Jonathan Chavarria
CS 480 PA5
December 4, 2022
Dr. Tavakkoli

1. **Details of changes made to classes**
    a. Texture Setup
    Used the Texture files provided from CS480-PA5.zip

    b. Vertex Setup
    Used the Vertex setup in graphic_headers.h provided from CS480-PA5.zip

    c. Buffer Setup
    Used the Bugger setup in Sphere and Mesh provided from CS480-PA5.zip

    d. Model Matrix Setup

```
void Sphere::setupVertices() {
    std::vector<int> ind = getIndices();
    std::vector<glm::vec3> vert = getVertices();
    std::vector<glm::vec2> tex = getTexCoords();
    std::vector<glm::vec3> norm = getNormals();


    int numIndices = getNumIndices();

    for (int i = 0; i < numIndices; i++) {
        Vertices.push_back(Vertex(vert[ind[i]], norm[ind[i]], tex[ind[i]]));
        Indices.push_back(i);
    }
}
```

Added Vertex(vert[ind[i]], norm[ind[i]], tex[ind[i]) to the Vertices.pushback in the Sphere setupVertices class.

```
for (int j = 0; j < iMeshFaces; j++) {
    const aiFace& face = mesh->mFaces[j];
    for (int k = 0; k < 3; k++) {
        // update here for each mesh's vertices to assign position, normal, and texture coordinates
        Vertices.push_back(Vertex(glm::vec3(mesh->mVertices[face.mIndices[k]].x, mesh->mVertices[face.mIndices[k]].y, mesh->mVertices[face.mIndices
            mesh->HasNormals() ? glm::vec3(mesh->mNormals[face.mIndices[k]].x, mesh->mNormals[face.mIndices[k]].y, mesh->mNormals[face.mIndices[k]]
            glm::vec3(1.f, 1.f, 1.f),
            mesh->HasTextureCoords(0) ? glm::vec2(mesh->mTextureCoords[0][face.mIndices[k]].x, mesh->mTextureCoords[0][face.mIndices[k]].y) :
            glm::vec2(1.f, 1.f)));

    }
```

Added the appropriate Vertices.push_back for texture coordinates in the LoadModelFromFile function for Mesh.

e. Model Matrix Update
Used the Update functions provided provided from CS480-PA5.zip

f. Rendering

```
glBindVertexArray(vao);
// Enable vertex attibute arrays for each vertex attrib
glEnableVertexAttribArray(posAttribLoc);
glEnableVertexAttribArray(colAttribLoc);
glEnableVertexAttribArray(tcAttribLoc);

// Bind your VBO
glBindBuffer(GL_ARRAY_BUFFER, VB);

// Set vertex attribute pointers to the load correct data. Update here to load the correct attributes.
glVertexAttribPointer(posAttribLoc, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), 0);
glVertexAttribPointer(colAttribLoc, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof(Vertex, normal));
glVertexAttribPointer(tcAttribLoc, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof(Vertex, texcoord));


// If has texture, set up texture unit(s): update here for texture rendering
// If has texture, set up texture unit(s) Update here to activate and assign texture unit
if (m_texture != NULL) {
    glUniform1i(hasTextureLoc, true);
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, m_texture->getTextureID());
}
else
    glUniform1i(hasTextureLoc, false);



// Bind your Element Array
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, IB);
// Render
glDrawElements(GL_TRIANGLES, Indices.size(), GL_UNSIGNED_INT, 0);

// Disable vertex arrays
glDisableVertexAttribArray(posAttribLoc);
glDisableVertexAttribArray(colAttribLoc);
glDisableVertexAttribArray(tcAttribLoc);

glBindBuffer(GL_ARRAY_BUFFER, 0);
```

Made the same changes for the Render functions of both Sphere and Mesh. Enabled the three Vertex Attributes, set them appropriately, binding the Texture if there was one, then rendered the object.

2. Solar system details

a. Model Stacks and Rendering
Used code provided.

b. Updates

Used the previously provided PA4 code as a base, then made changes.

```
std::vector<float> speed, dist, rotSpeed, scale;
glm::vec3 rotVector;
glm::mat4 localTransform;
// position of the sun
modelStack.push(glm::translate(glm::mat4(1.f), glm::vec3(0, 0, 0)));  // sun's coordinate
localTransform = modelStack.top();      // The sun origin
localTransform *= glm::rotate(glm::mat4(1.0f), (float)dt * .5f, glm::vec3(0.f, 1.f, 0.f));
localTransform *= glm::scale(glm::vec3(1.25, 1.25, 1.25));
if (m_sphere != NULL)
    m_sphere->Update(localTransform);

// position of the first planet
speed = { .75, .75, .75 };
dist = { 7. , 0., 7.};
rotVector = { 0. , 1., 0. };
rotSpeed = { 1., 1., 1. };
scale = { .75,.75,.75 };
localTransform = modelStack.top();              // start with sun's coordinate
localTransform *= glm::translate(glm::mat4(1.f),
    glm::vec3(cos(speed[0] * dt) * dist[0], sin(speed[1] * dt) * dist[1], sin(speed[2] * dt) * dist[2]));
modelStack.push(localTransform);            // store planet-sun coordinate
localTransform *= glm::rotate(glm::mat4(1.f), rotSpeed[0] * (float)dt, rotVector);
localTransform *= glm::scale(glm::vec3(scale[0], scale[1], scale[2]));
if (m_sphere2 != NULL)
    m_sphere2->Update(localTransform);
```

Above are the updates for the sun and the planet.The sun is in the center and rotates on its Y axis slowly. The Earth orbits the sun on the XY plane and rotates on its Y axis.

```
// position of the first moon
speed = { 3, 3, 3 };
dist = { 1.25, .50, 1.25 };
rotVector = { 1.,1.,1. };
rotSpeed = { .25, .25, .25 };
scale = { .20f, .20f, .20f };
localTransform = modelStack.top();
localTransform *= glm::translate(glm::mat4(1.f),
    glm::vec3(cos(speed[0] * dt) * dist[0], sin(speed[1] * dt) * dist[1], sin(speed[2] * dt) * dist[2]));
modelStack.push(localTransform);            // store moon-planet-sun coordinate
localTransform *= glm::rotate(glm::mat4(1.f), rotSpeed[0] * (float)dt, rotVector);
localTransform *= glm::scale(glm::vec3(scale[0], scale[1], scale[2]));
if (m_sphere3 != NULL)
    m_sphere3->Update(localTransform);

modelStack.pop();
modelStack.pop();

speed = { 1, 1., 1. };
dist = { 0, 6., 6. };
rotVector = { 1 , 0, 0 };
rotSpeed = { 1, 1, 1. };
scale = { .02,.02,.02 };
localTransform = modelStack.top();
localTransform *= glm::translate(glm::mat4(1.f),
    glm::vec3(sin(speed[0] * dt) * dist[0], cos(speed[1] * dt) * dist[1], sin(speed[2] * dt) * dist[2]));
modelStack.push(localTransform);            // store planet-sun coordinate
localTransform *= glm::rotate(glm::mat4(1.f), -80.f, glm::vec3(1, 0, 0));
localTransform *= glm::rotate(glm::mat4(1.f), rotSpeed[0] * (float)dt , rotVector);
localTransform *= glm::scale(glm::vec3(scale[0], scale[1], scale[2]));
if (m_mesh != NULL)
    m_mesh->Update(localTransform);
```

Above are the updates for the moon and spaceship. The moon orbits the Earth with a slight tilt off of the XY plane. And the spaceship orbits the Sun on the YZ plane. I make the spaceship face the sun by adding a -80 degree rotation for each update. I also had to change the sin cos order in the translation for the spaceship as the 0 in dist would cause the cos function to mess up with the 0.

3. Camera Movements

```
void Camera::updateView(glm::vec3 cameraUpdate) {

    cameraFront = cameraUpdate;
    view = glm::lookAt(cameraPos, cameraFront + cameraPos, cameraUp);
}

void Camera::cameraPosVert(float speed) {
    //cameraPos += speed * cameraUp;
    cameraPos += speed * cameraFront;
    view = glm::lookAt(cameraPos, cameraFront + cameraPos, cameraUp);
}

void Camera::cameraPosHorz(float speed) {
    cameraPos += glm::normalize(glm::cross(cameraFront, cameraUp)) * speed;
    view = glm::lookAt(cameraPos, cameraFront + cameraPos, cameraUp);
}

void Camera::zoom(float fov) {
    projection = glm::perspective(glm::radians(fov), //the FoV typically 90 d
        float(800) / float(600), //Aspect Ratio, so Circles stay Circular
        0.01f, //Distance to the near plane, normally a small value like this
        100.0f); //Distance to the far plane,

}
```

Camera functions, updateView uses the mouse movements to rotate the camera. cameraPosVert moves the camera forwards or backwards depending on the speed input (+/-). cameraPosHorz moves the camera left or right in a similar way. Zoom changes the perspective based on fov float that is determined by the scroll wheel inputs.

NOTE: When you run the program and your mouse is hovering over the play button in Visual Studio, move the camera (mouse) down and to the left to put the solar system in view!

4. Graphics/Engine pipeline

Graphics was much unchanged besides the update function.

```
bool firstMouse = true;
float yaw = -90.0f; // yaw i
float pitch = 0.0f;
float lastX = 800.0f / 2.0;
float lastY = 600.0 / 2.0;
float fov = 45.0f;
```

Had to declare these functions globally in Engine so that the scroll_callback could appropriately access and edit.

```
m_graphics->getCamera()->updateView(cameraFront);
m_graphics->getCamera()->zoom(fov);

if (glfwGetKey(m_window->getWindow(), GLFW_KEY_D) == GLFW_PRESS)
m_graphics->getCamera()->cameraPosHorz(0.5f);
if (glfwGetKey(m_window->getWindow(), GLFW_KEY_A) == GLFW_PRESS)
m_graphics->getCamera()->cameraPosHorz(-0.5f);
if (glfwGetKey(m_window->getWindow(), GLFW_KEY_W) == GLFW_PRESS)
    m_graphics->getCamera()->cameraPosVert(0.5f);
if (glfwGetKey(m_window->getWindow(), GLFW_KEY_S) == GLFW_PRESS)
    m_graphics->getCamera()->cameraPosVert(-0.5f);
```

Significant portion of Engine::ProcessInput().

```
static void scroll_callback(GLFWwindow* window, double xoffset, double yoffset)
{
    fov -= (float)yoffset;
    if (fov < 1.0f)
        fov = 1.0f;
    if (fov > 45.0f)
        fov = 45.0f;


}
```

Scroll_Callback which was set in the initialization function of Engine.