BBM414 Computer Graphics Lab.
Programming Assignment 4 # - 2024

Ahmet Uman
2210356129
Department of Computer Engineering
Hacettepe University
Ankara, Turkey
b2210356129@cs.hacettepe.edu.tr

Overview

The purpose of this assignment is to get familiar with get familiar with 3 dimensional shapes with first person camera viewing. We will load three objects and create a controllable camera on 3D scene. Transformations (scale, rotate, translate), perspective camera, dynamic window resizing and mouse input features are integrated in the project. This study provides an opportunity to understand basic graphic transformations regarding WebGL usage.

1 Part 1

In this project, we created a pyramid shape using a WebGL scene. Various methods were used to change the shape's appearance and behavior with user inputs. The features developed include controlling the camera, providing rotation with the pointer lock mechanism, and assigning different colors to the pyramid surfaces.

To achieve this, transformation operations were performed on the vertices of the pyramid in the vertex shader. This transformation operation was calculated using the perspective projection matrix and the model-view matrix (Model-View Matrix).

```
attribute vec4 vPosition;
attribute vec4 vColor;
varying vec4 fColor;

uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;

void main() {
    gl_Position = projectionMatrix * modelViewMatrix * vPosition;
    fColor = vColor;
}
```

The vertices of the pyramid were defined using a vector array. The pyramid has four base points and a vertex. The base of the pyramid was defined as a quadrilateral and its side surfaces were created using triangles. Different colors were assigned to each surface. The camera movement was controlled with the radius, theta and phi variables. These variables were updated using mouse movement and the camera angle was changed. The user can change the scene parameters using the sliders. After all the calculations were completed, the scene was drawn over and over again.

2 Part 2

2.1 Setting Up the Scene

The project starts with a Monkey Head model created in Blender. This model is loaded into the scene in OBJ format. The model is parsed and positioned using WebGL, and each model is customized with separate transformation matrices. Initial state in the scene:

Left Monkey: Positioned at (-2, 0, 0) on the left. Middle Monkey: Positioned at (0, 0, 0) in the middle. Right Monkey: Positioned at (2, 0, 0) on the right.

Positioning Matrix: Translation matrix is used and each model is initialized to a fixed position with the help of this matrix.

```
const leftMonkey = createTranslationMatrix(-2, 0, 0);

const centerMonkey = createTranslationMatrix(0, 0, 0);

const rightMonkey = createTranslationMatrix(2, 0, 0);
```

Shaders are used to ensure that each model is drawn correctly on the screen and that transformations are applied at the GPU level.

Vertex Shader:

```
attribute vec4 aPosition;
attribute vec3 aNormal;
uniform mat4 uModelMatrix;
uniform mat4 uViewMatrix;
uniform mat4 uProjectionMatrix;
varying vec3 vNormal;

void main() {
    gl_Position = uProjectionMatrix * uViewMatrix * uModelMatrix * aPosition;
    vNormal = aNormal;
}
```

This shader is responsible for applying each model first to the model matrix (animation), then to the view matrix and finally to the projection (perspective) matrix.

```
precision mediump float;
varying vec3 vNormal;
uniform vec3 uLightDirection;

void main() {
    float light = max(dot(normalize(vNormal), normalize(uLightDirection)), 0.0);
    gl_FragColor = vec4(vec3(1.0, 0.5, 0.2) * light, 1.0);
}
```

2.2 Animation and Transformations

Each monkey in the scene has a unique animation to show the different transformations. The animations are managed using transformation matrices that are dynamically recalculated every frame.

1.Leftmost Monkey (Scaling): This monkey is constantly scaled up and down, creating a zoom effect. The scaling factor is controlled by a sinusoidal function to provide smooth transitions between dimensions. A translation matrix places the monkey at the leftmost position in the scene. A scaling matrix changes its size, where the scaling factor, determined by the sine of the elapsed time, oscillates between 0.7 and 1.3. The final model matrix is a product of the translation and scaling matrices.

```
const scaleFactor = 1 + Math.sin(deltaTime * 0.0005) * 0.3;
const leftMatrix = multiplyMatrices(
createTranslationMatrix(-2, 0, 0),
createScalingMatrix(scaleFactor, scaleFactor, scaleFactor)
);
```

This animation visually emphasizes smooth dynamic scaling without abruptly resizing the model.

2. Medium Monkey (Rotation): The medium monkey rotates around the Y-axis. The rotation angle increases linearly over time, creating a uniform and continuous rotation. A translation matrix positions the monkey at the center of the scene. A rotation matrix changes its orientation around the Y-axis. The rotation angle is updated incrementally based on the elapsed time. The final model matrix is the product of the translation and rotation matrices.

```
rotationAngle += deltaTime * 0.0002;
const middleMatrix = multiplyMatrices(
createTranslationMatrix(0, 0, 0),
createRotationYMatrix(rotationAngle)
);
```

This animation shows a consistent rotation that remains smooth even over long periods of time.

3. Rightmost Monkey (Translation): This monkey moves up and down the Y-axis, simulating a jumping motion. When the motion reaches certain limits, it changes direction, creating a visually different animation. A translation matrix sets the monkey's Y coordinate, which is updated linearly over time. When the Y coordinate crosses predefined limits, the direction of the motion is reversed (+0.5 and -0.5). The final model matrix is a simple translation matrix.

```
verticalOffset += upDirection * deltaTime * 0.0003;
if (verticalOffset > 0.5 || verticalOffset < -0.5) {
upDirection *= -1;
}
const rightMatrix = createTranslationMatrix(2, verticalOffset, 0);
```

This animation introduces dynamic vertical movement that adds variety to the overall composition of the scene.

2.3 Controllable Camera

The camera allows users to dynamically explore the scene via mouse inputs. This interaction enhances the user experience by providing full control over the viewing perspective. The camera's movements are implemented using special matrix operations.

Basic Features:

1. Left Click (Rotate Camera):

Rotates the camera around the center of the scene. Uses spherical coordinates to calculate the camera's position relative to the center. Prevents rotation by compressing the vertical angle to prevent gimbal lock.

The horizontal angle and vertical angle are updated based on mouse movement. These angles are converted to Cartesian coordinates to calculate the camera's new position.

```
theta -= dx * turningSpeed;
phi -= dy * turningSpeed;

phi = Math.max(epsilon, Math.min(Math.PI - epsilon, phi)); // Clamp phi
```

```
camera.eye = [
camera.center[0] + radius * Math.sin(phi) * Math.cos(theta),
camera.center[1] + radius * Math.cos(phi),
camera.center[2] + radius * Math.sin(phi) * Math.sin(theta),
];
```

This implementation provides smooth and intuitive camera rotation, preventing unnatural behaviors such as panning.

2. Middle Click (Zoom):

Moves the camera closer or further away from the scene by adjusting the radius of the global coordinates.

A normalized direction vector points from the camera to the center. The camera's position is adjusted along this direction based on mouse movement.

```
const direction = normalizeVector(subtractVectors(camera.center, camera.eye));
camera.eye = [
camera.eye[0] + direction[0] * zoom,
camera.eye[1] + direction[1] * zoom,
camera.eye[2] + direction[2] * zoom,
];
```

This feature provides precise zoom control that allows users to focus on specific parts of the scene.

3. Right Click (Pan):

Moves the camera laterally in the X and Y directions according to its orientation.

The camera's right and up vectors are used to calculate the pan offsets. These offsets are added to both the camera's position and target (center).

```
const right = normalizeVector(crossProduct(camera.up,
    subtractVectors(camera.center, camera.eye)));
const up = camera.up;

camera.eye = [
camera.eye[0] + right[0] * panX + up[0] * panY,
camera.eye[1] + right[1] * panX + up[1] * panY,
camera.eye[2] + right[2] * panX + up[2] * panY,
];

camera.center = [
camera.center[0] + right[0] * panX + up[0] * panY,
camera.center[1] + right[1] * panX + up[1] * panY,
camera.center[2] + right[2] * panX + up[2] * panY,
];
```

This feature complements panning and zooming by providing precise side adjustments, allowing for full exploration of the scene.

2.4 Dynamic Adaptation

The camera's projection matrix is dynamically recalculated to maintain the proper aspect ratio and prevent distortion when resizing the canvas. The updated aspect ratio ensures consistent rendering of monkey models.

```
window.addEventListener("resize", () => {
gl.canvas.width = window.innerWidth;
gl.canvas.height = window.innerHeight;
gl.viewport(0, 0, gl.canvas.width, gl.canvas.height);

camera.aspect = gl.canvas.width / gl.canvas.height;
const newProjectionMatrix = createPerspectiveMatrix(camera.fovy, camera.aspect,
    camera.zNear, camera.zFar);
gl.uniformMatrix4fv(uProjectionMatrix, false, newProjectionMatrix);
});
```