Frontend UIUX Documentation

This document describes the frontend components of a 3D graphics application built with HTML5 Canvas and JavaScript. The frontend handles user interaction, including camera controls, object manipulation, geometry input, and initialization via a welcome screen. It interfaces with the 3D engine by updating variables that influence rendering.

Overview

The frontend consists of an HTML structure with a canvas for rendering and various controls (sliders, buttons, text areas) for user input. JavaScript code manages these elements, updates their displays, and passes user inputs to the rendering logic. Key features include:

- **Camera Controls**: Sliders for view angle and pitch angle.
- **Object Controls**: Sliders for translation (X, Y, Z), scaling (X, Y, Z), and rotation speed.
- **Geometry Input**: Text areas for editing vertices, indices, and colors, with an update button.
- **Reset Functionality**: A button to restore default settings.
- **Welcome Screen**: An initial overlay with a start button to launch the application.

Components

1. Canvas Setup

The rendering surface is an HTML5 Canvas element, accessed and configured as follows:

```
const canvas = document.getElementById("renderer");
const ctx = canvas.getContext("2d");
const canvasWidth = canvas.width;
const canvasHeight = canvas.height;
const framebuffer = ctx.createImageData(canvasWidth, canvasHeight);
```

- **Purpose**: Provides the drawing surface for the 3D engine.
- **Usage**: The `framebuffer` is manipulated by the engine and displayed via `ctx.putImageData()`.

2. Camera Controls

Two sliders adjust the camera's orientation:

- **View Angle Slider** (`#viewAngle`):
  - Controls the yaw (horizontal rotation) of the camera.
  - Event listener updates `viewAngle` and refreshes the display:

  ```
  viewAngleSlider.addEventListener("input", (e) => {
      viewAngle = parseFloat(e.target.value);
      updateDisplays();
  });
  ```

- **Pitch Angle Slider** (`#pitchAngle`):
  - Controls the pitch (vertical tilt) of the camera, initialized at 10°.
  - Similar event listener updates `pitchAngle`.
- **Display**: Values are shown in `#viewAngleValue` and `#pitchAngleValue` (e.g., "0°", "10°").

3. Object Controls

Sliders manipulate the 3D object's position, scale, and rotation speed:

- **Translation Sliders** (`#translateX`, `#translateY`, `#translateZ`):
  - Adjust the object's position in 3D space.
  - Example for X-axis:

  ```
  translateXSlider.addEventListener("input", (e) => {
      translateX = parseFloat(e.target.value);
      updateDisplays();
  });
  ```

- Values displayed in `#translateXValue`, etc., rounded to one decimal (e.g., "0.0").
- **Scale Sliders** (`#scaleX`, `#scaleY`, `#scaleZ`):
  - Adjust the object's size along each axis, defaulting to 1.
  - Similar event listeners update `scaleX`, `scaleY`, `scaleZ`.
- **Speed Slider** (`#speed`):
  - Controls the automatic rotation speed around the Y-axis, defaulting to 0.1.
  - Listener:

    ```
    speedSlider.addEventListener("input", (e) => {
        speed = parseFloat(e.target.value);
        updateDisplays();
    });
    ```

  - Displayed in `#speedValue` (e.g., "0.10").

4. Geometry Input

Users can define custom geometry via text areas:

- **Text Areas**:
  - `#vertexInput`: Lists vertices (e.g., `[0, 1, 0]`).
  - `#indexInput`: Defines triangle faces (e.g., `[0, 1, 2]`).
  - `#colorInput`: Sets vertex colors (e.g., `[255, 0, 0, 255]`).
- **Update Geometry Button** (`#updateGeometryButton`):
  - Parses inputs and validates them:

    ```
    updateGeometryButton.addEventListener("click", () => {
        try {
            const newVertices = parseVertices(vertexInput.value);
            const newIndices = parseIndices(indexInput.value);
            const newColors = parseColors(colorInput.value);
            // Validation checks...
            vertices = newVertices;
            indices = newIndices;
            colors = newColors;
            alert("Shape updated!");
        } catch (error) {
            alert("Error: " + error.message);
        }
    });
    ```

  - Ensures vertex and color counts match, indices are valid, etc.
- **Parsing Functions**:
  - `parseVertices()`, `parseIndices()`, `parseColors()`: Convert text to arrays, throwing errors for invalid formats.

5. Reset Button

The `#resetButton` restores defaults:

```
resetButton.addEventListener("click", () => {
    viewAngle = 0;
    pitchAngle = 10;
    translateX = translateY = translateZ = 0;
    speed = 0.1;
    currentRotateY = 0;
    scaleX = scaleY = scaleZ = 1;
    vertices = defaultVertices.map(v => [...v]);
    indices = defaultIndices.map(i => [...i]);
    colors = defaultColors.map(c => [...c]);
    updateDisplays();
    fillTextAreas();
});
```

- Resets all sliders, rotation, and geometry to initial pyramid shape.

6. Welcome Screen

An overlay (`#welcome-screen`) with a start button (`#start-engine-btn`):

```
startEngineBtn.addEventListener("click", () => {
    welcomeScreen.style.display = 'none';
    mainApp.style.display = 'flex';
    document.body.style.overflow = 'auto';
    updateDisplays();
    fillTextAreas();
    if (!animationFrameId) {
        lastTimestamp = performance.now();
        animationFrameId = requestAnimationFrame(render);
    }
});
```

- Hides the welcome screen, shows the main app, and starts the render loop.

7. Utility Functions

- **updateDisplays()**: Updates all slider value displays:

    ```
    function updateDisplays() {
        viewAngleValue.textContent = viewAngle + '°';
        // ... other updates ...
        speedValue.textContent = speed.toFixed(2);
    }
    ```

- **fillTextAreas()**: Populates text areas with current geometry data.

Interaction Flow

1. **Start**: User clicks the start button, revealing controls and starting the render loop.
2. **Adjust**: Sliders update camera and object properties in real-time.
3. **Customize**: Users edit geometry in text areas and click "Update Geometry" to apply changes.
4. **Reset**: The reset button reverts to defaults if needed.

This frontend provides an intuitive interface for interacting with the 3D engine, passing user inputs to affect rendering dynamically.