COSC363 Computer Graphics
Lab06: 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`.

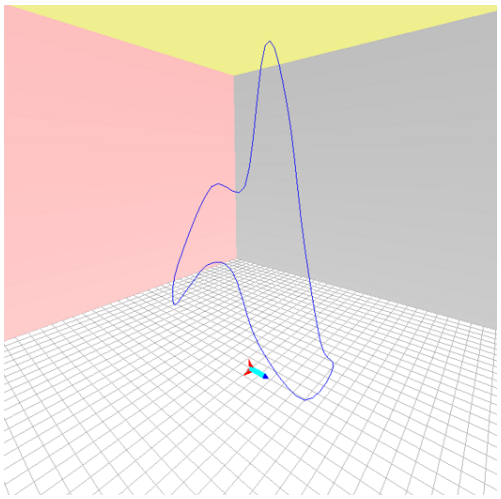

Fig. 1a

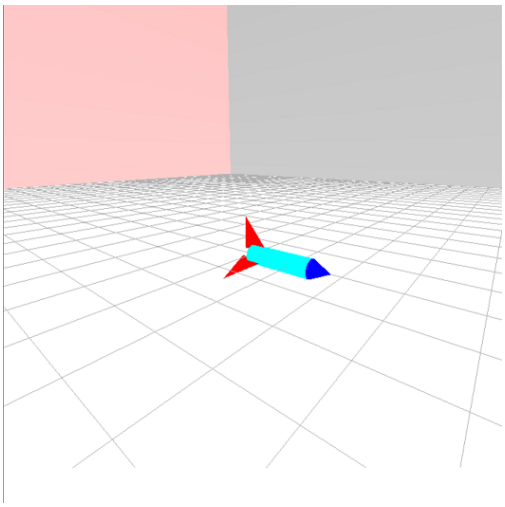

Fig. 1b

The objective of this lab exercise is to define the 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. The vector $\boldsymbol{v} = Q-P$ denotes the target direction at the current position. Compute this vector inside the `display()` function and normalize it (Slide [6]-3). The initial direction of the rocket model (Fig. 2a) is always $\boldsymbol{u}$ = (1, 0, 0). Using the dot-product of the two vectors (Slide [6]-4), 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 $(u_yv_z - u_zv_y, u_zv_x - u_xv_z, u_xv_y - u_yv_x)$. 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).

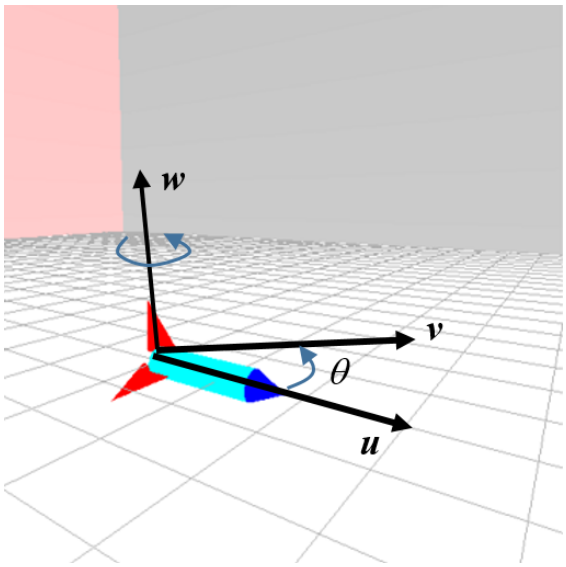

Fig. 2a

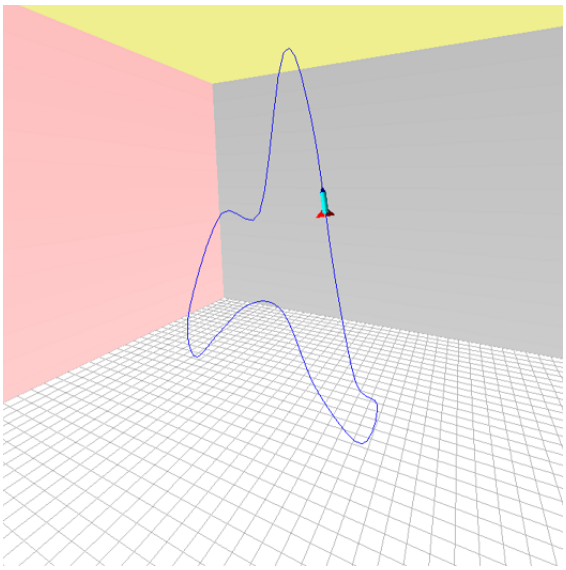

Fig. 2b

II. Model3Dvn.cpp:

`Model3Dvn.cpp` is a slightly modified version of `Model3D.cpp` used in Lab01 for loading and displaying 3D model files. In `Model3D.cpp`, the normal vectors were computed when each triangle was drawn. `Model3Dvn.cpp` on the other hand pre-computes the normal vectors in the initialization stage and stores the components in arrays (fnx[], fny[], fnz[]). Please go through the code in the function `normal()` to understand how the normal vectors are calculated. The normal vector (fnx[indx], fny[indx], fnz[indx]) represents the face normal of a triangle, and is assigned as the common normal vector for the whole triangle in the `display()` function (Fig. 3a). The program generates the display shown in Fig. 4a. The computation of lighting using face normals gives a nearly uniform illumination on each face, and shows discontinuity in the variation of shades along polygonal edges.

Vertex normals represent the average normal direction at a vertex, and are computed by gathering the face normals of all faces that share that vertex. A vertex normal vector is assgined to each vertex in the definition of an object (Fig. 3b). Per-vertex normals give a more accurate representation of local orientation of a surface, and thus

a smooth shading of the surface without revealing the underlying polygonal structure (Fig. 4b).

```
glBegin(GL_TRIANGLES);
   glNormal3f(...);
   glVertex3f(...);      }
   glVertex3f(...);      } Face 0
   glVertex3f(...);      }
   glNormal3f(...);
   glVertex3f(...);      }
   glVertex3f(...);      } Face 1
   glVertex3f(...);      }
     ...
glEnd();
```

Fig. 3a. Object definition using face normals

```
glBegin(GL_TRIANGLES);
   glNormal3f(...);
   glVertex3f(...);
   glNormal3f(...);
   glVertex3f(...);
   glNormal3f(...);
   glVertex3f(...);
   glNormal3f(...);
   glVertex3f(...);
     ...
glEnd();
```

Fig. 3b. Object definition using vertex normals

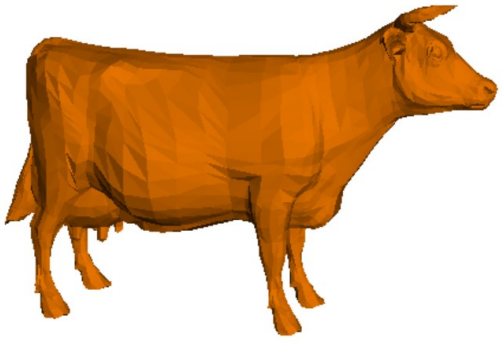

Fig. 4a. An object rendered using face normals

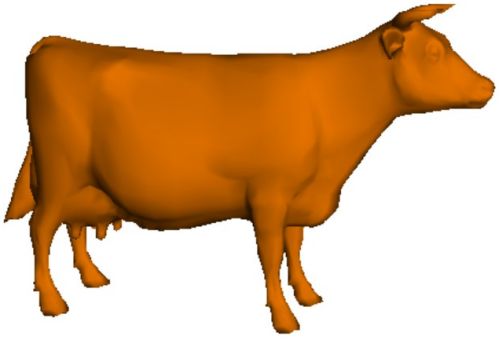

Fig. 4b. An object rendered using vertex normals

The program includes the declarations of arrays (vnx[], vny[], vnz[]) for storing vertex normal components. These arrays act as accumulators of face normals. When a face normal vector is computed, it is added to the corresponding elements of the vertex normal arrays for all three vertices of that face (Fig. 5). Thus, for each vertex we get the sum of face normals of all triangles that share that vertex.

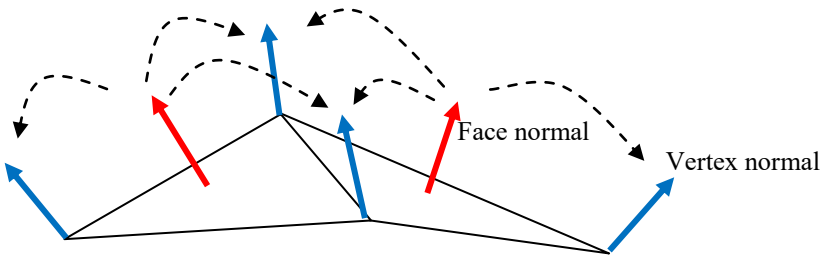

Fig. 5.

Modify the `normal()` function to add the face normal components nx, ny, nz to elements of arrays (vnx[], vny[], vnz[]) with indices t1[indx], t2[indx], t3[indx]. Also modify the glBegin..glEnd block in the `display()` function by specifying vertex normal vectors at each vertex as shown in Fig. 3b. The program should produce a display similar to that shown in Fig. 4b.

Ref:

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

III. GLM:

`GLM` is a convenient mathematics library written in C++ language. It is designed specifically for graphics application development using OpenGL. We will be using GLM extensively in the implementation of methods for ray tracing and OpenGL-4 shader applications. This lab provides a quick introduction to GLM using the program `GLM_Examples.cpp` which contains examples of types, operations and functions of the GLM library and their usefulness in performing vector and matrix computations. A set of powerpoint slides on GLM functions, GLM.pdf , is also provided. Please go through the files and gain a general understanding of how the library is commonly used for specifying vector and matrix operations.

IV. Quiz-06

The quiz will remain open until **5pm, 7-May-2021**.

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.