UNIT IV
OpenGL 3D viewing functions

In OpenGL, 3D viewing typically involves setting up the camera, perspective, and viewing transformations. There are several key functions and techniques you can use to implement 3D viewing. Below are some of the core functions related to 3D viewing in OpenGL:

1. Projection Matrix Setup

The projection matrix is responsible for setting up the perspective of the 3D world. There are two common types of projections in 3D rendering:

- **Perspective projection**: Objects appear smaller as they get further from the camera.
- **Orthographic projection**: Objects remain the same size regardless of distance from the camera.

Perspective Projection:

You typically use `glFrustum` or `gluPerspective` (if using legacy OpenGL) to set up perspective projections:

```
// Legacy OpenGL (fixed pipeline)
gluPerspective(fovy, aspect, zNear, zFar);
```

- `fovy`: Field of view in the y direction (in degrees).
- `aspect`: Aspect ratio (width/height) of the viewport.
- `zNear`: Distance to the near clipping plane.
- `zFar`: Distance to the far clipping plane.

Orthographic Projection:

Use `glOrtho` or `gluOrtho2D` for setting up orthographic projection:

```
// Legacy OpenGL (fixed pipeline)
glOrtho(left, right, bottom, top, zNear, zFar);
```

- `left`, `right`, `bottom`, `top`: Define the coordinates of the orthographic projection box.
- `zNear` and `zFar`: Define the range of depth for the near and far clipping planes.

2. Camera/View Matrix Setup

The view matrix is responsible for positioning and orienting the camera in the 3D world. The camera can be considered as an observer looking at a scene from a specific position, and OpenGL provides several ways to transform the camera view.

LookAt Function (Using `gluLookAt` or `glm::lookAt`):

The `gluLookAt` function (legacy OpenGL) or `glm::lookAt` (modern OpenGL, using GLM math library) is used to set the camera position and direction.

```
// Legacy OpenGL (fixed pipeline)
gluLookAt(eyeX, eyeY, eyeZ, centerX, centerY, centerZ, upX, upY, upZ);
```

- `eyeX, eyeY, eyeZ`: Position of the camera (eye).
- `centerX, centerY, centerZ`: Point in the scene that the camera is looking at (target).
- `upX, upY, upZ`: Defines the up direction of the camera, usually (0, 1, 0) for a vertical up axis.

Modern OpenGL with GLM:

In modern OpenGL, you would use the GLM library for math operations, and you can use `glm::lookAt`:

```
glm::mat4 view = glm::lookAt(glm::vec3(eyeX, eyeY, eyeZ), glm::vec3(centerX, 
centerY, centerZ), glm::vec3(upX, upY, upZ));
```

3. Model-View Matrix Setup

The model-view matrix combines the transformations of the model and the camera (view). It's often necessary to adjust an object’s position, scale, and rotation, relative to the camera’s view.

You can use `glTranslatef`, `glRotatef`, and `glScalef` (legacy OpenGL) to manipulate the model-view matrix:

```
glPushMatrix();
glTranslatef(tx, ty, tz);  // Translation
glRotatef(angle, rx, ry, rz);  // Rotation
glScalef(sx, sy, sz);  // Scaling
// Draw object
glPopMatrix();
```

- `tx, ty, tz`: Translation along x, y, and z axes.
- `angle`: Angle in degrees to rotate.
- `rx, ry, rz`: Rotation axis.
- `sx, sy, sz`: Scale factors along x, y, and z.

In modern OpenGL, transformations are done using shaders and matrices, typically by multiplying model, view, and projection matrices.

4. Viewport Setup

To map the 3D scene to the 2D screen, you need to specify the size and position of the viewport. This is done using `glViewport`:

```
// Set the viewport to cover the entire window
glViewport(0, 0, windowWidth, windowHeight);
```

Example of a Complete Setup:

```
// Set the perspective projection
```

```
    gluPerspective(45.0f, (float)windowWidth / (float)windowHeight, 0.1f, 100.0f);

    // Set the camera position and orientation
    gluLookAt(0.0f, 0.0f, 5.0f, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f);
```

5. Model-View and Projection Matrix Multiplication

Modern OpenGL typically requires manually multiplying the model, view, and projection matrices in shaders. You would typically load these matrices into the vertex shader as uniforms. You use libraries like GLM to handle the matrix math.

```
    glm::mat4 projection = glm::perspective(glm::radians(45.0f), aspectRatio, 0.1f,
100.0f);
    glm::mat4 view = glm::lookAt(cameraPosition, cameraTarget, upVector);
    glm::mat4 model = glm::mat4(1.0f); // Identity matrix for model transformation

    // Pass matrices to shader (using OpenGL's uniform system)
    GLuint projectionLoc = glGetUniformLocation(shaderProgram, "projection");
    GLuint viewLoc = glGetUniformLocation(shaderProgram, "view");
    GLuint modelLoc = glGetUniformLocation(shaderProgram, "model");

    glUniformMatrix4fv(projectionLoc, 1, GL_FALSE, glm::value_ptr(projection));
    glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
    glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
```

6. Handling Movement/Camera Controls

To make the camera interactive, you would typically update the camera's position based on user input (e.g., keyboard and mouse).

- **Mouse Look**: Adjust the camera's orientation based on mouse movement.
- **Keyboard Movement**: Move the camera along the x, y, and z axes based on user input.

Summary

To summarize, in OpenGL 3D, the key functions and steps are:

1. **Set up the projection matrix** (`glFrustum`, `gluPerspective`, `glOrtho`).
2. **Position and orient the camera** using `gluLookAt` or `glm::lookAt`.
3. **Apply transformations to models** (translation, rotation, scaling) using `glTranslatef`, `glRotatef`, `glScalef`, or matrix transformations in modern OpenGL.
4. **Set up the viewport** using `glViewport`.
5. **Use the proper matrix math** for model, view, and projection combinations, especially in modern OpenGL.