TINGE

Week 2

Programming Assignment

4 February 2025

§1 3D and 4D Vectors

You need to create 2 classes: One for a 3D Vector and another for a 4D Vector.

- Vectors contain n floats, a_i, which signifies that the vector represents $a_1e_1 + a_2e_2 + \cdots + a_ne_n$, where n is the dimension of the vector and $\{e_i\}$ is a linearly independent set of vectors (similar to coordinate axes). Eg: a 3D vector $v = \{1, 2, 3\}$ represents the vector as $v = 1e_x + 2e_y + 3e_z$.
- Create constructors and destructors for this class, preferably both a default constructor and a constructor that takes input.
- Define important functions:
  - Dot product.
  - Cross product.
  - Length: Norm of the vector ($||x||$).
  - Normalize: Make the length = 1 without changing the direction.
- Look up Operator Overloading if you do not already know it. This allows simple arithmetic operators to be used conveniently on class objects.
- Set up operator overloading for these operators:
  - $+, -, *, /$
  - $==$ (returns true if vectors are identical, false otherwise)

Note: Name these classes something other than `std::vector`. Since they should be different for 3D and 4D vectors, you could use names like `Vector3D` and `Vector4D`.

Here's how the structure of the class is supposed to look like:

```
class Vector3D {
public:
    float x, y, z;
    Vector3D();  // Default constructor
    Vector3D(float x, float y, float z);  // Parameterized constructor
    ~Vector3D();  // Destructor
    float dot_product(Vector3D &v2);  // Class Functions
    Vector3D operator+(Vector3D &v2);  // Operator overloading
};
```

§2 3×3 and 4×4 Matrices

You need to create 2 classes: One for a 3×3 Matrix and another for a 4×4 Matrix.

- Store the matrix elements in arrays or `std::vector`.
- Create constructors and destructors.
- Define some important functions:
  - Transpose: Should return the transpose of the matrix, and not modify the elements of the existing matrix.
  - Inverse: Compute and return the inverse of the matrix, ensuring it exists.
  - Scale: Takes in n floats: s_i, and returns a Matrix with these s_i as diagonal entries, with remaining elements being 0. Here is an example for a 3×3 Matrix:

$$\begin{bmatrix} s_1 & 0 & 0 \\ 0 & s_2 & 0 \\ 0 & 0 & s_3 \end{bmatrix}$$

- Additionally, for a 3×3 Matrix, implement these extra functions:
  - Rotation Matrix: Takes the yaw, pitch, and roll angles (α, β, γ) and returns the rotation matrix as such:

$$R = \begin{bmatrix} \cos\alpha\cos\beta & \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma & \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma \\ \sin\alpha\cos\beta & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma \\ -\sin\beta & \cos\beta\sin\gamma & \cos\beta\cos\gamma \end{bmatrix}$$

    For more information on rotation matrices, see Wikipedia or just google up what a Rotation Matrix is. Also learn about what the Euler Angles are.
  - Translate: Take in a position (p_x, p_y, p_z) and return a 4×4 position matrix as such:

$$\begin{bmatrix} p_x & 0 & 0 & 0 \\ 0 & p_y & 0 & 0 \\ 0 & 0 & p_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

    While you're at it, think about why this might be useful to us? Or about how this isnt a 3×3 Matrix.
- Implement operator overloading for these classes with the following operators: $+$, $-$, $*$, $==$
- **Bonus:** Implement the / operator to allow multiplication of the first matrix with the inverse of the second matrix.
- You should also be able to multiply a matrix with a vector:
  - If a matrix is of order $n \times n$ and a vector is of order $n \times 1$, the result should be another vector of order $n \times 1$. *Think about how and why this can be used.*
  - Implement this operation using operator overloading between two different classes. Refer to operator overloading concepts if needed.

If you have any query about this assignment or anything about these topics in general, feel free to text us. Also let us know when you are done implementing this, you will get some brownie points for doing so 😎.

Good luck on your code! Here's KK signing off~