Project Final Report

Runqiu Pan

December 2nd, 20`9

Manual

This project aims to create an interactive OpenGL game which depicts the free kick part of a soccer game. To run the game, please follow the instructions below:

1. Navigate to the main folder where Proj.cpp is in.
2. Type in command and run:
   ```
   premake4 gmake
   ```
   This will use Premake program to conveniently generate a Makefile for the compilation of the program. Premake reads a file called premake4.lua, which is fundamentally the same as the ones in previous assignments. There were attempts to modify this file. See detail in Object 8.
3. Type in command and run:
   ```
   make
   ```
   This will compile the program, creating an executable called Proj.
4. Type in command and run:
   ```
   ./Proj scene.lua
   ```
   There is one argument to this program, which the file name of a lua script which defines the models used in the game. The name of the lua script which was created for the game is scene.lua

The game starts with an overview of the whole world, and by switching views, the user is able to look more closely. There is the main game section, where the player can shoot the ball into the net. There is also a fan view in which the player can view the fans, and a tree view in which the player could relax his eyes by watching a lively tree. There are buttons in the control panel that can switch views.

There are several controls in this game. There is a menu, and in the menu, the player can enable or disable the display of the texture mapping of the soccer field, the display of fans and seats, actions of fans jumping up to cheer for the match, and the display of the tree. By disabling features, the performance of the game is greatly improved. The player can move the camera using left, right, up down arrow keys and pageUp, pageDown keys. Clicking on "See tree" would set a random camera view for the tree.

In addition to view switching, in the control panel, there are also the main control for the player to use and shoot the ball. There is a Power slider, which controls how much power the player uses to kick the ball, a Vertical Direction slider which controls how high the player wants to kick, and a Horizontal Direction which control how much left or right the player wants to kick. Finally, there is the shoot button, which brings up all the excitement after clicking. There is a reset feature which would reset the kicker, the ball, and the goalkeeper, which is the button "R".

There is nothing very interesting in the console output. Most of the output is from the media player used to play the sound of various effects, and there is also the result of whether the player goals, in case it is not clearly from the game itself. No debug information is shown in the console.

Implementation

The base structure code of this project is from Assignment 3. In this section, ten objectives are listed, and there are descriptions of implementation for each objective.

Objective 1: the scene is modeled properly with objects mentioned in the proposal.

In the scene, objects are introduced in two ways. The first one is to use the lua script, which includes primitive 3D models like cube and sphere. Cylinder is added as a new mesh, and it is used heavily in the game. The objects include the ground, the soccer field, the goal which includes the crossbar and two posts, and both goals are rendered. There is a soccer ball, a kicker and a goalkeeper who are stickman figures, with a cylinder trunk and limbs, and a spherical head. I used cubes for seats, and stickman figures for fans. Trees are rendered using cylinders as well.

The second way is to use vertices indices defined manually and bound to the buffer in the same classic way as in Assignment 1. The particles used in the particle system are defined in this way, as well as the soccer field, which is a rectangle positioned right above the ground model.

The unit of measure is 1:1. That is, for every translation of the model about 1 unit, it is equivalent to one meter in the real life. Each fan takes one-meter wide seat, which makes it 90 people on each side of the field.

The codes are mostly in Proj.cpp and scene.lua

Objective 2: soccer field is mapped with a texture.

The original idea was to map to a lua model, then I changed my mind and texture map the image to a rectangle specified traditionally. I used ELEMENT_ARRAY_BUFFER this time which I did not use for Assignment 1. Creating a texture was a challenge. For this I created a new set of vertex buffer and fragment buffer for texture, so that it could safely store texture coordinates without the risk of polluting lua object models. A new set of vertex shader and fragment shader are also created. I got stuck on this objective for a while because the texture was skewed when mapped to the rectangle. I originally thought I might do it wrong with the model-view transformation, but then it turned out that it was because my texture size was too small. I found a much larger one and it worked.

Creating the texture is at Proj.cpp: 159, and applying the texture is at Proj.cpp: 908. The shader files are in Assets/ : texture_VertexShader.vs and texture_FragmentShader.fs. It also uses a library stb_image.

Objective 3: kicker's animation looks realistic and the ball flies in a reasonable direction.

The stickman would pull back his left leg, fast swing it forward to hit the ball, and return to the original position. To make it more realistic, the movement considers the power value set in the control. The higher the power is, the faster the kicker would swing the leg. The upper body rotates with the left leg. The technique used in this animation is rotation around arbitrary axis, which is the body in this case.

A bug in this objective is that the kicker could not fully return to the original gesture after kicking the ball. I highly suspect that it is related to floating point calculation, although I did not look into it. Resetting does not help, since it is hard to track how much angle the player turns in each kick, given the floating-point error, resetting can only set the body position, but the arms and legs would be in the same incorrect relative position to the body. The head and the right leg are not affected since they are not rotated.

The flying trail of the soccer ball involves physics. Gravity is introduced, as well as some other forces roughly. The kicking power and vertical direction are related in the way that for the same power, the higher the player kicks, the closes the ball flies. Trigonometric functions are used to calculate the speed and acceleration of the ball split vertically and horizontally.

There are some challenges in coordinating bouncing with the overall movement. I set the maximum bounce time to be 5, so that the ball would not hang in the air when it stops moving horizontally. It looks realistic enough, but more could be done to apply more realistic physics laws.

The code for simulation is at Proj.cpp: 848 and 704

Objective 4: the goalkeeper moves towards the correct direction.

In the final implementation, the goalkeeper moves horizontally towards to ball. There is a bounding box of the goalkeeper. At the time the ball crosses the goal line, if the ball's position is outside the goal or insider the bounding box, then the kick loses this round.

After 0,3 seconds of reaction time, the goalkeeper will compare his position with the balls, and move trying to match the ball's x position. The goalkeeper has a maximum speed., and he won't go outside the range of the goal.

Code at Proj.cpp: 808

Objective 5: a tree is rendered around the field using L-system.

When enabled, there is a tree at the north-east corner of the field. It is rendered using L-system. The tree trunk and tree branches are made using the cylinder model, which gets slimmer and shorter after each iteration. A maple leaf was modeled using line loops, but due to time constraint, it was not able to be incorporated onto the tree. The challenge in this is to figure out where the leaves grow on the branch, and the orientation of each leaf.

The rules are simple symbolically:

Symbols: A, L, R

Axiom: A

Rule: A -> LARA

L means that the branch extends another branch which is formed by rotating the original branch 15 degrees on z axis, and 60 degrees on y axis. The rotation base is the root of that branch. Then it is moved two meters to the tip of the original branch. R means rotating -30 degrees on the z axis, 30 degrees on the y axis, with the same rotation point and extension. The recursion runs 12 rounds.

Code at: Proj.cpp: 955

Objective 6: seats and fans are rendered correctly, and animation for fans cheering (human wave).

The fan and seat model are designed in the lua script. Each of them is one meter apart, so there are 90 people on each side of the field in each row. For waving, at each time, each instance's height is determined by the time and its index, with small difference between itself and its previous instance, so that a curve wave is achieved.

Code at: Proj.cpp: 673 and 926

Objective 7: Particle system is implemented as fireworks for scoring.

After the kicker scores, a particle system is activated. A Particle class is created to determine its position and velocity. Position and velocity are vec3 variables of the class, and the constructor initialized the particle at position (0, 0, 0), with one-unit speed upward. Each particle is a small triangle rendered using glDrawArrays.

Before rendering, 512 particles are initialized, and its velocity is reset to be a new vec3 variable (z always set to 0.0f). At each rendering, each particle translates according to its own velocity, so it looks like a explosion. Gravity force is implemented to increase its realism.

For data structure, the programs put all position values in one array and render all triangles at once, instead of binding buffer, reading data for each particle. In this way, the performance gets improved.

One thing needs to be mentioned here: after the explosion, the particles spreads out randomly, and overall it forms a rectangle. It is not due to view matrix, but the random number generator which specifies all the velocity in this strange way. I do not have a solution for this yet, but there could be one upon further digging.

Code at: Proj.cpp: 215 and 966

Objective 8: The goal net moves when hit with the ball.

I did not implement this objective.

Objective 9: Sound is added when the ball is kicked or when scoring.

Sound is added in three places. The first one is when the kicker kicks the ball, the second one is when the it goals, in which situation the player will hear crowd cheering. The third place is when the ball hits the post or the crossbar. The third one is hard to trigger, so that a hidden trigger keyboard "c" will always reset the kicker at the same location for the player to try.

This objective was hard to achieve because the lab does not allow installation of new tools for audio engineering. I found a lot of solutions, but they are required third party software. In the end, I had to use linux's system() calls to run command-line instructions directly, and Linux happens to have a default media player using the command "ffplay" to play a mp3 file. This way is not recommended, but it is the only way I could think of under the limitation of CG lab.

Objective 10: use alpha blending to create shadows for the kicker and the ball.

I did not implement this objective.