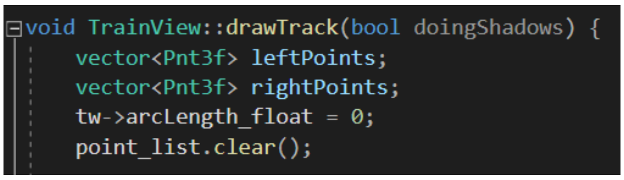
Tech document

This document will explain how I finish this project. I broke this document into two parts, including track, train.

1. Track

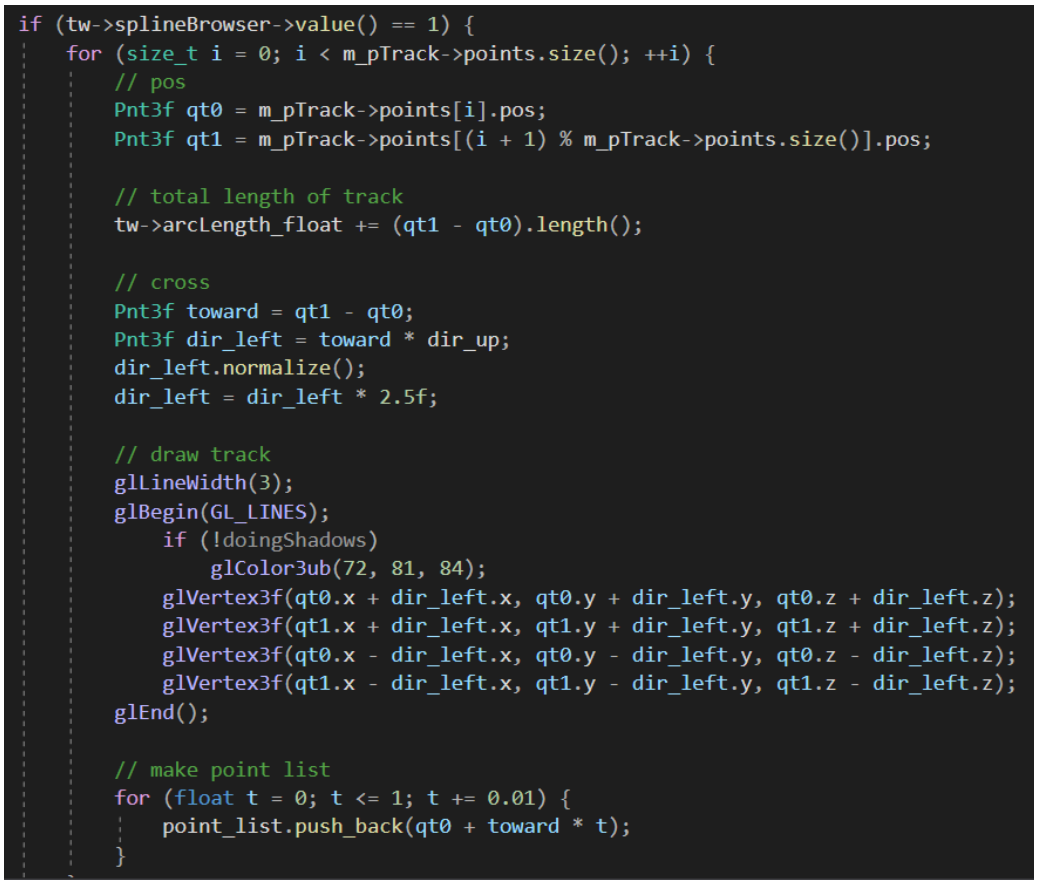
I drew track with the drawTrack function in TrainView.



```
void TrainView::drawTrack(bool doingShadows) {
    vector<Pnt3f> leftPoints;
    vector<Pnt3f> rightPoints;
    tw->arcLength_float = 0;
    point_list.clear();
```

At first, I had two vectors to store the point on the left and right track respectively. Then, I reset length and point_list. Point_list is a vector that stores the points on the track.

1-a. Linear track

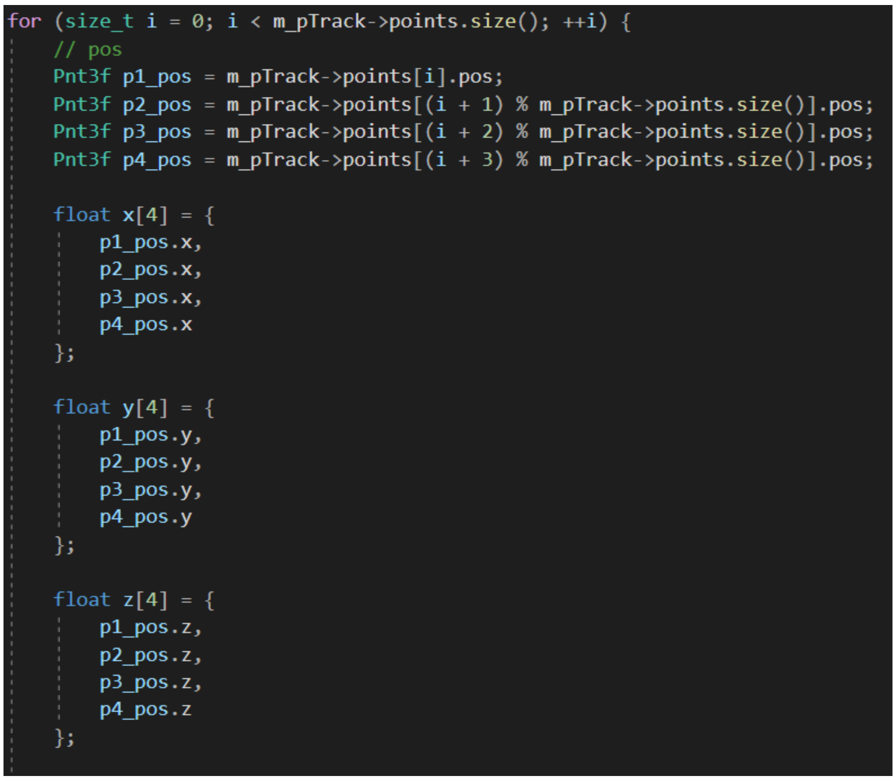


```
if (tw->splineBrowser->value() == 1) {
    for (size_t i = 0; i < m_pTrack->points.size(); ++i) {
        // pos
        Pnt3f qt0 = m_pTrack->points[i].pos;
        Pnt3f qt1 = m_pTrack->points[(i + 1) % m_pTrack->points.size()].pos;

        // total length of track
        tw->arcLength_float += (qt1 - qt0).length();

        // cross
        Pnt3f toward = qt1 - qt0;
        Pnt3f dir_left = toward * dir_up;
        dir_left.normalize();
        dir_left = dir_left * 2.5f;

        // draw track
        glLineWidth(3);
        glBegin(GL_LINES);
            if (!doingShadows)
                glColor3ub(72, 81, 84);
            glVertex3f(qt0.x + dir_left.x, qt0.y + dir_left.y, qt0.z + dir_left.z);
            glVertex3f(qt1.x + dir_left.x, qt1.y + dir_left.y, qt1.z + dir_left.z);
            glVertex3f(qt0.x - dir_left.x, qt0.y - dir_left.y, qt0.z - dir_left.z);
            glVertex3f(qt1.x - dir_left.x, qt1.y - dir_left.y, qt1.z - dir_left.z);
        glEnd();

        // make point list
        for (float t = 0; t <= 1; t += 0.01) {
            point_list.push_back(qt0 + toward * t);
        }
```

I first derived the coordinates of the point. Toward points the front, and we can use toward cross up to get the left vector. The left vector is used to find the left and right track's position.

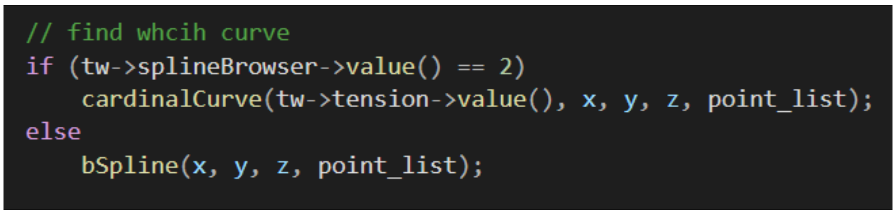
After drawing the track, I broke each way from one control point to the next one into 100 segment and store all the points.

1-b. curve track



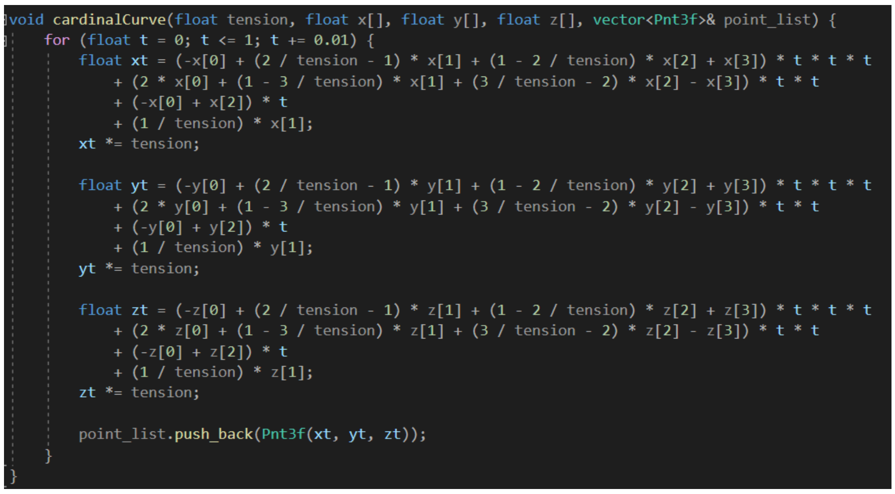
```cpp
for (size_t i = 0; i < m_pTrack->points.size(); ++i) {
    // pos
    Pnt3f p1_pos = m_pTrack->points[i].pos;
    Pnt3f p2_pos = m_pTrack->points[(i + 1) % m_pTrack->points.size()].pos;
    Pnt3f p3_pos = m_pTrack->points[(i + 2) % m_pTrack->points.size()].pos;
    Pnt3f p4_pos = m_pTrack->points[(i + 3) % m_pTrack->points.size()].pos;

    float x[4] = {
        p1_pos.x,
        p2_pos.x,
        p3_pos.x,
        p4_pos.x
    };

    float y[4] = {
        p1_pos.y,
        p2_pos.y,
        p3_pos.y,
        p4_pos.y
    };

    float z[4] = {
        p1_pos.z,
        p2_pos.z,
        p3_pos.z,
        p4_pos.z
    };
```

In this part, I first found the position of the control point and that of the next 3 ones.

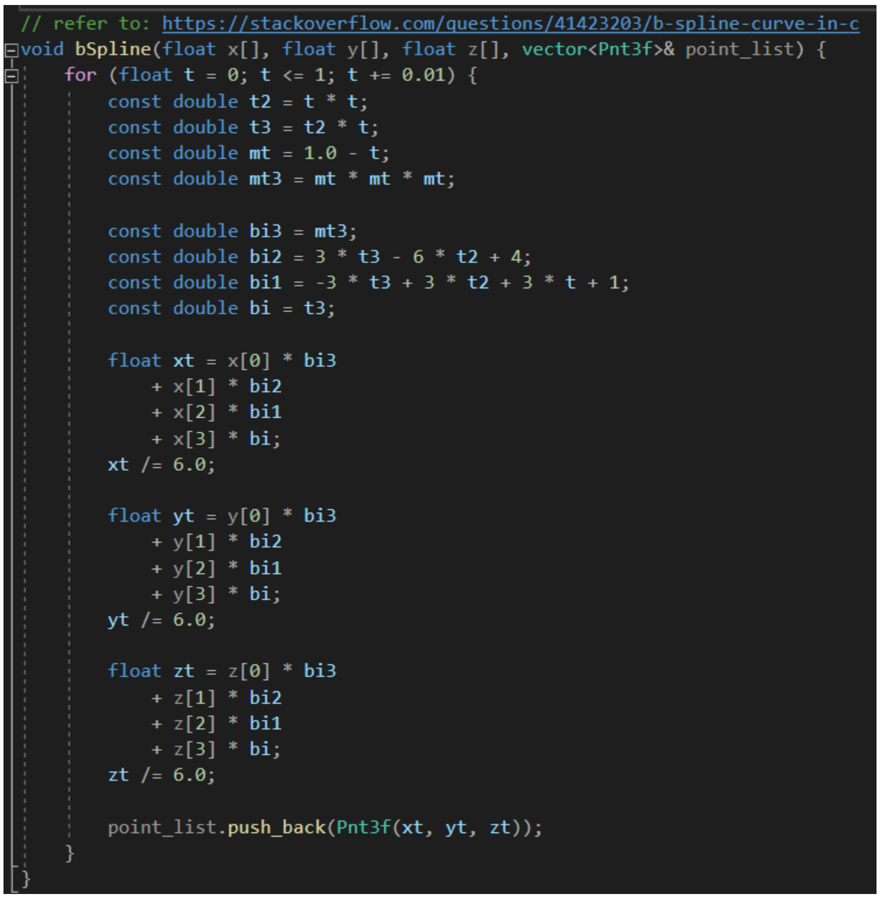


```cpp
// find whcih curve
if (tw->splineBrowser->value() == 2)
    cardinalCurve(tw->tension->value(), x, y, z, point_list);
else
    bSpline(x, y, z, point_list);
```

Next, I put the position information into different function according to the type of the curve to find the points on the track.

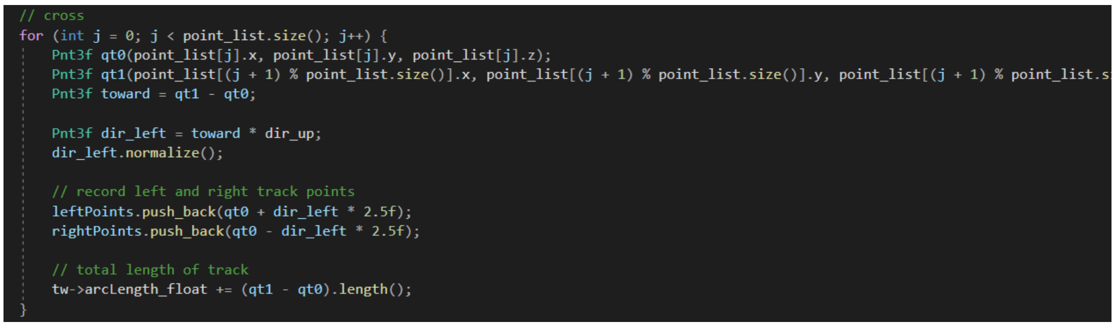


```cpp
void cardinalCurve(float tension, float x[], float y[], float z[], vector<Pnt3f>& point_list) {
    for (float t = 0; t <= 1; t += 0.01) {
        float xt = (-x[0] + (2 / tension - 1) * x[1] + (1 - 2 / tension) * x[2] + x[3]) * t * t * t
            + (2 * x[0] + (1 - 3 / tension) * x[1] + (3 / tension - 2) * x[2] - x[3]) * t * t
            + (-x[0] + x[2]) * t
            + (1 / tension) * x[1];
        xt *= tension;

        float yt = (-y[0] + (2 / tension - 1) * y[1] + (1 - 2 / tension) * y[2] + y[3]) * t * t * t
            + (2 * y[0] + (1 - 3 / tension) * y[1] + (3 / tension - 2) * y[2] - y[3]) * t * t
            + (-y[0] + y[2]) * t
            + (1 / tension) * y[1];
        yt *= tension;

        float zt = (-z[0] + (2 / tension - 1) * z[1] + (1 - 2 / tension) * z[2] + z[3]) * t * t * t
            + (2 * z[0] + (1 - 3 / tension) * z[1] + (3 / tension - 2) * z[2] - z[3]) * t * t
            + (-z[0] + z[2]) * t
            + (1 / tension) * z[1];
        zt *= tension;

        point_list.push_back(Pnt3f(xt, yt, zt));
    }
}
```



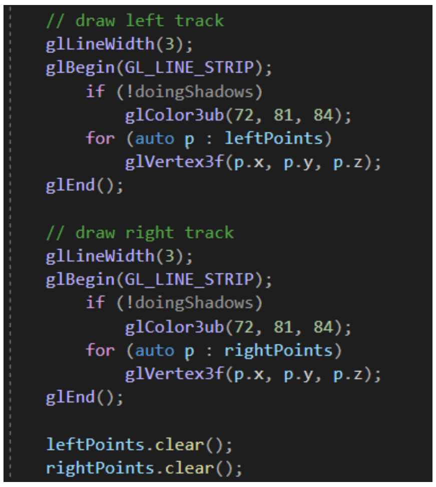
```cpp
// refer to: https://stackoverflow.com/questions/41423203/b-spline-curve-in-c
void bSpline(float x[], float y[], float z[], vector<Pnt3f>& point_list) {
    for (float t = 0; t <= 1; t += 0.01) {
        const double t2 = t * t;
        const double t3 = t2 * t;
        const double mt = 1.0 - t;
        const double mt3 = mt * mt * mt;

        const double bi3 = mt3;
        const double bi2 = 3 * t3 - 6 * t2 + 4;
        const double bi1 = -3 * t3 + 3 * t2 + 3 * t + 1;
        const double bi = t3;

        float xt = x[0] * bi3
            + x[1] * bi2
            + x[2] * bi1
            + x[3] * bi;
        xt /= 6.0;

        float yt = y[0] * bi3
            + y[1] * bi2
            + y[2] * bi1
            + y[3] * bi;
        yt /= 6.0;

        float zt = z[0] * bi3
            + z[1] * bi2
            + z[2] * bi1
            + z[3] * bi;
        zt /= 6.0;

        point_list.push_back(Pnt3f(xt, yt, zt));
    }
}
```

In this section, I ran into some problem about glm matrix multiplication, so I decided to brutally solve it.



```cpp
// cross
for (int j = 0; j < point_list.size(); j++) {
    Pnt3f qt0(point_list[j].x, point_list[j].y, point_list[j].z);
    Pnt3f qt1(point_list[(j + 1) % point_list.size()].x, point_list[(j + 1) % point_list.size()].y, point_list[(j + 1) % point_list.s
    Pnt3f toward = qt1 - qt0;

    Pnt3f dir_left = toward * dir_up;
    dir_left.normalize();

    // record left and right track points
    leftPoints.push_back(qt0 + dir_left * 2.5f);
    rightPoints.push_back(qt0 - dir_left * 2.5f);

    // total length of track
    tw->arcLength_float += (qt1 - qt0).length();
}
```

Then, I did the same thing that I had done for linear track. I found out the vector pointing front and named it “toward”, and I used toward and up vector to find the left vector. Finally, I used the left vector to calculate the left and right track.

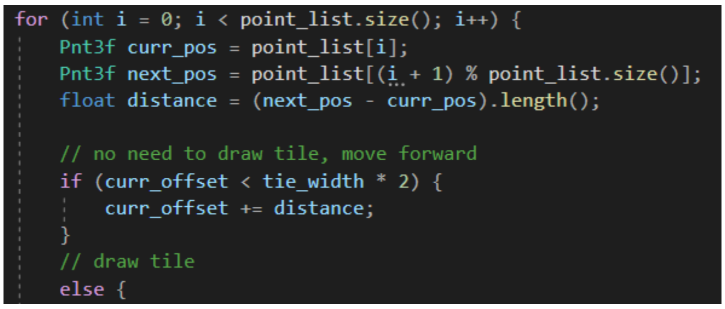


```
// draw left track
glLineWidth(3);
glBegin(GL_LINE_STRIP);
    if (!doingShadows)
        glColor3ub(72, 81, 84);
    for (auto p : leftPoints)
        glVertex3f(p.x, p.y, p.z);
glEnd();

// draw right track
glLineWidth(3);
glBegin(GL_LINE_STRIP);
    if (!doingShadows)
        glColor3ub(72, 81, 84);
    for (auto p : rightPoints)
        glVertex3f(p.x, p.y, p.z);
glEnd();

leftPoints.clear();
rightPoints.clear();
```

After drawing the left and right track, I cleared both of the vectors to prevent repeating drawing.

1-c. track tiles

The basic idea I used when I drew the tiles was that I kept moving forward and used how far I go to decide whether to draw a tile.



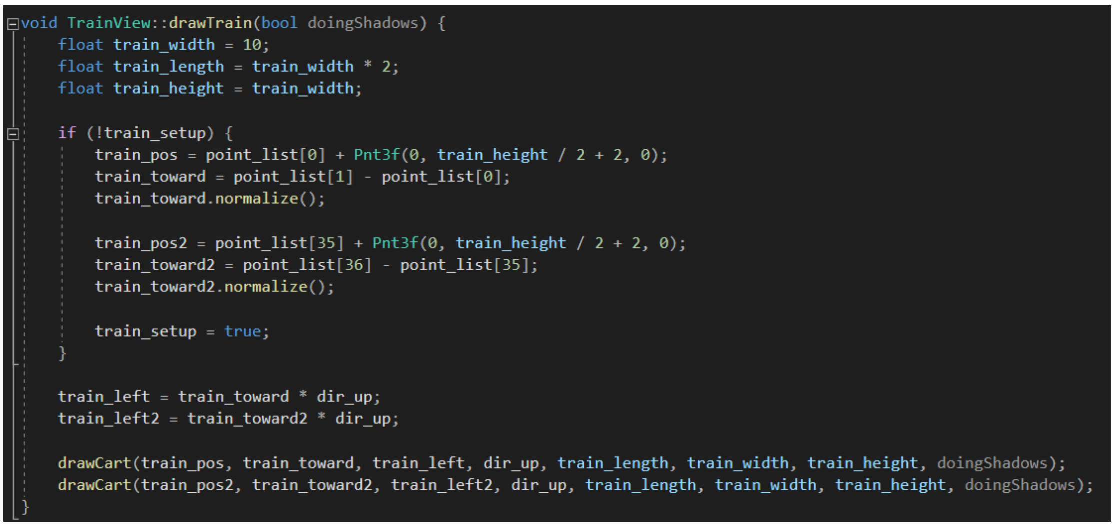
```
for (int i = 0; i < point_list.size(); i++) {
    Pnt3f curr_pos = point_list[i];
    Pnt3f next_pos = point_list[(i + 1) % point_list.size()];
    float distance = (next_pos - curr_pos).length();

    // no need to draw tile, move forward
    if (curr_offset < tie_width * 2) {
        curr_offset += distance;
    }
    // draw tile
    else {
```

I moved along the points on the track. Once I moved more than twice the tile's width, I would draw a tile, clear the current moving distance, and keep moving forward until I met the end of the track.

2. Train

2-a. draw train

I drew train with the drawTrain function in TrainView.



```
void TrainView::drawTrain(bool doingShadows) {
    float train_width = 10;
    float train_length = train_width * 2;
    float train_height = train_width;

    if (!train_setup) {
        train_pos = point_list[0] + Pnt3f(0, train_height / 2 + 2, 0);
        train_toward = point_list[1] - point_list[0];
        train_toward.normalize();

        train_pos2 = point_list[35] + Pnt3f(0, train_height / 2 + 2, 0);
        train_toward2 = point_list[36] - point_list[35];
        train_toward2.normalize();

        train_setup = true;
    }

    train_left = train_toward * dir_up;
    train_left2 = train_toward2 * dir_up;

    drawCart(train_pos, train_toward, train_left, dir_up, train_length, train_width, train_height, doingShadows);
    drawCart(train_pos2, train_toward2, train_left2, dir_up, train_length, train_width, train_height, doingShadows);
}
```

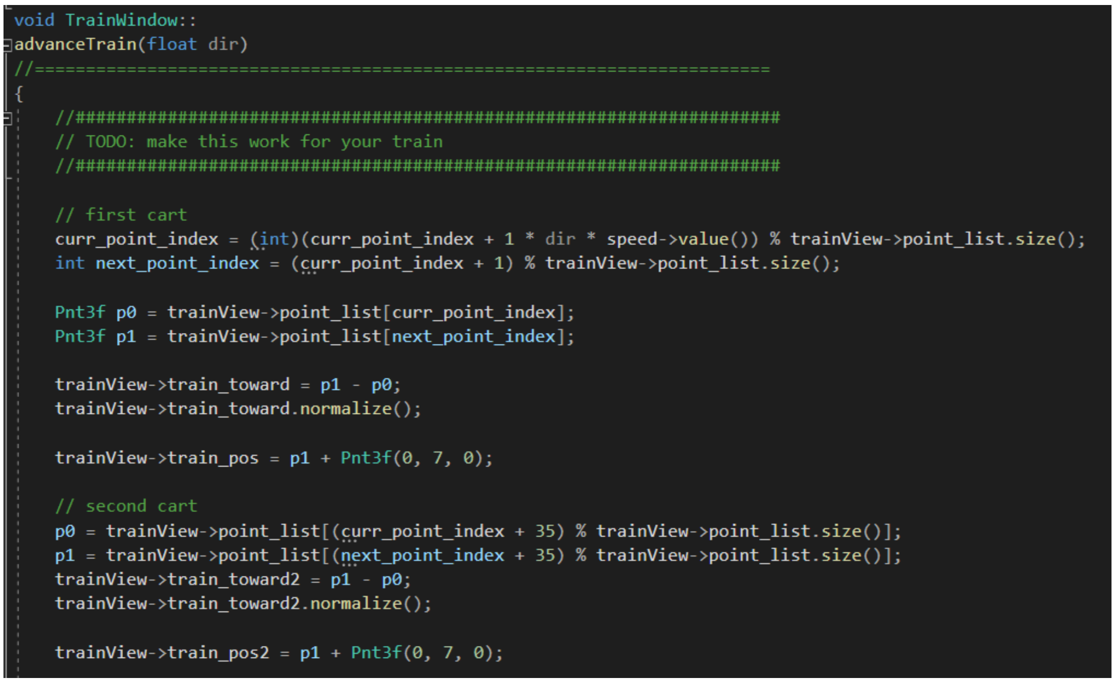
When the program first executes, the train is not set up yet. Therefore, I first checkout if it is set up, if no, set it to the beginning of the track.

Same, I used the point where they were standing to find out the coordinates (forward, up, and left).

Then, I sent the information of my two train into drawCart to draw them repectively. drawCart takes the information about the train, including the position, coordinates, and the size.

2-b. train running

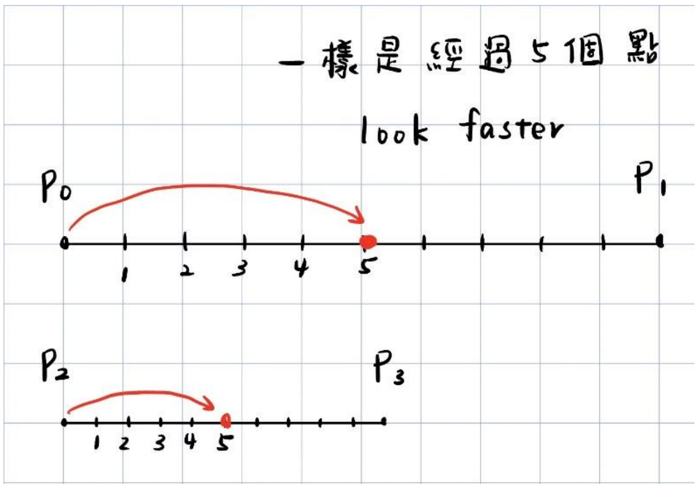
This part was done in the advanceTrain function in TrainWindow.



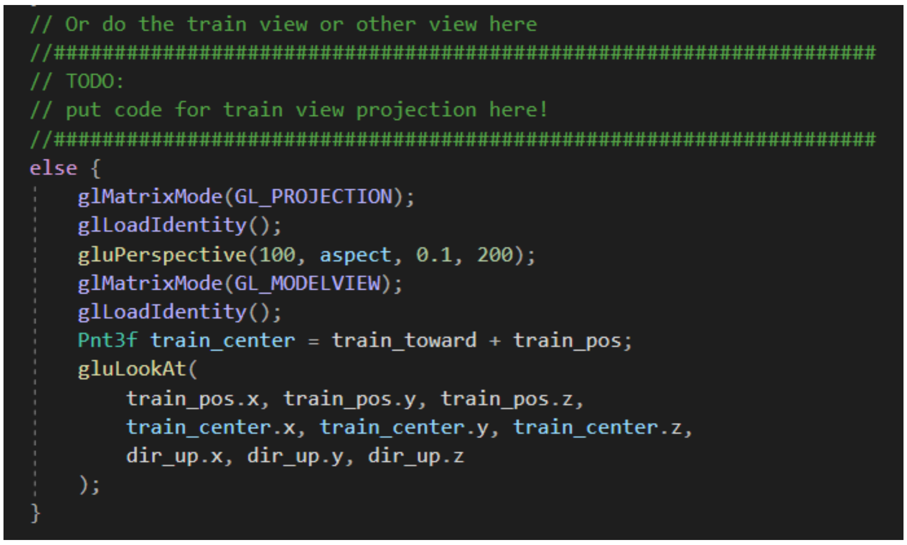
```
void TrainWindow::
advanceTrain(float dir)
//========================================================================
{
    //#########################################################################
    // TODO: make this work for your train
    //#########################################################################

    // first cart
    curr_point_index = (int)(curr_point_index + 1 * dir * speed->value()) % trainView->point_list.size();
    int next_point_index = (curr_point_index + 1) % trainView->point_list.size();

    Pnt3f p0 = trainView->point_list[curr_point_index];
    Pnt3f p1 = trainView->point_list[next_point_index];

    trainView->train_toward = p1 - p0;
    trainView->train_toward.normalize();

    trainView->train_pos = p1 + Pnt3f(0, 7, 0);

    // second cart
    p0 = trainView->point_list[(curr_point_index + 35) % trainView->point_list.size()];
    p1 = trainView->point_list[(next_point_index + 35) % trainView->point_list.size()];
    trainView->train_toward2 = p1 - p0;
    trainView->train_toward2.normalize();

    trainView->train_pos2 = p1 + Pnt3f(0, 7, 0);
```

I moved the train along the track using the points I recorded before. The faster the speed is, the larger interval the next point gets. This is how I control the speed of the train.

As what I had done before, I broke the way between two control points into 100 segments, regardless of their distance. This creates the effect of simple physics.



2-c. train view



```
// Or do the train view or other view here
//########################################################################
// TODO:
// put code for train view projection here!
//########################################################################
else {
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(100, aspect, 0.1, 200);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    Pnt3f train_center = train_toward + train_pos;
    gluLookAt(
        train_pos.x, train_pos.y, train_pos.z,
        train_center.x, train_center.y, train_center.z,
        dir_up.x, dir_up.y, dir_up.z
    );
}
```

In gluLookAt, I set the eye (position of viewer) as the same as train's position. Center is the direction the viewer is looking, so it is train's position + train's toward. I set up is always (0, 1, 0).

Reference:
Track tiles
https://github.com/okh8609/CG_Project3_RollerCoaster/blob/master/Roller%20Coaster/Src/TrainView.cpp

b-spline function
https://stackoverflow.com/questions/41423203/b-spline-curve-in-c