浙江大学

课程名称：　计算机动画

姓　　名：　刘佳润

学　　院：　计算机科学与技术学院

专　　业：　数字媒体技术

学　　号：　3180105640

指导教师：　于金辉

2020 年　　9 月　　20 日

浙江大学实验报告

课程名称：计算机动画 实验类型：综合

实验项目名称：路径曲线与运动物体控制

学生姓名：刘佳润 专业：数字媒体技术 学号：3180105640

同组学生姓名：无 指导老师：于金辉

实验地点： 实验日期：2020 年 9 月 20 日

一、实验目的和要求

- 掌握 Cardinal 样条曲线的表示和算法，了解控制参数对曲线形状的影响。
- 对照 Cardinal 样条曲线的数学表示和程序代码的对应关系。

二、实验内容和原理

根据 grain 和 tension 值以及控制点绘制出相应的 Cardinal 样条曲线作为路径。在路径曲线上放置一小汽车，使其在路径上运动起来，汽车运动速度可调。

三、实验器材

C++ 11 & Qt 5.14.2

工程依托 Qt Creator 4.11.1 完成

四、实验步骤

1. Cardinal 曲线的生成

$$\begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \tau \begin{bmatrix} -1 & 2/\tau-1 & -2/\tau+1 & 1 \\ 2 & -3/\tau+1 & 3/\tau-2 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 1/\tau & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{p}_{i-1} \\ \mathbf{p}_i \\ \mathbf{p}_{i+1} \\ \mathbf{p}_{i+2} \end{bmatrix}$$

上图为 Cardinal 曲线的矩阵表示。在本实验中，我们要根据给定的若干控制点的位置以及确认的 grain 值（每两点之间的插值点个数）以及 tension 值（样条曲线通过控制点的平滑程度）来生成样条曲线。关键在于得到插值点的坐标。

在 Spline 类里，我们定义了一系列关于样条曲线的生成：

```
class Spline
{
public:
    // x[],y[]为控制点的坐标，n为控制点的数量
    // grain为每两个控制点之间的插值点数目，tension为通过控制点位置的曲线平滑程度
    Spline(QPoint p[100], int n, int grain, float tension);    // 构造函数：创建样条曲线
    ~Spline();

    QPoint spline[2056];    // 插值点数组

private:
    void CubicSpline(int np, QPoint* konts, int grain, float tension);      // 计算插值点
    void getCardinalMatrix(float tao);                                      // 根据tension计算Cardinal矩阵系数
    float Matrix(float a, float b, float c, float d, float u);              // 矩阵运算

    QPoint knots[1000];          // 控制点数组
    float m[16];            // 储存Cardinal矩阵
};
```

其中 void Spline::**CubicSpline**(int np, QPoint* knots, int grain, float tension) 函数计算插值点：根据 tension 带入函数 void Spline::**getCardinalMatrix**(float tao)得到转换矩阵 m，然后将此矩阵和四个点（即公式中最右面的列矩阵）带入矩阵运算函数 float

Spline::**Matrix**(float a, float b, float c, float d, float u)，从而得到插值点坐标。

另外，在生成曲线的过程中，需要对端点进行重复赋值！

下面是样条类的构造函数代码：

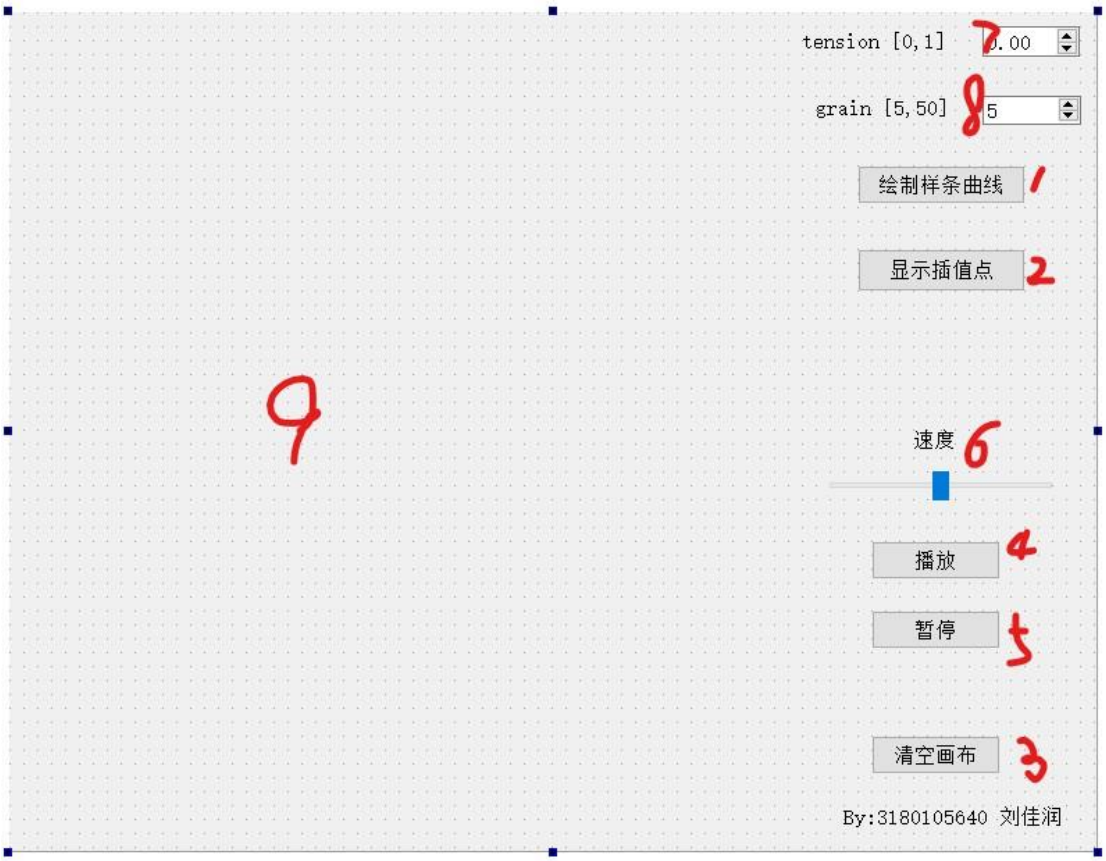
```
// 初始化：生成Cardinal样条曲线
Spline::Spline(QPoint p[100], int n, int grain, float tension)
{
    for(int i = 1; i <= n; i++)
    {
        knots[i] = p[i-1];
    }

    // 首尾控制点要重复赋值!
    knots[0] = p[0];
    knots[n+1] = p[n-1];

    // 生成插值点
    CubicSpline(n + 2, knots, grain, tension);
}
```

下面是三个关键函数的相关代码：

```
// 插值函数
void Spline::CubicSpline(int np, QPoint* knots, int grain, float tension)
{
    QPoint *s, *k0, *kml, *k1, *k2;
    int i, j;
    float u[50];
    getCardinalMatrix(tension);
    // 根据分段值确定u[]
    for(i = 0; i < grain; i++)
        u[i] = ((float) i)/grain;
    s = spline;     // 插值点数组
    kml = knots;    // 由控制点数组得到公式右边的列矩阵
    k0 = kml + 1;
    k1 = k0 + 1;
    k2 = k1 + 1;
    // 求插值点的坐标，除去前后的重合部分
    for(i = 0; i < np-3; i++)
    {
        for(j = 0; j < grain; j++)
        {
            float tmpx = Matrix(kml->x(), k0->x(), k1->x(), k2->x(), u[j]);
            float tmpy = Matrix(kml->y(), k0->y(), k1->y(), k2->y(), u[j]);
            s->setX(tmpx);
            s->setY(tmpy);
            s++;
        }
        k0++; kml++; k1++; k2++;
    }
    // 末尾的端点
    s->setX(k0->x());
    s->setY(k0->y());
}

// 根据tension生成Cardinal矩阵
void Spline::getCardinalMatrix(float tao)
{
    m[0]=-tao;   m[1]=2.-tao;  m[2]=tao-2.;   m[3]=tao;
    m[4]=2.*tao; m[5]=tao-3.;  m[6]=3.-2*tao; m[7]=-tao;
    m[8]=-tao;   m[9]=0.;      m[10]=tao;     m[11]=0.;
    m[12]=0.;    m[13]=1.;     m[14]=0.;      m[15]=0.;
}

// 矩阵运算
float Spline::Matrix(float a, float b, float c, float d, float u)
{
    float p0, p1, p2, p3;
    p0 = m[0]*a + m[1]*b + m[2]*c + m[3]*d;
    p1 = m[4]*a + m[5]*b + m[6]*c + m[7]*d;
    p2 = m[8]*a + m[9]*b + m[10]*c + m[11]*d;
    p3 = m[12]*a + m[13]*b + m[14]*c + m[15]*d;
    return (u*u*u*p0 + u*u*p1 + u*p2 + p3);
}
```

2. 图形界面的设计

Qt 的 widget 类可以图形化设计 ui，方便操作。最终的布局如下图所示：



上图的几个控件以及对应的槽函数如下表所示：

编号	控件名	功能说明	槽函数
1	draw_pushButton	绘制曲线	draw()
2	point_pushButton	显示插值点	show_point()
3	clear_pushButton	清空画布	clear()
4	play_pushButton	播放动画	play()
5	pause_pushButton	切换动画暂停/恢复	pause_resume()

6	speed_horizontalSlider	控制速度	无
7	tension_doubleSpinBox	控制 tension 值	无
8	grain_spinBox	控制 grain 值	无
9	pix	画布与演示区	无
未标注	timer	计时器，控制动画播放以及调整	ani_control()

Widget 类的部分关键成员变量如下：

```
private:
    Ui::Widget *ui;
    QPixmap pix;
    Spline *mSpline;

    // 控制点坐标
    QPoint p[100];

    State state;    // 绘制模式

    int n;                  // 控制点
    int part;               // 分段数
    int grain;              // 每两个控制点之间的插值点数目
    int grain_total;        // 插值点的总个数
    float tension;          // 通过控制点位置的曲线平滑程度

    // 小车及动画相关参数
    QLabel *car;
    QPropertyAnimation *ani;
    QTimer *timer;
    QImage *img;

    float speed;
    float angle;
```

其中，State 是一个自定义的 enum 数据结构，包括了几种用来表示绘制状态的量：START 为初始，DRAW 为绘制曲线，POINT 为绘制插值点，PLAY 为播放动画，PAUSE 为动画暂停，STOP 为动画结束。

```
// 绘制模式与状态判定
enum State {START, DRAW, POINT, PLAY, PAUSE, STOP};
```

3. 绘制相关函数

- void Widget::*mousePressEvent*(QMouseEvent *event)，对鼠标点击事件作出响应

```
void Widget::mousePressEvent(QMouseEvent *event)
{
    if(event->pos().x() < size().width()&& event->pos().x() > 0
            && event->pos().y() < (size().height()-250) && event->pos().y() > 0)
    // 框定合适的绘制范围（区域）
    {
        p[n] = event->pos();
        n++;
        part++;
    }
    if(n >= 2)
        ui->draw_pushbutton->setEnabled(true);      // 允许绘制曲线
    update();
}
```

- void Widget::*paintEvent*(QPaintEvent *)，绘制曲线与插值点。根据状态不同，绘制不同的图形。DRAW 状态下，根据样条的插值点之间用直线相连，调用 drawLine 函数；POINT 状态下，在插值点出用 drawEllipse 函数。

```
void Widget::paintEvent(QPaintEvent *)
{
    QPainter paint(&pix);
    QPainter painter(this);
    paint.setRenderHint(QPainter::Antialiasing);    // 反走样功能开启

    // 设置控制点及其连线
    if(state == START && n != 0)
    {
        if(n == 1)
        {
            paint.setPen(QPen(Qt::black, 1, Qt::SolidLine, Qt::RoundCap));  // 黑色画控制点
            paint.drawEllipse(p[0],2,2);
        }
        else
        {
            paint.setPen(QPen(QColor(10,250,250), 0.5, Qt::SolidLine, Qt::RoundCap));   // 淡蓝色画模拟直线
            paint.drawLine(p[n-2], p[n-1]);
            paint.setPen(QPen(Qt::black, 1, Qt::SolidLine, Qt::RoundCap));
            paint.drawEllipse(p[n-1], 2, 2);
        }
    }
    else if(state == DRAW)
    {   // 绘制曲线——插值点之间用直线相连
        for(int i = 0; i < part * grain; i++)
        {
            paint.setPen(QPen(QColor(250, 0, 0), 1, Qt::SolidLine, Qt::RoundCap)); // 红色画样条曲线
            paint.drawLine(mSpline->spline[i], mSpline->spline[i+1]);
        }
    }
    else if(state == POINT)
    {   // 显示插值点
        for(int i = 0; i < part * grain; i++)
        {
            paint.setBrush(QColor(10, 200, 10)); // 绿色画插值点
            paint.drawEllipse(mSpline->spline[i], 1, 1);
        }
    }
    painter.drawPixmap(0, 0, pix);
}
```

4. 槽函数

- void Widget::draw()，绘制曲线。直接调用 Spline 类的构造函数，传递给 widget 类中的 mSpline 成员。

```
void Widget::draw()
{
    state = DRAW;   // 绘制曲线模式
    grain = ui->grain_spinBox->value();
    tension = ui->tension_doubleSpinBox->value();
    if(n > 1)
    {
        // 生成Cardinal曲线
        mSpline = new Spline(p, n, grain, tension);

        // 允许显示插值点和播放动画
        ui->point_pushButton->setEnabled(true);
        ui->play_pushButton->setEnabled(true);

        update();
    }
}
```

- void Widget::clear()，清空画布

```
void Widget::clear()
{
    // 全部初始化
    state = START;
    n = 0;
    part = -1;
    grain = 0;
    tension = 0;

    pix.fill(Qt::white);

    ui->draw_pushButton->setEnabled(false);
    ui->point_pushButton->setEnabled(false);
    ui->play_pushButton->setEnabled(false);
    ui->pause_pushButton->setEnabled(false);
    ui->speed_horizontalSlider->setValue(7500);
    ui->grain_spinBox->setValue(5);
    ui->tension_doubleSpinBox->setValue(0);

    car->setVisible(false);
    update();
}
```

- void Widget::show_point()，显示插值点

```
void Widget::show_point()
{
    if(state != POINT)
    {
        state = POINT;  // 显示插值点模式
    }
    update();
}
```

- void Widget::play()，播放动画。这里调用了 QPropertyAnimation 类，创建了 ani 成员变量。通过一个 QTimer 计时器 timer 来进行即时控制。只需要自定义帧动画的时长、初始帧、关键帧（插值点处）以及结束帧即可。

```
void Widget::play()
{
    state = PLAY;       // 动画播放模式
    grain = ui->grain_spinBox->value();
    tension = ui->tension_doubleSpinBox->value();
    speed = 15000 - ui->speed_horizontalSlider->value();

    ui->pause_pushButton->setText(QString::fromUtf8("暂停"));
    ui->pause_pushButton->setEnabled(true);     // 允许暂停

    grain_total = part * grain;
    timer->setInterval(speed / grain_total);
    double u = 1.0f / grain_total;

    // 设置动画时长，初始帧，关键帧，结束帧
    // 动画的帧沿着插值点分布即可
    ani = new QPropertyAnimation(car, "pos");
    ani->setDuration(speed);
    ani->setStartValue(QPoint(mSpline->spline[0].x() - 40, mSpline->spline[0].y() - 75));
    for(int i = 1; i < grain_total; i++)
    {
        ani->setKeyValueAt(u*i, QPoint(mSpline->spline[i].x() - 40, mSpline->spline[i].y() - 75));
    }
    ani->setEndValue(QPoint(mSpline->spline[grain_total].x() - 40, mSpline->spline[grain_total].y() - 75));

    timer->start();
    update();

}
```

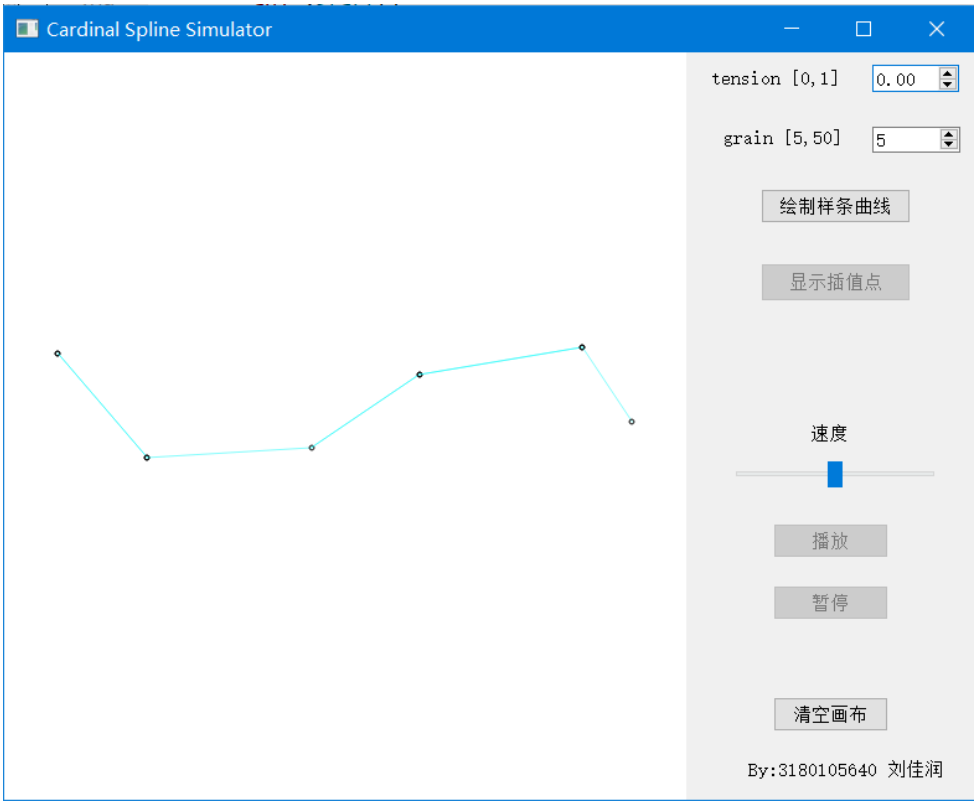
- void Widget::pause_resume()，暂停与恢复动画

```
void Widget::pause_resume()
{
    if(state == PLAY)        // 播放模式—暂停
    {
        ani->pause();
        state = PAUSE;
        ui->pause_pushButton->setText(QString::fromUtf8("继续"));
    }
    else if(state == PAUSE) // 暂停模式—恢复播放
    {
        ani->resume();
        state = PLAY;
        ui->pause_pushButton->setText(QString::fromUtf8("暂停"));
    }

}
```

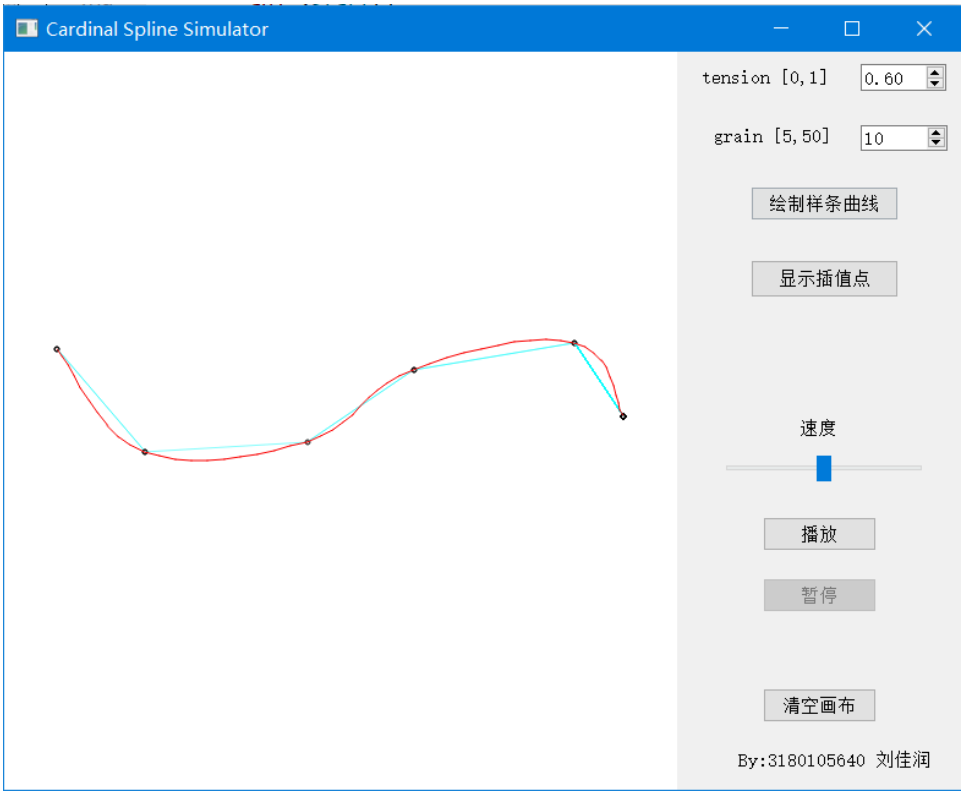
- void Widget::ani_control()，动画控制。这个函数作为 slot 传递给 timer。我们需要在初始和结束的时候对动画进行控制，同时在每个关键帧调整小车图像的角度。具体方法是在每个插值点出带入插值点的纵横坐标之差，根据 atan() 函数反求切线角度，并根据该角度得到一个旋转矩阵，将该矩阵映射到小车的图像上。

```
void Widget::ani_control()
{
    int t = ani->currentTime();
    if(t == 0)
        ani->start();
    if(t > 0)
    {
        int i = round(t / speed * grain_total);
        // 计算斜率和角度
        // 分别带入插值点的纵横坐标之差，得到tan，反求切线角度
        // float atan2(float y, float x)
        // 该函数可以对任何象限的角度求解
        if(i == 0)
            angle = atan(tension*(2*mSpline->spline[1].y() - mSpline->spline[0].y() - mSpline->spline[2].y())/
                    (2*mSpline->spline[1].x() - mSpline->spline[0].x() - mSpline->spline[2].x()))/ PI * 180;
        else if(i == grain_total)
            angle = atan(tension*(2*mSpline->spline[n-1].y() - mSpline->spline[n].y() - mSpline->spline[n-2].y())/
                    (2*mSpline->spline[n-1].x() - mSpline->spline[n].x() - mSpline->spline[n-2].x()))/ PI * 180;
        else
            angle = atan(tension*(mSpline->spline[i+1].y() - mSpline->spline[i-1].y())/
                    (mSpline->spline[i+1].x() - mSpline->spline[i-1].x()))/ PI * 180;

        QMatrix trans;
        trans.rotate(angle);

        // 更新小车图片
        QPixmapCache::clear();
        car->setPixmap(QPixmap::fromImage(*img).transformed(trans, Qt::SmoothTransformation));
        car->setVisible(true);
    }
    if(t >= speed - 150)
    {
        car->setPixmap(QPixmap::fromImage(*img));
        timer->stop();
        ani->stop();
        ui->pause_pushButton->setText(QString::fromUtf8("暂停"));
        ui->pause_pushButton->setEnabled(false);
    }
}
```

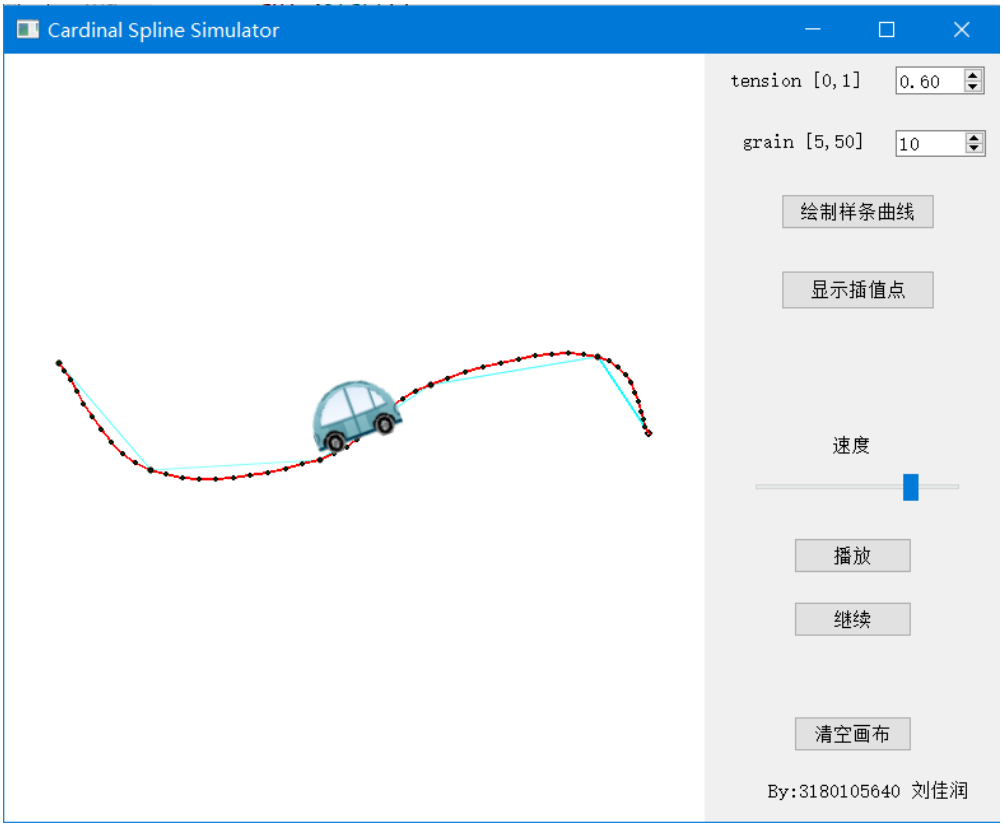
五、实验结果



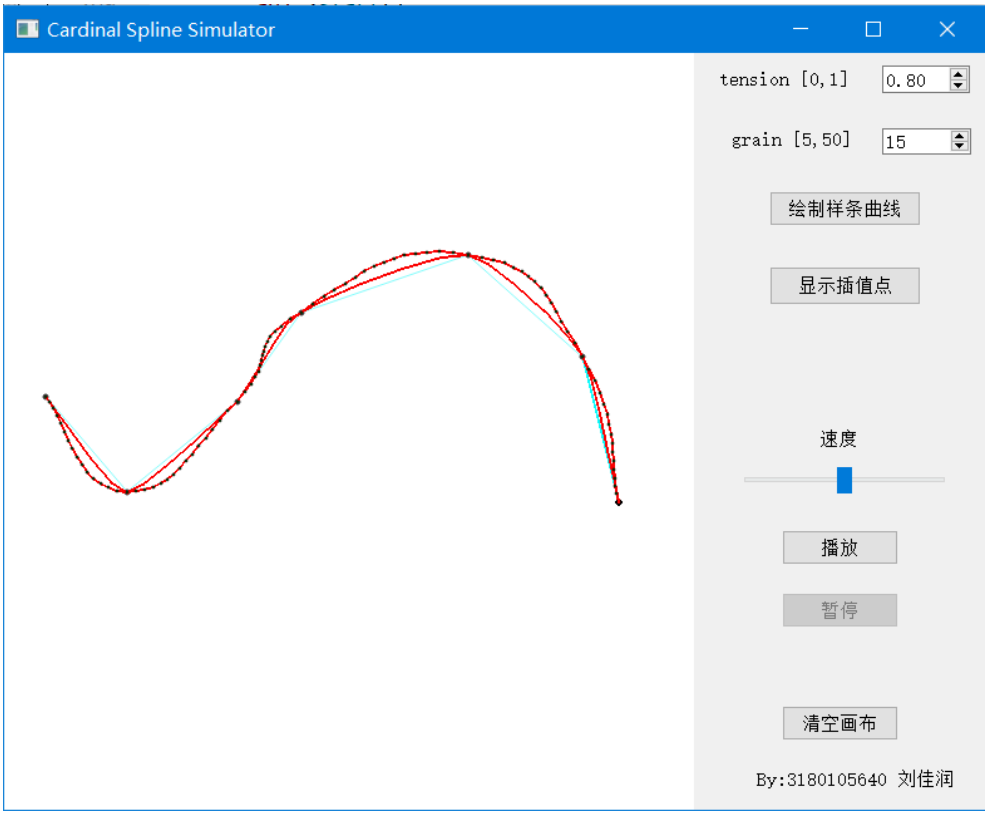
模拟直线的绘制



样条曲线的的生成



插值点显示与动画显示及控制



多条曲线的同时绘制

具体成果演示请参见录屏动画。

六、反思与总结

通过实验，熟悉并了解了 Qt 的编程模式，对于 Qt 的 Widget 类有了更深的了解，掌握了一门新的图形化界面编程的方法。

在编写曲线的绘制函数时参照课件的方法，基本没有遇到什么大的困难。本实验中主要遇到的难点还是在动画的设置与播放处。尤其是在小车运行的过程中需要时刻计算切线角度来旋转小车。最终查阅了一些资料，采用了 QPropertyAnimation 类的帧动画，手动设置起始帧与终止帧，并且在每一个关键帧（插值点）处，调用了 C++数学库中的 atan 函数来计算切线角度，通过矩阵变换的映射来实现小车的

旋转。最终效果还可以。

另外，因为 Qt 中的内存管理机制比较复杂，所以一些地方其实申请开辟出的空间并没有得到很好的回收。这一点是今后编程中需要注意的。尽量少用全局的指针。临时申请的空间可以在赋值之后立刻释放处理掉。