a rotating teapot. The floor plane contains a small rectangular section to which a glass texture is mapped (Fig. 6(a)). The light is positioned towards the right side of the scene, with coordinates (80, 80, 0).

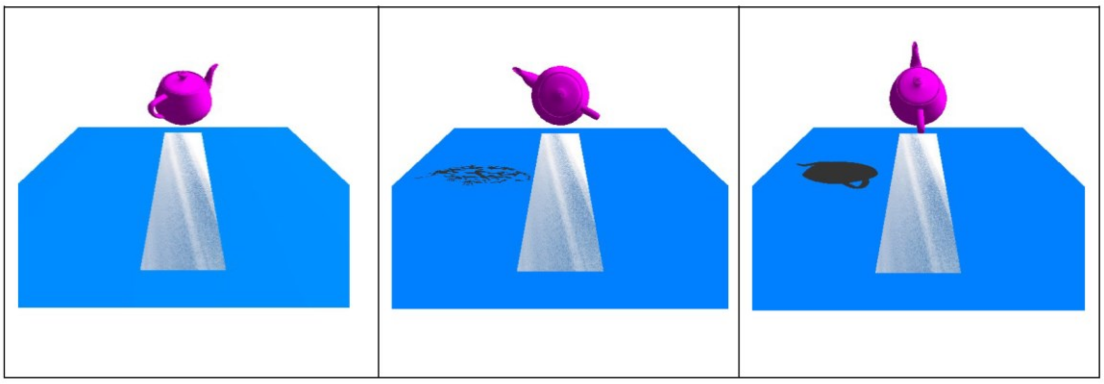
Fig. 6(a) Fig. 6(b) Fig. 6(c)

1. Inside the `display()` function, define a shadow matrix as given on slides [6]-32, 33. The elements of the shadow matrix are light's coordinates *lx* (=80), *ly* (=80), *lz* (=0). As shown on the slide, the generation of planar shadows is a two pass rendering process. First, we draw the teapot with all its transformations, and with lighting enabled. We then draw the teapot again (along with all its transformations), but this time we also multiply the transformation matrix by the shadow matrix. The shadow matrix "projects" the transformed teapot on to the floor plane. Remember to change the colour of the teapot's shadow to a dark gray value, and disable lighting so that the shadow polygons have a uniform colour. Enable lighting again after drawing the shadow. The shadow is rendered on the floor plane (with y-value 0) and therefore results in an artefact called depth fighting (or *z*-fighting) because of equal depth values for fragments belonging to the floor and the shadow (Fig. 6(b)). Move the floor plane slightly downward, by setting the value of the variable `floor_height` (in function floor()) to -0.1 The shadow of the teapot should now get rendered correctly (Fig. 6(c)).

2. We will now render the reflection of the teapot on the glass plate. Draw the teapot (along with all its transformations) for the third time, but this time also multiplying by the scale transformation glScalef(1, -1, 1) which inverts the teapot along the *y*-axis. Note: The teapots must be drawn first, and finally the floor plane. The reflection of the teapot will not be visible as it is occluded by the floor plane.

3. Set the alpha value (opacity) of glass (in function floor()) to a small value, say 0.3. This is the fourth parameter in `glColor4f(…)` called before drawing the glass plate. Note that blending is already enabled in the program. Please make sure that the texture environment parameter is set to GL_MODULATE. The output of the program is shown in Fig. 7(a).

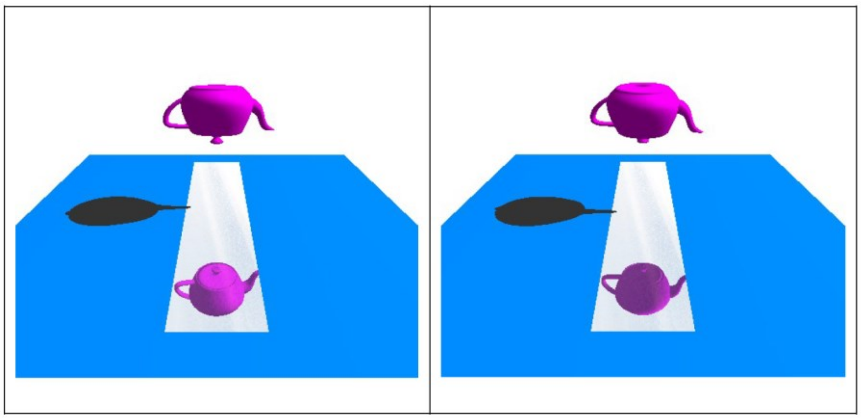

Fig. 7(a) Fig. 7(b)

4. The reflection in Fig. 7(a) is not rendered correctly. The lid of the teapot is in shadow, but the lid of its reflection is illuminated by the light source. In fact, the whole reflected object is illuminated from the light source's position. We need to invert the light source's *y*-value before drawing the reflection:

```
// Invert light's position before drawing reflection
light[1] = -light[1];
glLightfv(GL_LIGHT0, GL_POSITION, light); //new position
```

// Draw Reflection

5. Change the light's position back to its original value after drawing the reflected teapot (and before drawing the floor). The final output is in Fig. 7(b).

Ref:

[6]: COSC363 lecture slides, "6-Mathematical Preliminaries".

III. Quiz-05

The quiz will remain open until 11:55pm, **28-Apr-2023**.

A quiz can be attempted only once. A question within a quiz may be attempted multiple times. However, a fraction of the marks (25%) will be deducted for each incorrect answer.