Deliverable 2: Core Scene Development and Hierarchy

The scene currently consists of the reef material as the terrain and a sine-waved water overlay with two yellow fishes swimming using a sine function. For flora, I have chosen a pair of coral plants, one of them currently animated.

**Demonstration Video Link:** https://youtu.be/YVgzuxqkhbs

**Hierarchy:**

1. **Fish**: The yellow fishes are controlled by a single hierarchy. The bigger fish is the parent, and the smaller one is the child. Their motions are synchronized to the same sine wave, but due to the size and location difference, they appear distinct.

```
//read shader properties into the program
glUseProgram(shaderProgramID_fish);
int matrix_location = glGetUniformLocation(shaderProgramID_fish,
"fish_model");
int view_mat_location = glGetUniformLocation(shaderProgramID_fish,
"fish_view");
int proj_mat_location = glGetUniformLocation(shaderProgramID_fish,
"fish_proj");
//initialize display matrices
mat4 fish_view = identity_mat4();
mat4 persp_proj_fish = perspective(45.0f, (float)width / (float)height,
0.1f, 1000.0f);
mat4 fish_model = identity_mat4();
//control initial appearance
fish_model = rotate_z_deg(fish_model, 150.0f);
fish_model = rotate_y_deg(fish_model, -90.0f);
//control motion on keyboard/mouse input
fish_model = translate(fish_model, vec3(move_x, move_y, -40.0f));
fish_model = rotate_x_deg(fish_model, rotateX);
fish_model = rotate_y_deg(fish_model, rotateY);
//bind object location matrices and render
glUniformMatrix4fv(proj_mat_location, 1, GL_FALSE, persp_proj_fish.m);
glUniformMatrix4fv(view_mat_location, 1, GL_FALSE, fish_view.m);
glUniformMatrix4fv(matrix_location, 1, GL_FALSE, fish_model.m);
glDrawArrays(GL_TRIANGLES, 0, fish_mesh_data.mPointCount);
// Set up the child matrix
mat4 fishModelChild = identity_mat4();
fishModelChild = rotate_z_deg(fishModelChild, 180);
fishModelChild = rotate_y_deg(fishModelChild, rotate_y);
fishModelChild = translate(fishModelChild, vec3(0.0f, 5.9f, 0.0f));
//ensure child fish is visible distinctly by translating along y and z
axes
fish_model = translate(fish_model, vec3(0.0f, -move_y, -40.0f));
//Apply the root matrix to the child matrix
fishModelChild = fish_model * fishModelChild;
//Update the appropriate uniform and draw the mesh again
glUniformMatrix4fv(matrix_location, 1, GL_FALSE, fishModelChild.m);
glDrawArrays(GL_TRIANGLES, 0, fish_mesh_data.mPointCount);
```

```
//control the fish sine-wave movement
void updateScene() {
      //calculate regular time difference
```

```
	static DWORD last_time = 0;
	DWORD curr_time = timeGetTime();
	if (last_time == 0)
		last_time = curr_time;
	float delta = (curr_time - last_time) * 0.001f;
	last_time = curr_time;
	//control fish model translation along y-axis with a sine wave
	move_y -= delta;
	move_y = sin(move_y*10+delta) * 1.26f;
	//control linear fish model translation along x-axis
	move_x -= 2.0f * delta;
	//ensure fish stays in view-frame
	move_x = fmodf(move_x, 50);
	//re-draw the scene
	glutPostRedisplay();
}
```

2. **Green plant**: Hierarchy is used here to generate random swaying of the plant underwater. Fifty separate instances are drawn in the same vicinity with minor variance, in order to achieve this effect.

```
//read shader properties into the program
glUseProgram(shaderProgramID_plant);
int matrix_location = glGetUniformLocation(shaderProgramID_plant,
"plant_model");
int view_mat_location = glGetUniformLocation(shaderProgramID_plant,
"plant_view");
int proj_mat_location = glGetUniformLocation(shaderProgramID_plant,
"plant_proj");
//initialize display matrices
mat4 plant_view = identity_mat4();
mat4 persp_proj_plant = perspective(45.0f, (float)width / (float)height,
0.1f, 1000.0f);
mat4 plant_model = identity_mat4();
//control initial appearance
plant_model = rotate_z_deg(plant_model, 150.0f);
plant_model = rotate_y_deg(plant_model, 90.0f);
plant_model = translate(plant_model, vec3(0.0f, 60.0f, -500.0f));
//control motion on keyboard/mouse input
plant_model = rotate_x_deg(plant_model, rotateX);
plant_model = rotate_y_deg(plant_model, rotateY);
plant_view = translate(plant_view, vec3(base_x, base_y, base_z));
plant_model = rotate_x_deg(plant_model, rotateX);
plant_model = rotate_y_deg(plant_model, rotateY);
//bind object location matrices and render
glUniformMatrix4fv(proj_mat_location, 1, GL_FALSE, persp_proj_plant.m);
glUniformMatrix4fv(view_mat_location, 1, GL_FALSE, plant_view.m);
glUniformMatrix4fv(matrix_location, 1, GL_FALSE, plant_model.m);
glDrawArrays(GL_TRIANGLES, 0, plant_mesh_data.mPointCount);
//define number of child instances
int num_instances = 50;
//define position range from negative to positive on each axis
float positionRangeX = 100.0f;
float positionRangeZ = 100.0f;
float positionRangeY = 10.0f;
//run a loop for the child instances
for (int i = 0; i < num_instances; i++) {
  //Generate random positions within the specified range
  float randX = (static_cast<float>(std::rand()) / RAND_MAX) *
positionRangeX - (positionRangeX / 2);
  float randY = (static_cast<float>(std::rand()) / RAND_MAX) *
positionRangeY - (positionRangeY / 2);
  float randZ = (static_cast<float>(std::rand()) / RAND_MAX) *
positionRangeZ - (positionRangeZ / 2);
  //Set up the child matrix for this instance
  mat4 plantModelChild = identity_mat4();
  //Ensure child plant instance is visible distinctly
  plantModelChild = rotate_z_deg(plantModelChild, 150.0f);
  plantModelChild = rotate_y_deg(plantModelChild, 90.0f);
  //Apply pre-generated translation to give animated effect
  plantModelChild = translate(plantModelChild, vec3(randX, randY,
randZ));
  //Apply the root matrix to the child matrix
  plantModelChild = plant_model * plantModelChild;
  //Update the appropriate uniform and draw the mesh again
  glUniformMatrix4fv(matrix_location, 1, GL_FALSE, plantModelChild.m);
  glDrawArrays(GL_TRIANGLES, 0, plant_mesh_data.mPointCount);
}
```

**Camera Control:**

1. **Keyboard control:** I have used the four arrow keys for translating the camera viewport. The motion is mapped to move the same as the left, right, up and down keys.

```
//feed the translate variables, changing direction based the keyboard
input
void specialKeypress(int key, int x, int y) {
    switch (key) {
        case GLUT_KEY_RIGHT:
            base_x -= 2.0f;
            move_x -= 2.0f;
            break;
        case GLUT_KEY_LEFT:
            base_x += 2.0f;
            move_x += 2.0f;
            break;
         case GLUT_KEY_UP:
            base_y += 2.0f;
            move_y += 2.0f;
            break;
         case GLUT_KEY_DOWN:
            base_y -= 2.0f;
            move_y -= 2.0f;
            break;
    }
    glutPostRedisplay();
}
.
.
.
//indicate to the system to track the keyboard input
glutSpecialFunc(specialKeypress);
```

2. **Trackpad/Mouse Control:** We can also drag around the scene within the viewport using our mouse/trackpad as well.

```
//track the current mouse screen position
void mousePassive(int x, int y) {
      mouseX = x;
      mouseY = y;
}
//use the current mouse position to compare with last mouse position and
use it to feed the translate variables with a 2.0 sensitivity
void mouseMotion(int x, int y) {
       const float SPEED = 2.0f;
       base_x += fmodf((mouseX - x) / SPEED,2);
       base_y += fmodf((mouseY - y) / SPEED,2);
       move_x += fmodf((mouseX - x) / SPEED,2);
       move_y += fmodf((mouseY - y) / SPEED,2);
       rotateX += fmodf((mouseX - x) / SPEED,2);
       rotateY += fmodf((mouseY - y) / SPEED,2);
       mousePassive(x, y);
       glutPostRedisplay();
}
.
```

```
.
.
//indicate to the system to track the mouse/trackpad movement
glutMotionFunc(mouseMotion);
glutPassiveMotionFunc(mousePassive);
```