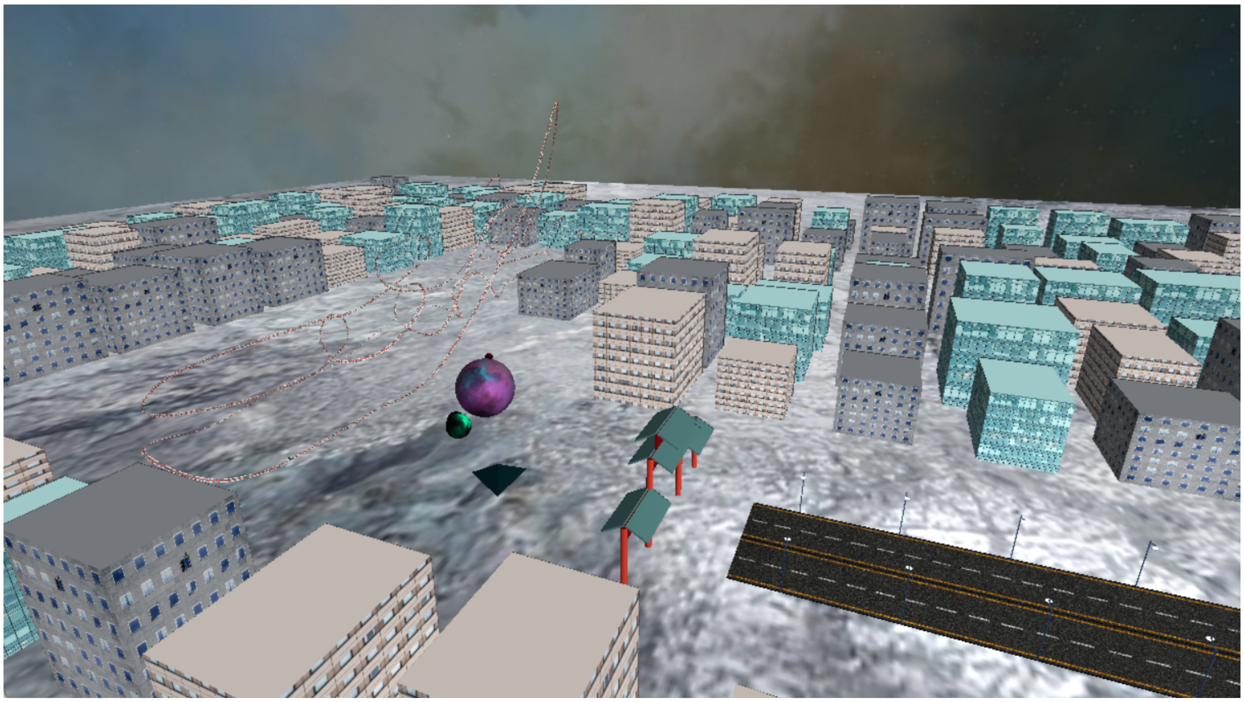
Subject: CSCI420 - Computer Graphics
Assignment 2: Simulating a Roller Coaster
Author: Baihua Yang
USC ID: 3281934045

Platform: Windows 11, Visual Studio 2022.
c++ version: ISO C++17 Standard.

Description: In this assignment, we use Catmull-Rom splines along with OpenGL core profile shader-based texture mapping and Phong shading to create a roller coaster simulation.



Core Credit Features

1. Uses OpenGL core profile, version 3.2 or higher - Y
2. Completed all Levels:
   Level 1 : - Y
   Level 2 : - Y
   Level 3 : - Y
   Level 4 : - Y
   Level 5 : - Y
3. Rendered the camera at a reasonable speed in a continuous path/orientation - Y
4. Run at interactive frame rate (>15fps at 1280 x 720) - Y
5. Understandably written, well commented code - Y
6. Attached an Animation folder containing not more than 1000 screenshots - Y
7. Attached this ReadMe File - Y

Extra Credit Features

1. Render a T-shaped rail cross section - N
2. Render a Double Rail - Y
3. Made the track circular and closed it with C1 continuity - Y
4. Any Additional Scene Elements? (list them here) - Y. An animated planet model(texture-mapped), street lamps(point lights), road(texture-mapped), buildings(texture-mapped) and a paifang model.
5. Render a sky-box - Y. Animated(Rotates with time).
6. Create tracks that mimic real world roller coaster - Y. Magic Mountain.
7. Generate track from several sequences of splines - N. Not used in this scene but the code supports this feature.
8. Draw splines using recursive subdivision - Y.
9. Render environment in a better manner - Y.
10. Improved coaster normals - N.
11. Modify velocity with which the camera moves - Y. But not exactly using the given formula.

```
void RollerCoaster::onUpdate() {
    // physical calculation
    float deltaTime = Timer::getInstance()->getDeltaTime();
    vec3 tangent = mix( // interpolates to get current tangent
        vertexTangents[currentVertexIndex],
        vertexTangents[currentVertexIndex % numOfVertices],
        currentSegmentProgress
    );
    float drag = dot(-tangent, Physics::GRAVITY); // drag speed at the given
tangent
    speed -= drag * deltaTime;
    speed = std::max(speed, minSpeed);
    float step = speed * deltaTime; // final move step

    // consume step
    while (step > 0) {
        float distanceToNext = // remaining distance to next vertex
            vertexDistances[currentVertexIndex] * (1 - currentSegmentProgress);
        if (step < distanceToNext) break; // done if remaining step is not
enough to move forward

        // move to the next vertex and reset distance from current vertex
        step -= distanceToNext;
        currentSegmentProgress = 0;
        currentVertexIndex++;

        // if reach the last vertex
        if (currentVertexIndex == numOfVertices - 1) {
            // if the roller-coaster is repeating
```

```
            if (!isRepeating) {
                step = 0;
                pause();
                reset(true);
            }
            // else stop
            else {
                reset(false);
                start();
            }
        }
    }
    // update progress between current and next vertices
    currentSegmentProgress += step / vertexDistances[currentVertexIndex];

    // update seat position
    // in function moveSeat(), the position is calculated by interpolating the
positions of current and next vertices by currentSegmentProgress
    moveSeat();
}
```

12. Derive the steps that lead to the physically realistic equation of updating u - Y.

```
let delta_h = maxHeight - currentHeight
1. delta_h = 0.5 * g * t^2 => t = sqrt(2 * delta_h / g)
2. velocity = a * t = g * t = g * sqrt(2 * delta_h / g) = sqrt(2 * g * delta_h)
3. from 1 and 2: d(delta_h) / dt = g * t => d(delta_h) = g * t * dt = dt *
sqrt(2 * g * delta_h)
We know length(dp / du) = speed at u
Now parameterize d(delta_h) by speed at u, we get u_new - u_old.
Therefore, u_new = u_old + dt * sqrt(2 * g * delta_h) / length(dp / du)
```

Additional Features: (Please document any additional features you may have implemented other than the ones described above)

1. Multiple light sources. (1 directional light and multiple point lights)
2. Controllable player and a world camera.
3. .sp contains only point positions, no need to include the number of points. track.txt contains only .sp file paths, no need to include the number of .sp files.
4. All details can be found in Utility.h and Utility.cpp.

```
template<class T> class Singleton;
class Timer;
class SceneManager;
class Entity;

class Component;
class Transform;
class Renderer;
class Physics;
class Camera;
class Light;
class DirectionalLight;
class PointLight;
class PlayerController;
class RollerCoaster;

class VertexArrayObject;

class Texture;
class Texture2D;
class Cubemap;

struct Shape;
```

Open-Ended Problems: (Please document approaches to any open-ended problems that you have tackled)

1. Use quaternion to represent rotations of objects.

Keyboard/Mouse controls: (Please document Keyboard/Mouse controls if any)

1. Press w, a, s, d to move player / world camera.
2. Press Spacebar to jump(player) / move upward(world camera).
3. Press c to move downward(world camera).
4. Press e to ride a roller-coaster. (Need to get close enough. Distance = 5 by default)
5. Press e to stop and reset a running roller-coaster.
6. Press r to lock / unlock player's view when riding a roller-coaster.
7. Press p to switch between player and world camera.
8. Rotate first-person view with mouse drag.
9. Press x to toggle screenshots recording.

Names of the .cpp files you made changes to:

1. basicPipelineProgram.cpp
2. pipelineProgram.cpp
3. hw2.cpp
4. Utility.cpp