Assignment 02: Modelling, Viewing and Projection

Documentation

Tanishq Jain, 2021294

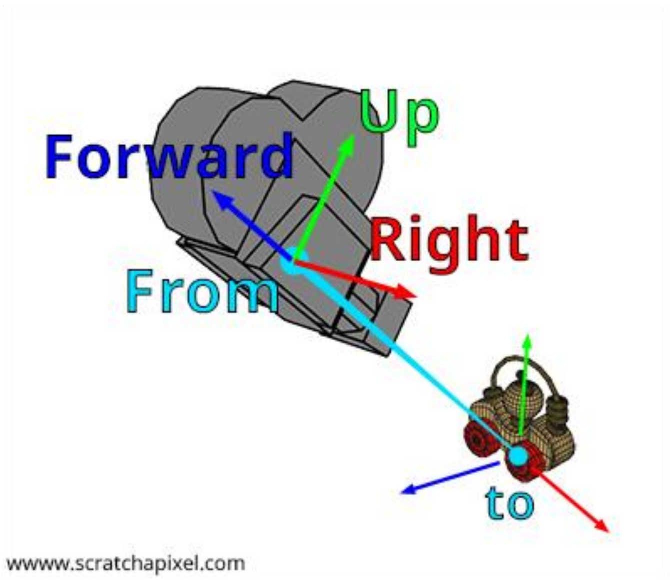
Question 1 and 2

We are given a scaled cube (with scale factor = 2 in Y direction) with bottom face centered at world space origin, a camera at initial *camPosition* = (20, 40, 80) rendering the cube.

1(a)

To program the arrow keys (and shift) to change the camera view of the scene by changing camera center co-ordinates in camera space. This results in a camera motion on a sphere around the origin.

We are given world space origin (0, 0, 0) as the look-at point and (0, 1, 0) as the *world_up* vector. The camera space can be defined using unit vectors along the 3 camera axes as the basis vectors. We call these vectors as *cam_front*, *cam_right* and *cam_up*. Since the look-up point is fixed at origin, at any instant cam_front vector is the vector from camera position to origin. Hence, *cam_front* = -1 * *camPosition*. To get the *cam_right* vector, we can take a cross product of the *cam_front* and *world_up* vectors. Hence, *cam_right* = cross(*cam_front*, *world_up*). Finally, we can get the cam_up vector by taking a cross product of the *cam_front* and *cam_right* vectors. Hence, *cam_up* = cross(*cam_front*, *cam_right*). Whenever we detect an arrow key press, we add one of these vectors to *camPosition* in order to move in the required direction. *cam_right*, *cam_up* and *cam_front* represent the X, Y and Z camera axes respectively. These are clearly illustrated by the following figure.



Implementation

CAM_SPEED – Variable denoting speed of the camera

camPosition – Vector denoting position of the camera

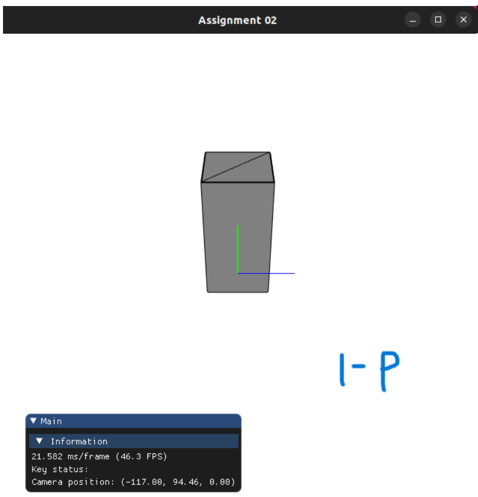
cam_front, cam_right, cam_up – Unit vectors along the 3 camera axes at any instant

glm::cross() – GLM function for taking cross product of two vectors

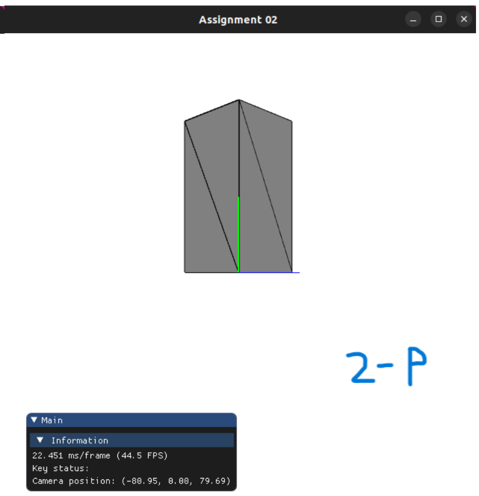
glm::normalize() – GLM function for converting a vector into a unit vector

1(b)

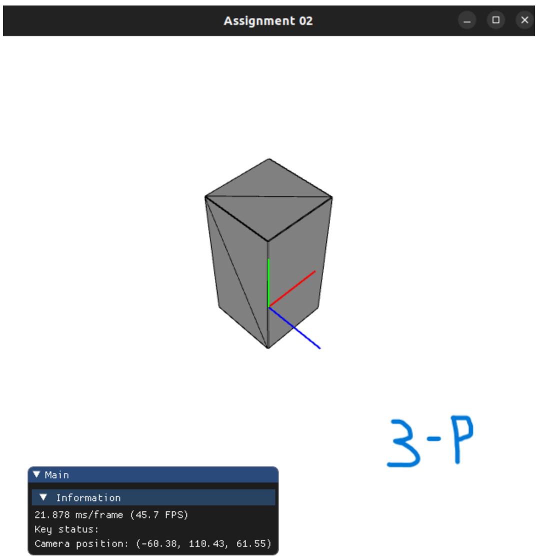
Move the camera to generate one-point perspective, two-point perspective and three-point perspective views.



1-P



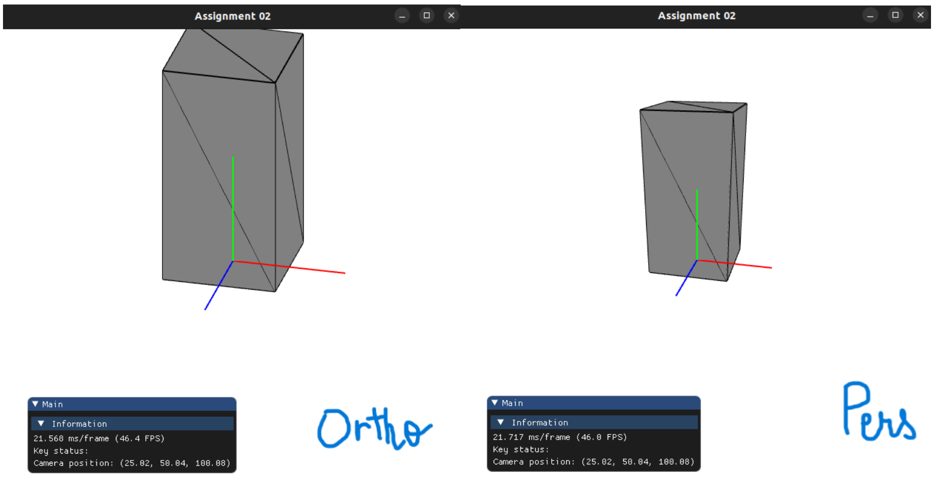
2-P



3-P

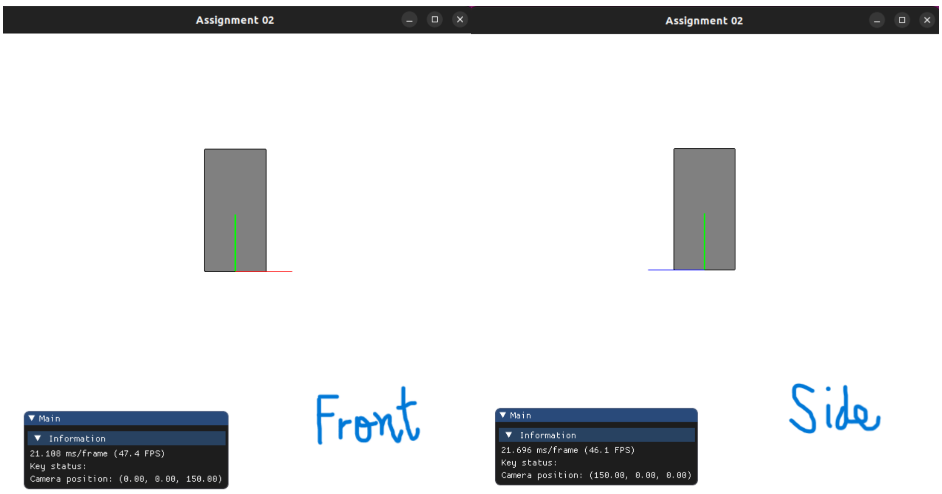
2(a)

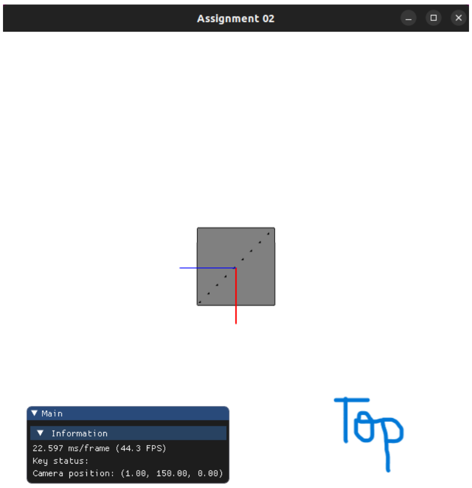
Switch between perspective (press X) and orthographic (press Z) projections. Perspective projection has already been implemented. For orthogonal projection, we use glm::ortho() function (instead of glm::perspective()) to get the projection transformation matrix projectionT inside the setupProjectionTransformation() function.



2(b)

Using arrow keys + modifier key (ctrl), move the camera to specific positions to generate top view, front elevation and side elevation. We need to snap the camera center to appropriate axes upon such input. We do this by setting camPosition to a position on the world X/Y/Z axis corresponding to the arrow key pressed.





Question 3

Find the inverse of the rigid body transformation A (where R is a 3x3 rotation matrix and t is a 3-vector).

$$A = \begin{bmatrix} R & t \\ 0\ 0\ 0 & 1 \end{bmatrix}$$

We can split A into a rotation and a translation matrix. Here, I is 3x3 identity matrix and z is 3x1 zero vector.

$$A = \begin{bmatrix} I & t \\ 0\ 0\ 0 & 1 \end{bmatrix} \begin{bmatrix} R & z \\ 0\ 0\ 0 & 1 \end{bmatrix}$$

Now, A^{-1} is the inverse of the product of these matrices. Hence, it is the product of their inverses in reverse order. Inverse of a translation matrix is a translation matrix with opposite signs in t. Inverse of a rotation matrix is its transpose.

$$A^{-1} = \begin{bmatrix} R & z \\ 0\ 0\ 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} I & t \\ 0\ 0\ 0 & 1 \end{bmatrix}^{-1}$$

$$= \begin{bmatrix} R^T & z \\ 0\ 0\ 0 & 1 \end{bmatrix} \begin{bmatrix} I & -t \\ 0\ 0\ 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} R^T & -R^T t \\ 0\ 0\ 0 & 1 \end{bmatrix}$$