KORAY ASLAN

28180

Report on the Cube Object Transformation and Animation Project

Link to ChatGPT Chat TASK 1

<https://chat.openai.com/share/c62cbc36-5a99-4101-98b4-0afa4693e085>

Link to ChatGPT Chat TASK 3

<https://chat.openai.com/share/81dd9cce-6121-4d9d-ab76-a95bffab9754>

**Introduction**

This project involved using ChatGPT to calculate and apply a ModelView matrix to a cube object, followed by creating an animation method for the object. The project was divided into three main tasks:

1. Task 1: Calculate the ModelView matrix using ChatGPT.

2. Task 2: Independently generate the same transformation matrix.

3. Task 3: Animate the cube object using the calculated transformation.

**Task 1: ChatGPT ModelView Matrix Calculation**

In Task 1, ChatGPT was used to compute the ModelView matrix for the cube object. The matrix provided by ChatGPT was as follows:

const transformationMatrix = new Float32Array([

0.4330126941204071, -0.2165063523054123, 0, 0.15,

0.25, 0.25, 0, -0.125,

0, 0, 0.25, 0,

0, 0, 0, 1

]);

This matrix was then transposed and used in the `getChatGPTModelViewMatrix()` function. The matrix represents a combination of translation, scaling, and rotation transformations applied to the cube.

**Task 2: Independent ModelView Matrix Calculation**

For Task 2, the ModelView matrix was calculated independently using provided transformation methods (`createScaleMatrix`, `createRotationMatrix\_X/Y/Z`, `createTranslationMatrix`) and parameters from `transformation-prompt.txt`. The resultant matrix differed from the one provided by ChatGPT. This discrepancy can be attributed to the different approaches and order of matrix operations (rotation, scaling, translation) used in each method. Matrix multiplication is not commutative, meaning the order of these transformations significantly affects the final result.

**Task 3: Cube Object Animation**

Task 3 involved animating the cube object using the transformation calculated in Task 2. The animation was designed to last 10 seconds, with the cube transitioning to the calculated transformation in the first 5 seconds and returning to its initial position in the last 5 seconds. The `getPeriodicMovement()` function was implemented for this purpose, using linear interpolation between the initial and target transformation matrices.

**Conclusion**

This project demonstrated the practical application of matrix transformations and animations in a 3D environment. The differences observed in the ModelView matrices between Tasks 1 and 2 highlight the importance of the sequence of transformations in 3D rendering. The final animation showcased a seamless transition of the cube between two states, illustrating the capabilities of matrix interpolation in creating smooth animations.