浙江大学

课程名称：计算机动画

姓　　名：猜猜

学　　院：计算机学院

专　　业：数字媒体技术

学　　号：

指导教师：

2019 年 10 月 4 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：猜猜 专业：数字媒体技术 学号：

同组学生姓名：无 指导老师：

实验地点： 实验日期：2019 年 10 月 2 日

一、 实验目的和要求

1. 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。
2. 掌握并实践 Cardinal 样条曲线的数学表示和程序代码的对应关系。
3. 设计并实现一个路径控制曲线，了解动画动态控制的基本原理和方法，提高相关动画编程能力。

二、 实验内容

运用 Cardinal 样条曲线的数学表示和算法，找到其与程序代码的对应关系。设计并实现一个路径控制曲线，体现控制参数对曲线形状的影响，在路径曲线上放置一小汽车，使其在路径上运动起来，汽车速度可调。

三、 实验器材

Windows 10

QT 5.12.0

四、 实验原理与过程分析

1. Cardinal 样条曲线的数学表示与程序代码的对应关系

Cardinal 样条曲线的矩阵表示如下：

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

对于插值点坐标的计算，我们运用 void CSpline::**CubicSpline**(int np, CPT* knots, int grain, float tension)函数，其中四个参数分别为控制点的数量，控制点的坐标，控制点之间的插值点个数和通过控制点位置的曲线平滑程度（即τ）。在此函数中，首先调用 void CSpline::**getCardinalMatrix**(float a1)函数，根据τ计算出矩阵 M，即上图中的中间矩阵；然后将此矩阵和四个点 kml, k0, k1, k2（即 p_{i-1}, p_i, p_{i+1}, p_{i+2}）的坐标代入函数 float CSpline::**Matrix**(float a, float b, float c, float d, float u)中，分别计算出各个插值点的横纵坐标，从而得到插值点的坐标。其对应代码如下：

```
1.      //计算插值点的坐标
2.  void CSpline::CubicSpline(int np, CPT* knots, int grain, float tension)
3.  {
4.      CPT *s, *k0, *kml, *k1, *k2;
5.      int i, j;
6.      float u[50]; //u
7.      getCardinalMatrix(tension); //根据 tension 计算出矩阵 M
8.
9.      //根据分段值确定 u[]，绘制曲线(u[]∈[0,1])
10.     for(i = 0; i < grain; i++){
11.         u[i] = ((float) i)/grain;
12.     }
13.
14.     s =spline;
15.     kml = knots;
16.     k0 = kml + 1;
17.     k1 = k0 + 1;
18.     k2 = k1 + 1;
19.
20.     //求插值点的坐标，一共 np-3 段（除去开始和最后的重合段），每段 grain 个插值点
21.     for(i = 0; i < np-3; i++){
22.         for(j = 0; j < grain; j++){
23.             s->x = Matrix(kml->x, k0->x, k1->x, k2->x, u[j]);
24.             s->y = Matrix(kml->y, k0->y, k1->y, k2->y, u[j]);
25.             s++;
26.         }
```

```
27.       k0++; kml++; k1++; k2++;
28.       }
29.
30.       //末尾的端点
31.       s->x = k0->x;
32.       s->y = k0->y;
33. }
```

函数中调用的两个函数代码如下所示：

①void CSpline::**getCardinalMatrix**(float a1)函数，其中 a1 即τ。

```
1.  //给 Cardinal 矩阵赋值
2.  void CSpline::getCardinalMatrix(float a1)
3.  {
4.      m[0]=-a1;   m[1]=2.-a1;  m[2]=a1-2.;   m[3]=a1;
5.      m[4]=2.*a1; m[5]=a1-3.;  m[6]=3.-2*a1; m[7]=-a1;
6.      m[8]=-a1;   m[9]=0.;     m[10]=a1;     m[11]=0.;
7.      m[12]=0.;   m[13]=1.;    m[14]=0.;     m[15]=0.;
8.  }
```

②float CSpline::**Matrix**(float a, float b, float c, float d, float u)函数，其返回值即插值点的横坐标或纵坐标。

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

2. 建立可视化窗口，通过鼠标点击给定某些控制点，通过这些控制点的位置用代码计算出控制点之间的插值点，可选择显示出样条曲线和插值点。

（1） 窗口的选择

QT 有三大窗口，分别是 QWidget，QMainWindow 和 QDialog。其中 QWidget 类是所有用户界面对象的基类。 窗口部件是用户界面的一个基本单元：它从窗口系统接收鼠

标、键盘和其它事件，并且在屏幕上绘制自己；QMainWindow 类提供一个有菜单条、锚接窗口（例如工具条）和一个状态条的主应用程序窗口。主窗口通常用在提供一个大的中央窗口部件（例如文本编辑或者绘制画布）以及周围菜单、工具条和一个状态条。QMainWindow 常常被继承，因为这使得封装中央部件、菜单和工具条以及窗口状态条变得更容易，当用户点击菜单项或者工具条按钮时，槽会被调用；QDialog 是最普通的顶级窗口，通常情况下，顶级窗口部件是有框架和标题栏的窗口。

由于在本次实验中有涉及到鼠标有关的事件，所以我创建 Widget 类作为应用程序的主界面。

对于窗口的设置：

```
pix = QPixmap(780, 330);
pix.fill(Qt::white); //将画布填充为白色
```

（2） 与绘制曲线和显示插值点有关的成员变量：

```
//画布上的点
float px[100];
float py[100];

//绘制模式
int flag;

bool repaint; //重画基础线条
QPixmap pix; //图像
int n; //控制点
int np; //总的点
int n0; //段数
int grain; //控制点之间的插值平滑点（即控制点之间有多少个插值点）
int count;  //插值点的总个数
float tension; //通过控制点位置的曲线平滑程度
```

其中画布上的所有控制点的横纵坐标被存放在 px[]和 py[]中；flag 表示绘制模式（0 表示初始状态，即不需绘制曲线或显示插值点时的状态；1 表示绘制曲线的状态；2 表示通过显示插值点的状态；3 表示小车运动）；repaint 表示是否需要重画线条，n 表示控制点的数量，np 和 n0 为了表示方便分别表示（n+2）和（n-1），grain 表示控制点之间的插值平滑点，tension 表示通过控制点位置的曲线平滑程度（即τ）。

（3） 与绘制曲线和显示插值点有关的函数：

①函数 void Widget::*mousePressEvent*(QMouseEvent *ev)，可以记录鼠标所点击的位置（即控制点的横纵坐标），并在控制点多于两个时（即可以连接成线时）激活绘制曲线按钮，并在函数最后进行 update，在动画期间。在绝大多数情况下，update 允许 Qt 来优化速度并且防止闪烁，，当 Qt 回到主事件中时，它规划了所要处理的绘制事件。这样允许 Qt 进行优化，从而得到更快的速度和更少的闪烁。

相关代码如下：

```
1.  void Widget::mousePressEvent(QMouseEvent *ev)
2.  {
3.      if((ev->pos().x() <size().width() )&& (ev->pos().x() >0 )
4.              && (ev->pos().y() < (size().height()-190)) &&  (ev->pos().y() >0
    ))
5.      {
6.          px[n] = ev->pos().x();
7.          py[n] = ev->pos().y();
8.          n++;
9.          np++;
10.         n0++;
11.     }
12.     if(n >= 2) ui->draw_pushbutton->setEnabled(true); //激活绘制曲线按钮
13.     update();
14. }
```

②函数 void Widget::*paintEvent*(QPaintEvent *ev)，用来绘制曲线或显示插值点。其函数思路为首先判断是否需要重新绘制控制点之间的连线（当 grain 和τ改变时需要重新绘制）并分为控制点为 1 个（无法连线）和多个（可以连线）两种情况；然后根据模式的不同来进行不同的绘制：当不需要绘制 Cardinal 曲线或显示插值点时直接连接相邻的控制点，显示线段和控制点，当需要绘制 Cardinal 曲线时连接相邻插值点形成曲线，当需要显示插值点时绘制插值点。在该函数中，绘制直线时运用的是 void drawLine()函数，绘制点时运用的是 void drawEllipse()函数，通过调整半径的大小可以调整点的大小，比较明显，而之前查资料查到的 void drawPoint()函数所呈现出来的点并不明显，所以在这里运用了 void drawEllipse()函数；当需要显示插值点时，则将所有的插值点绘制出来。在该函数中，绘制直线所用的颜色为蓝色，绘制 Cardinal 曲线的颜色为紫色，绘制插值点和控制点的颜色为黑色。

与该部分相关的代码如下：

```
1.  void Widget::paintEvent(QPaintEvent *ev)
2.  {
3.      QPainter paint(&pix);
4.      QPainter painter(this);
5.
6.      paint.setRenderHint(QPainter::Antialiasing); //反走样，抗锯齿
7.
8.      //如果需要重新绘制控制点及其连线
9.      if(repaint == true){
10.         if(n ==1){ //如果只有一个点，则不需要连线
11.             paint.setPen(Qt::black); //画点的颜色为黑色
12.             paint.drawEllipse(px[0], py[0], 2, 2);
13.         }
14.         else{
15.             for(int i = 0; i < n-1; i++){
16.                 paint.setPen(QPen(QColor(30,200,250), 1, Qt::SolidLine, Qt::
    RoundCap)); //画直线的颜色为蓝色
17.                 paint.drawLine(px[i],py[i],px[i+1],py[i+1]);
18.                 paint.setPen(Qt::black); //画点的颜色为黑色
19.                 paint.drawEllipse(px[i], py[i], 2, 2);
20.             }
21.             paint.drawEllipse(px[n-1], py[n-1], 2, 2);
22.         }
23.     }
24.
25.     //设置控制点及其连线
26.     if(flag == 0 && n != 0){
27.         if(n == 1){
28.             paint.setPen(QPen(QColor(30,200,250), 1, Qt::SolidLine, Qt::Roun
    dCap)); //画直线的颜色为蓝色
29.             paint.drawLine(px[0], py[0], px[0], py[0]);
30.             paint.setPen(Qt::black); //画点的颜色为黑色
31.             paint.drawEllipse(px[0], py[0], 2, 2);
32.         }
33.         else{
34.             paint.setPen(QPen(QColor(30,200,250), 1, Qt::SolidLine, Qt::Roun
    dCap)); //画直线的颜色为蓝色
35.             paint.drawLine(px[n-2], py[n-2], px[n-1], py[n-1]);
36.             paint.setPen(Qt::black); //画点的颜色为黑色
37.             paint.drawEllipse(px[n-1], py[n-1], 2, 2);
38.         }
39.     }
40.     else if(flag == 1){ //绘制曲线
41.         for(int j = 0; j < n0 * grain; j++){
```

```
            paint.setPen(QPen(QColor(200, 50, 200), 1, Qt::SolidLine, Qt::RoundCap)); //画 Cardinal 曲线的颜色为紫色
            paint.drawLine(mCSpline->spline[j].x, mCSpline->spline[j].y, mCSpline->spline[j+1].x, mCSpline->spline[j+1].y);
        }
    }
    else if(flag == 2){ //显示插值点
        for(int j = 0; j < n0 * grain; j++){
            paint.setPen(QPen(QColor(30, 200, 30), 1, Qt::SolidLine, Qt::RoundCap)); //画插值点的颜色为绿色
            paint.drawEllipse(mCSpline->spline[j].x, mCSpline->spline[j].y, 2, 2);
        }
    }

    painter.drawPixmap(0, 0, pix);
}
```

③函数 void Widget::**on_draw_pushbutton_clicked**()，用来绘制 Cardinal 曲线。当绘制按钮被按下时，对 flag 进行赋值（赋值为 1），代表进入绘制曲线模式，当控制点个数大于 1 调用 CSpline::**CSpline**(float x[100], float y[100], int n, int grain, float tension)函数生成 Cardinal 曲线，并激活显示插值点按钮和小车运动按钮，最后进行 update。程序自动调用 void Widget::***paintEvent***(QPaintEvent *ev)。

与该部分相关的代码如下：

```
void Widget::on_draw_pushbutton_clicked()
{
    flag = 1; //绘制曲线模式
    grain = ui->grain_spinbox->value();
    tension = ui->tension_doublespinbox->value();
    if(n > 1){

        //生成 Cardinal 曲线
        mCSpline = new CSpline(px, py, n, grain, tension);

        //激活显示插值点按钮和小车运动按钮
        ui->point_pushbutton->setEnabled(true);
        ui->play_pushbutton->setEnabled(true);

        update();
    }
}
```

④函数 void Widget::**on_point_pushbutton_clicked**()，用来显示插值点。当显示插值点按钮被按下时，对 flag 进行赋值（赋值为 2），代表进入显示插值点模式，最后进行 update。程序自动调用 void Widget::***paintEvent***(QPaintEvent *ev)函数。

与该部分有关的函数如下：

```
1.  void Widget::on_point_pushbutton_clicked()
2.  {
3.      flag = 2; //显示插值点模式
4.      update();
5.  }
```

3. 在路径曲线上放置一小汽车，使其在路径上运动起来，汽车速度可调。

（1）与小车运动有关的成员变量

```
1.  //有关小车
2.  QLabel *obj;
3.  QPropertyAnimation *ani;
4.  QTimer *timer;
5.  QImage *img;  //图片
6.  QMovie *mov;  //gif
7.  bool is_image = true;
8.
9.  float speed;
10. float angle;
```

其中 is_image 来表示用的小车图片是静态 image 还是动态的 gif，初始默认用小车的图片是静态 image；speed 表示小车的速度；angle 表示曲线的斜率、角度。

（2）与小车图片选用有关的设置

首先设置 obj 空间来承载图片，然后分别设置 image 图片和 gif 图片。

与该部分有关的代码如下：

```
1.  //设置 obj 控件来承载图片
2.  obj = new QLabel(this);
3.  obj->setFixedSize(100,100);
4.  obj->hide(); //隐藏小车的 label
5.
6.  //图片
7.  QImage * ori_img= new QImage(":/image/car.png");
8.  img = new QImage();
9.  if(img == nullptr) printf("Image is null! \n");
10. *img= ori_img->scaled(75,60,Qt::IgnoreAspectRatio);
```

```
11.
12. //gif
13. mov= new QMovie();
14. mov->setFileName(":/image/car.gif");
15. if(mov == NULL) printf("Gif is null! \n");
16. mov->setScaledSize((QSize(75,50)));
17. mov->start();
```

（3）与小车运动有关的函数

①函数 void Widget::**on_play_pushbutton_clicked**()，当按下”Play”按钮时触发使小车运动。函数首先将目前模式设置为 flag = 3，实际上该变量并没有在整个程序中起到作用，但是为了保持与前面的一致性还是添上了。接下来函数判断选择 image 进行运动还是选择 gif 进行运动，接下来启用定时器，每过一定的时间会调整小车的角度。设置动画时间，初始帧、关键帧和结束帧，最后开始计时，进行 update()。

与该部分有关的代码如下：

```
1.  void Widget::on_play_pushbutton_clicked()
2.  {
3.      flag = 3; // 小车运动模式
4.      grain = ui->grain_spinbox->value();
5.      tension = ui->tension_doublespinbox->value();
6.      speed = ui->speed_slider->value();
7.
8.      if(ui->image_radiobutton->isChecked() == true){ //选择 image 运动
9.          QPixmapCache::clear();
10.         obj->setPixmap(QPixmap::fromImage(*img));
11.         is_image = true;
12.     }
13.     else if(ui->gif_radiobutton->isChecked() == true){ //选择 gif 运动
14.         QPixmapCache::clear();
15.         obj->setMovie(mov);
16.         is_image = false;
17.     }
18.
19.     //每过一定的时长调整小车的角度
20.     count = n0 * grain;
21.     timer->setInterval((15000 - speed)/ count);
22.     double u = 1.0f / count;
23.
24.     //设置动画时间，初始帧，关键帧，结束帧
25.     ani = new QPropertyAnimation(obj , "pos");
26.     ani->setDuration(15000 - speed); //动画时间
27.     ani->setStartValue(QPoint(mCSpline->spline[0].x-40, mCSpline->spline[0].
    y - 75)); //初始帧
28.     for(int i = 1; i < count; i++){ //关键帧
```

```
29.         ani->setKeyValueAt(u*i, QPoint(mCSpline->spline[i].x-40, mCSpline->s
    pline[i].y - 75));
30.     }
31.     ani->setEndValue(QPoint(mCSpline->spline[count].x-40, mCSpline->spline[c
    ount].y - 75)); //结束帧
32.
33.     timer->start();
34.     update();
35. }
```

②函数 void Widget::**rotate**()用来判断小车的角度，首先启动动画，当时间在运动范围内时，如果选取运动的小车图片为 image，则计算插值点的角度，这里用的函数为 float atan2(float y, float x)。事实上刚开始我选择的函数是 float atan(float x)，但是该函数只能计算出第一象限和第四象限的点的角度，在第二象限和第三象限的角度计算会非常繁琐，于是我并没有采用在与 cspline 有关的程序代码中计算出斜率然后带入这里的方法，而是直接在 rotate()函数中直接带入插值点的纵坐标之差和横坐标之差，更加简便。接下来载入图片即可；如果选取运动的小车图片为 gif，则直接载入 gif 即可。当时间接近停止时，如果选取运动的小车图片为 image，设置其不再改变方向。最后计时停止，动画停止。

与这部分有关的代码如下：

```
1.  void Widget::rotate()
2.  {
3.      int t = ani->currentTime();
4.
5.      if(t == 0) ani->start();
6.      if(t > 0){
7.          if(is_image == true){//image，可以改变角度
8.              int i = round(t/(15000-speed) * count);
9.
10.             // 计算斜率和角度
11.             if(i == 0){
12.                 angle = atan2(tension*(2*mCSpline->spline[1].y-mCSpline->spl
    ine[0].y-mCSpline->spline[2].y),(2*mCSpline->spline[1].x-mCSpline->spline[0]
    .x-mCSpline->spline[2].x))/ M_PI * 180;
13.             }
14.             else if(i == count){
15.                 angle = atan2(tension*(2*mCSpline->spline[n-1].y-mCSpline->s
    pline[n].y-mCSpline->spline[n-2].y),(2*mCSpline->spline[n-1].x-mCSpline->spl
    ine[n].x-mCSpline->spline[n-2].x))/ M_PI * 180;
16.             }
17.             else angle = atan2(tension*(mCSpline->spline[i+1].y-mCSpline->sp
    line[i-1].y),(mCSpline->spline[i+1].x-mCSpline->spline[i-1].x))/ M_PI * 180;


18.
```

```
            QMatrix matrix;
            matrix.rotate(angle);
            QPixmapCache::clear();
            obj->setPixmap(QPixmap::fromImage(*img).transformed(matrix, Qt::
SmoothTransformation));

        }
        else if(is_image == false) obj->setMovie(mov); //gif，无法改变角度
        obj->show();
    }
    if(t >= 15000 - speed - 200){
        if(is_image == true)
            obj->setPixmap(QPixmap::fromImage(*img));
        else if(is_image == false) obj->setMovie(mov);
        timer->stop();
        ani->stop();
    }
}
```

4. 其他有关的按钮

（1）函数 void Widget::**on_clear_pushbutton_clicked**()，当”Clear”按钮被按下时，将所有有关变量回归初始状态，进行清除画布操作。

有关代码如下：

```
void Widget::on_clear_pushbutton_clicked()
{
    flag = 0;
    n = 0;
    np = 2;
    n0 = -1;
    grain = 0;
    tension = 0;
    repaint = false;

    pix.fill(Qt::white);
    ui->point_pushbutton->setEnabled(false);
    ui->play_pushbutton->setEnabled(false);

    ui->speed_slider->setValue(7500);
    ui->grain_spinbox->setValue(5);
    ui->tension_doublespinbox->setValue(0);
    obj->hide(); //隐藏小车的 label
```

```
19.
20.     update();
21. }
```

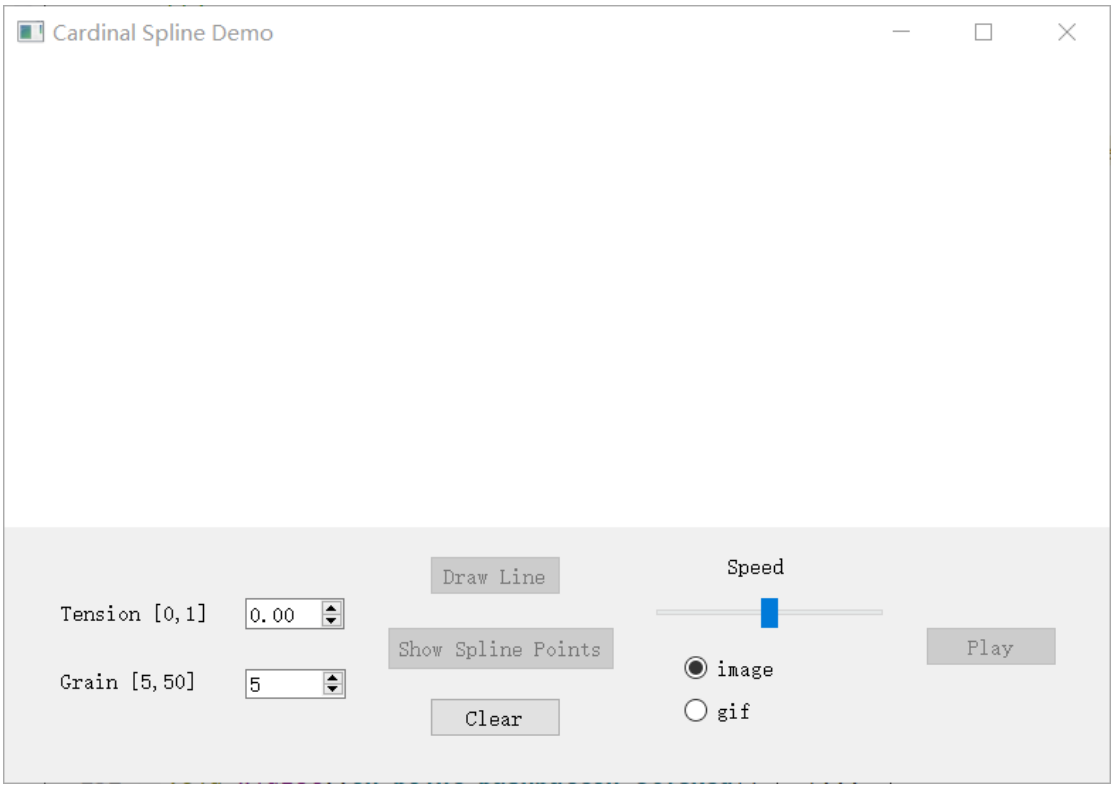
（2）函数 void Widget::**on_grain_spinbox_valueChanged**() 和 void Widget::**on_tension_doublespinbox_valueChanged**()，当 grain 和τ分别变化时，repaint 设置为 true，并触发 void Widget::**on_draw_pushbutton_clicked**()函数，实时显示 Cardinal 曲线的形态。

相关代码如下所示：

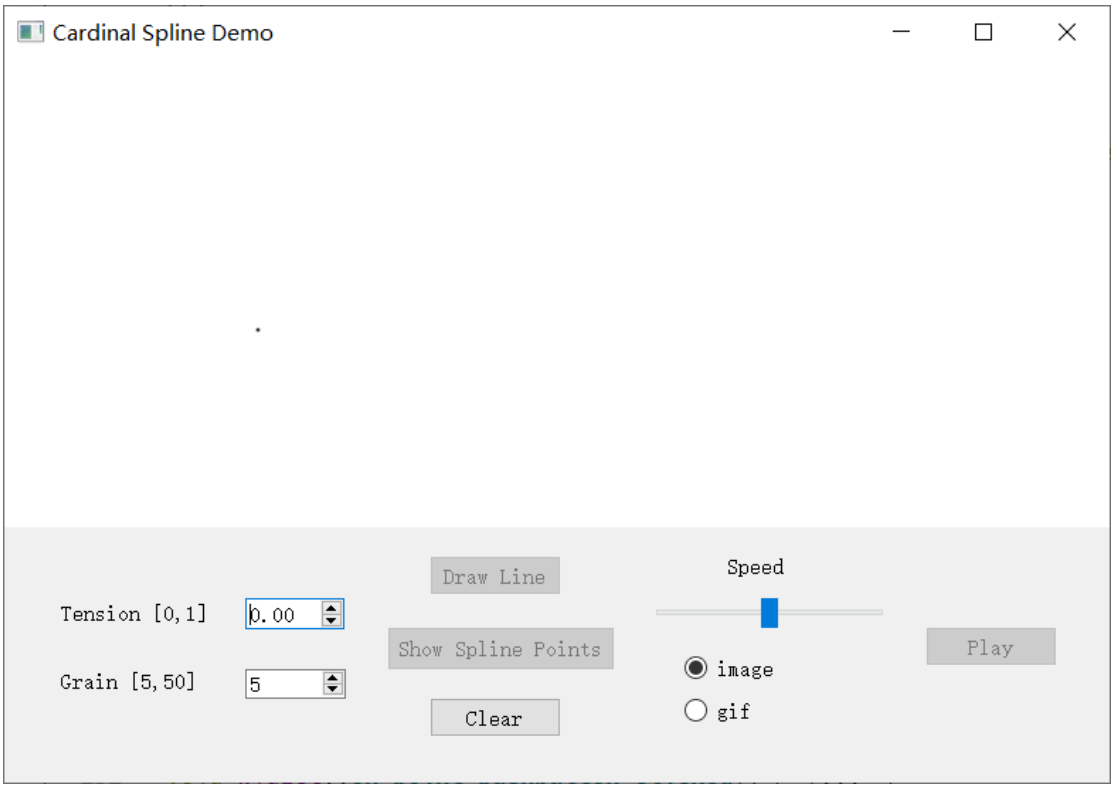
```
1.  void Widget::on_grain_spinbox_valueChanged()
2.  {
3.      pix.fill(Qt::white);
4.      repaint = true;
5.      ui->draw_pushbutton->clicked(true);
6.  }
7.
8.  void Widget::on_tension_doublespinbox_valueChanged()
9.  {
10.     pix.fill(Qt::white);
11.     repaint = true;
12.     ui->draw_pushbutton->clicked(true);
13. }
```

五、 实验结果

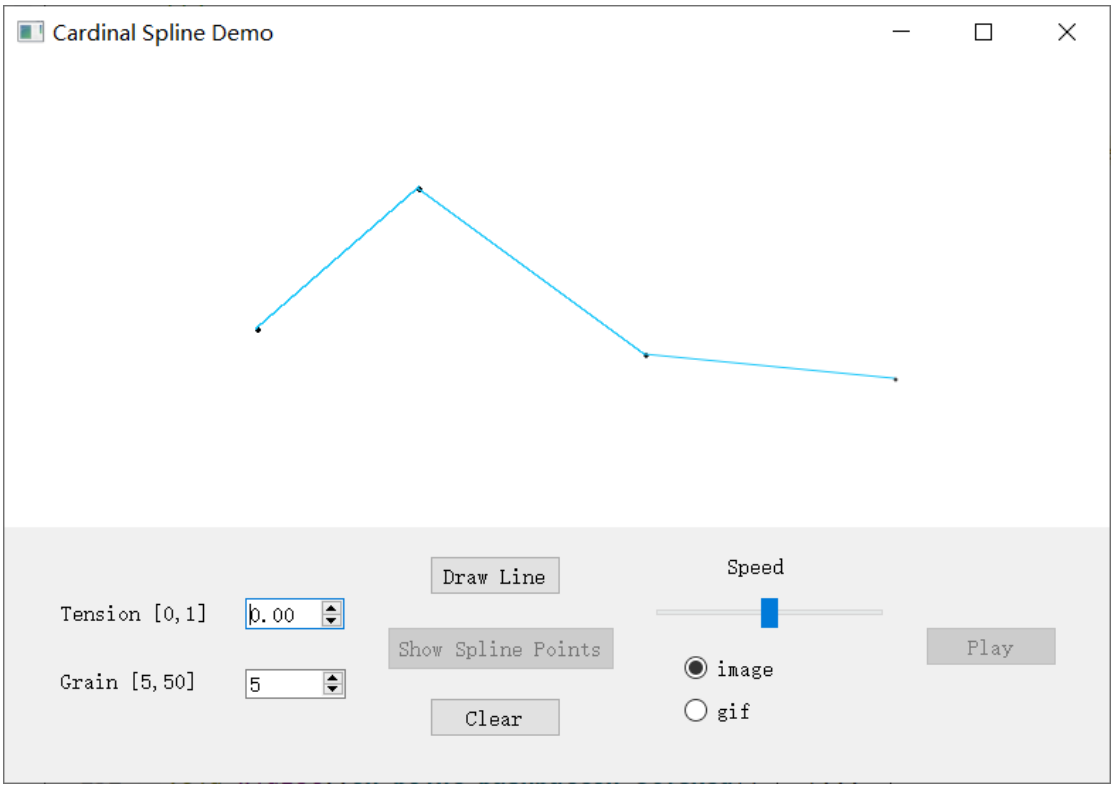
1.初始界面：



2.点一个点

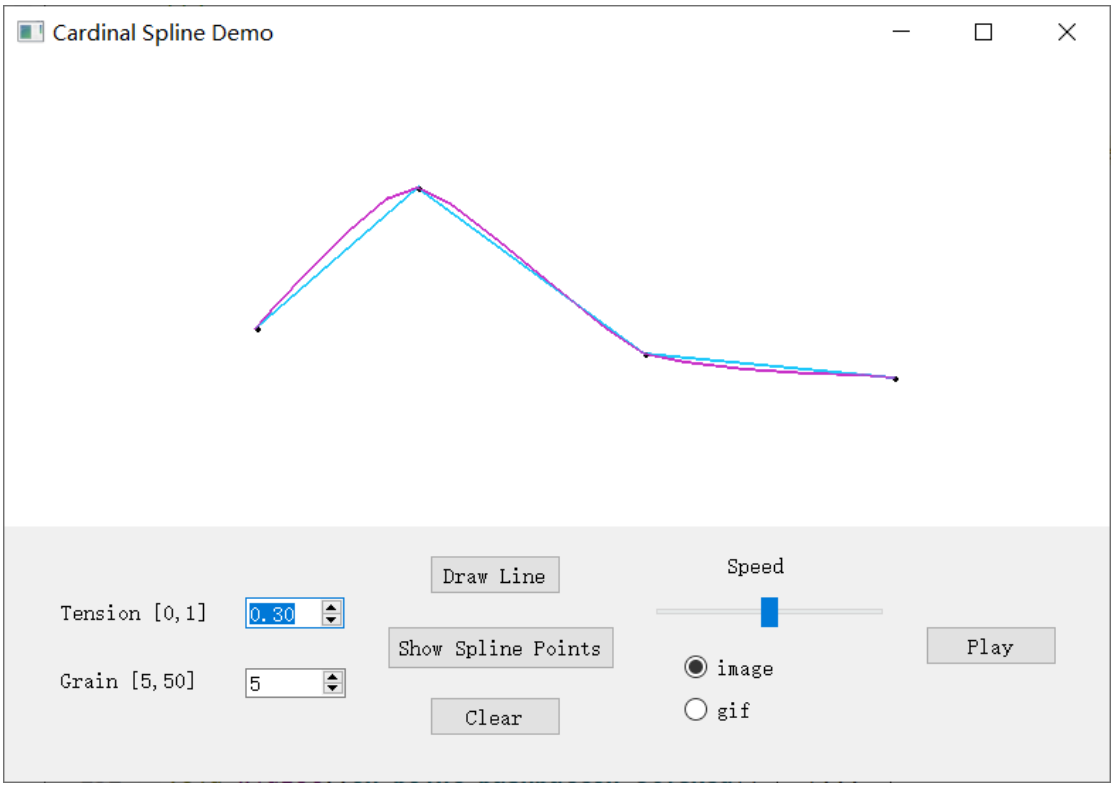


3.点两个点及以上

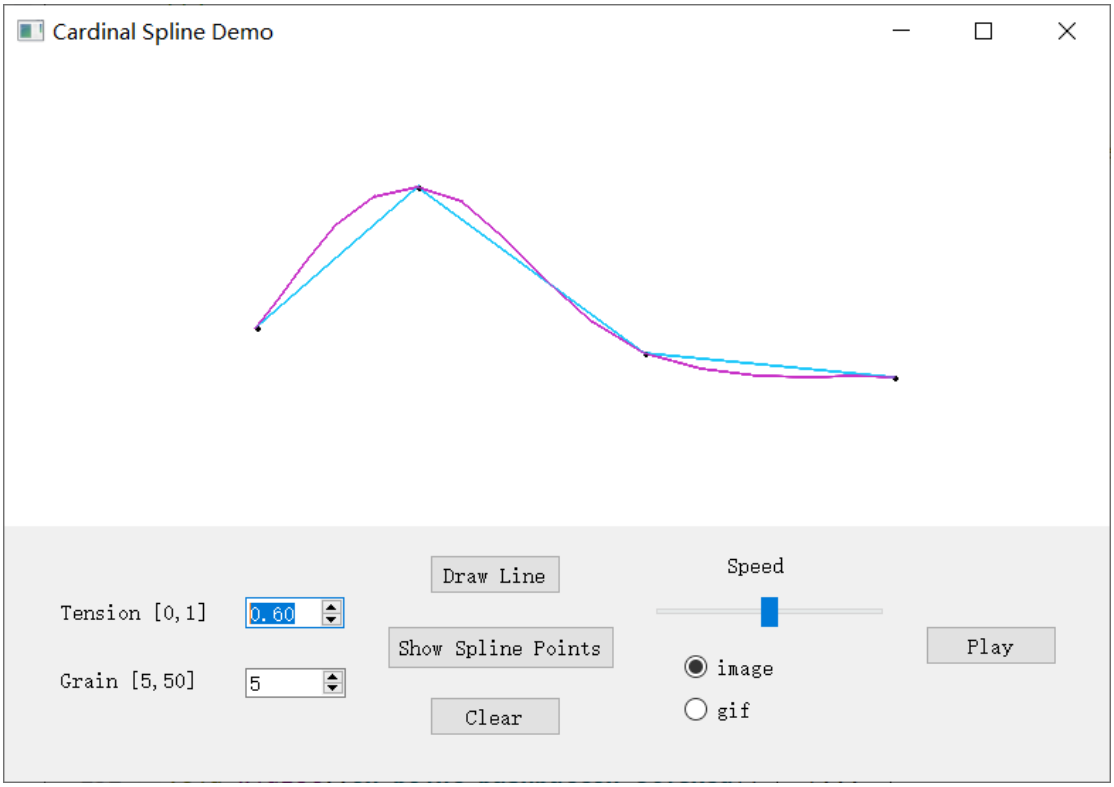


4.改变 tension 值

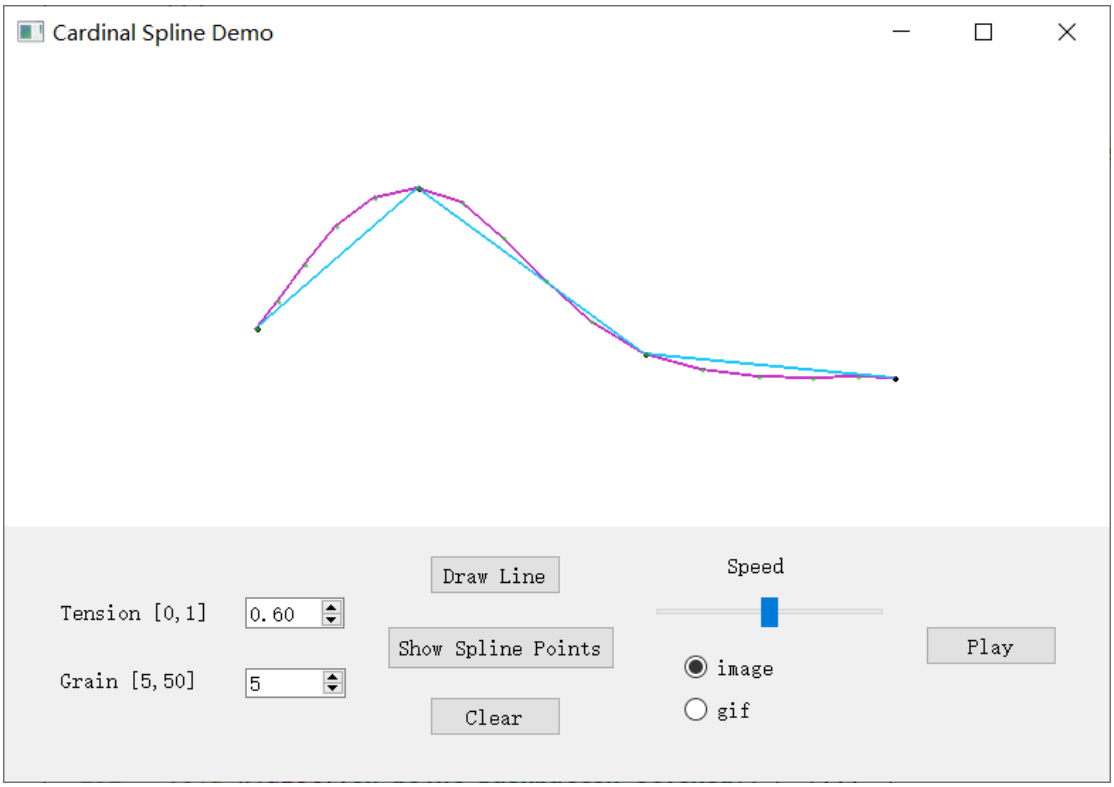
（1）tension = 0.30



（2）tension = 0.60

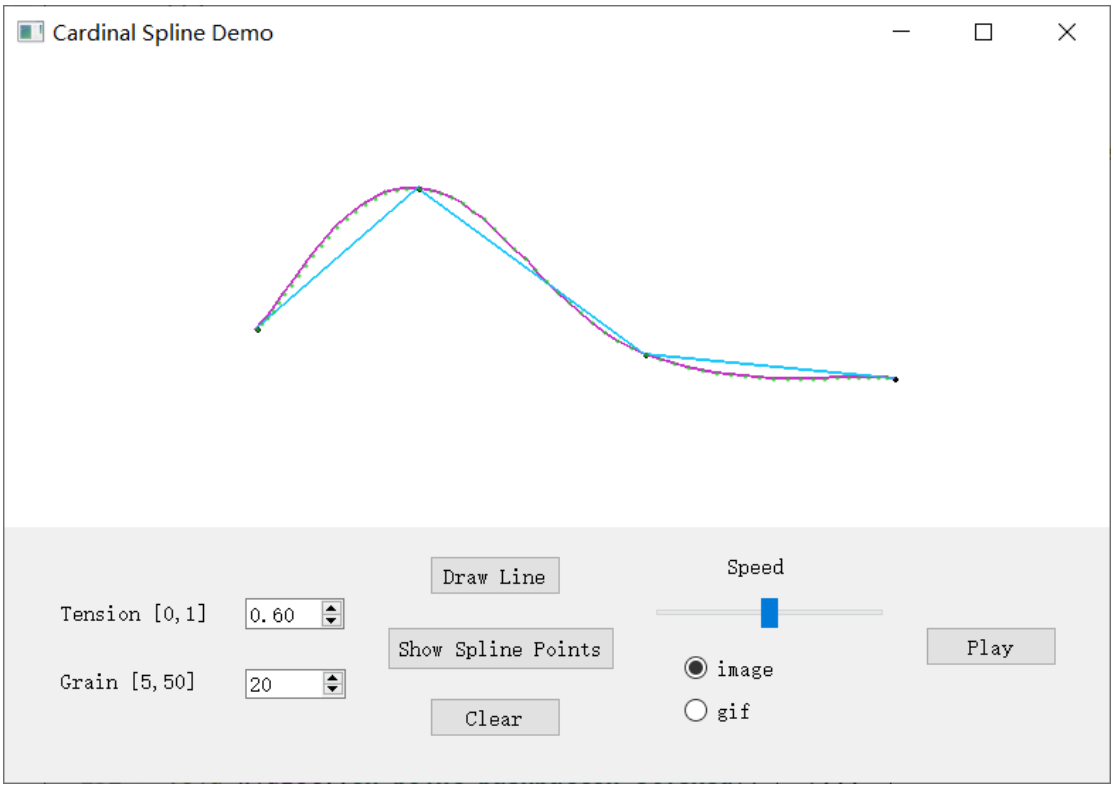


5.显示插值点（插值点数量为 5）

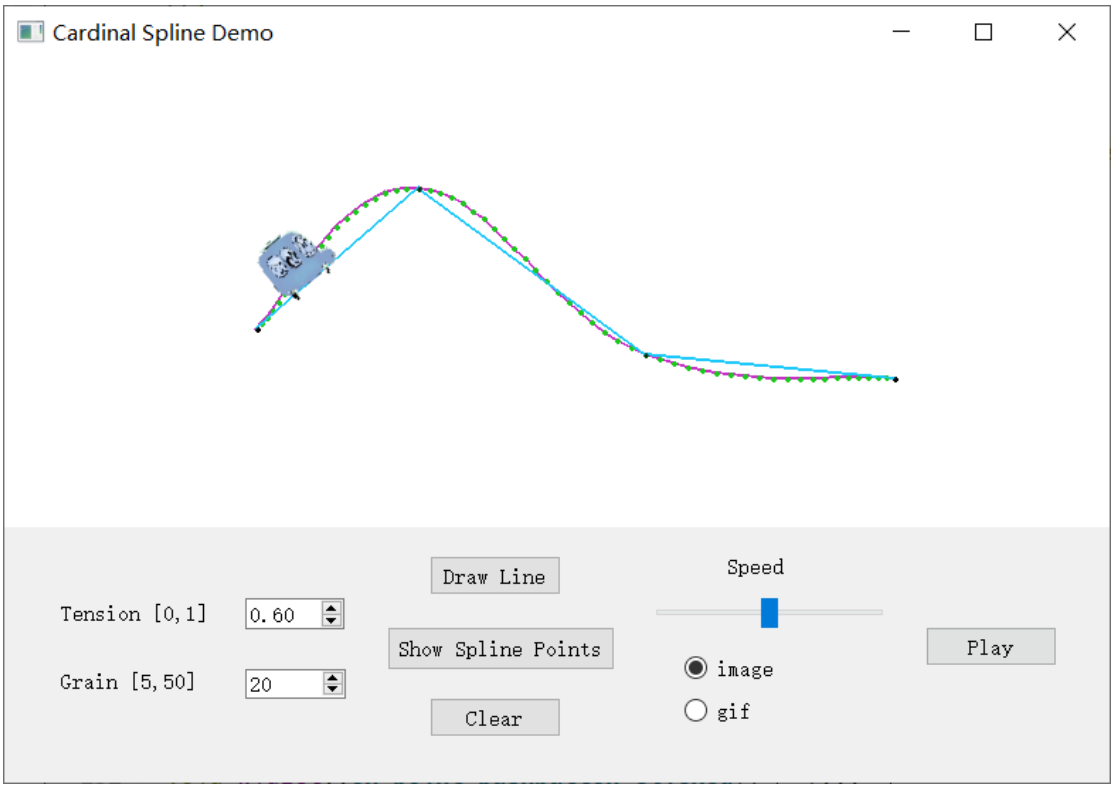


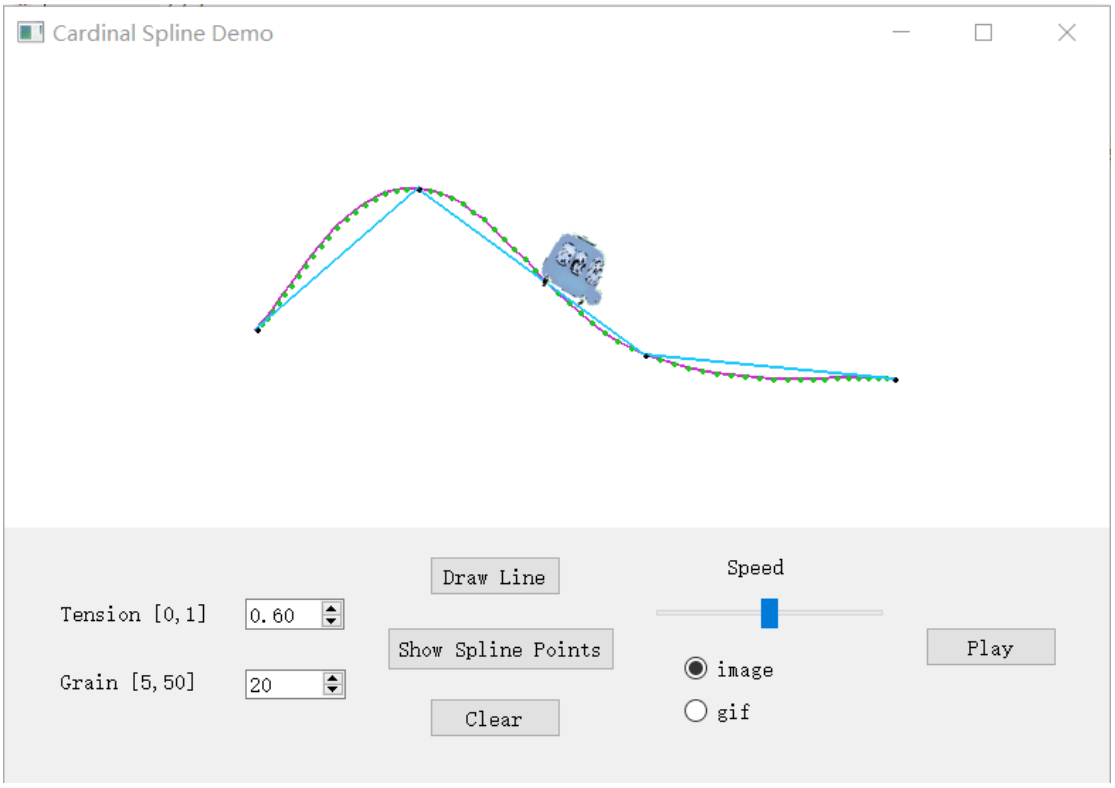
6.改变插值点数量

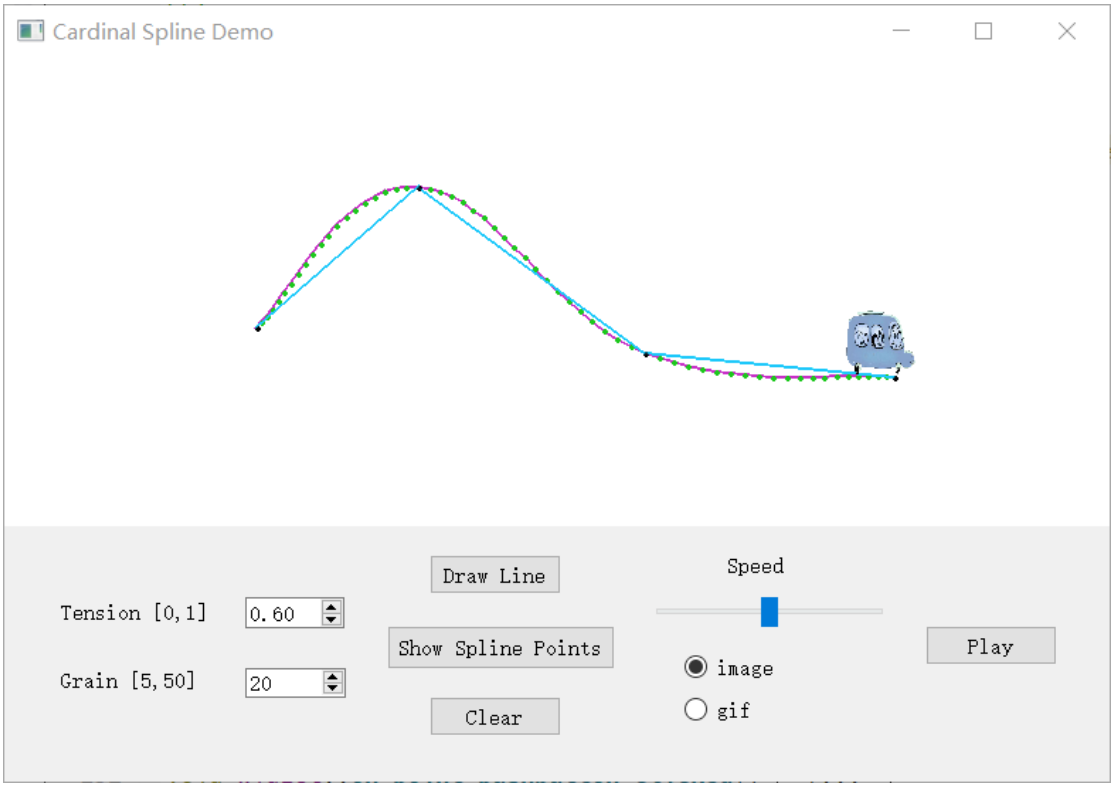
插值点数量变为 20，可以看出在控制点之间的插值点变多变密集



7.小车运动（选取 image）

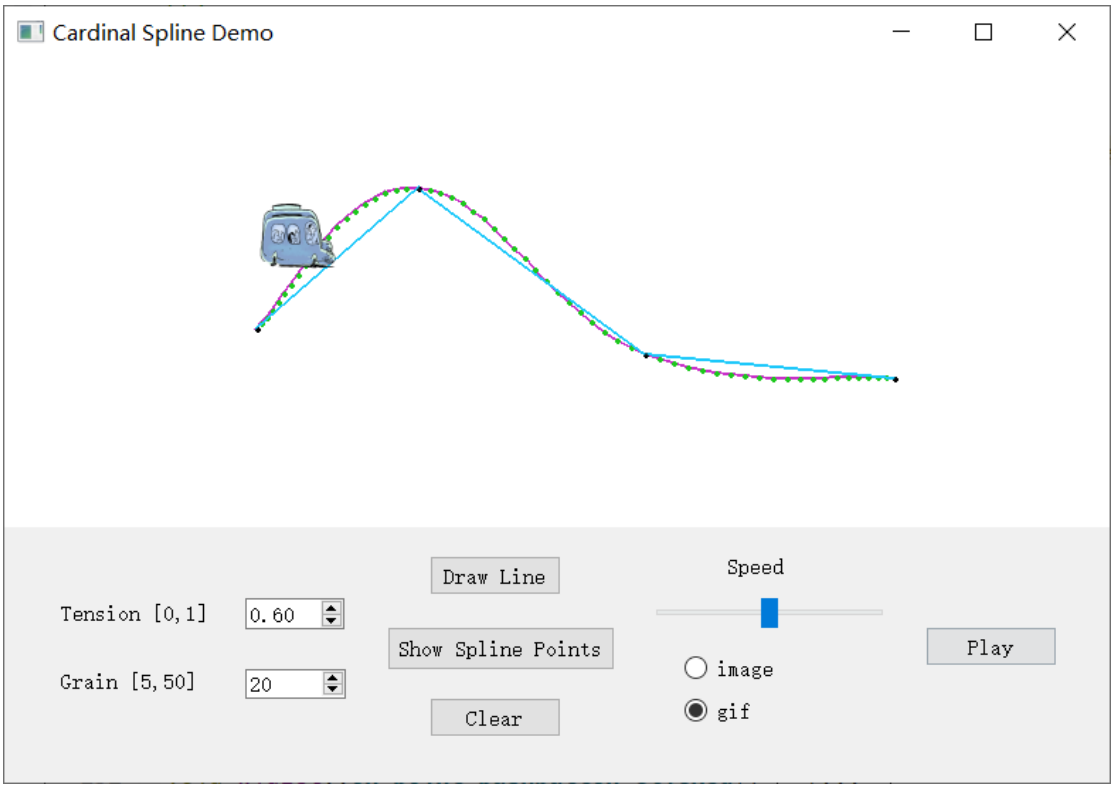


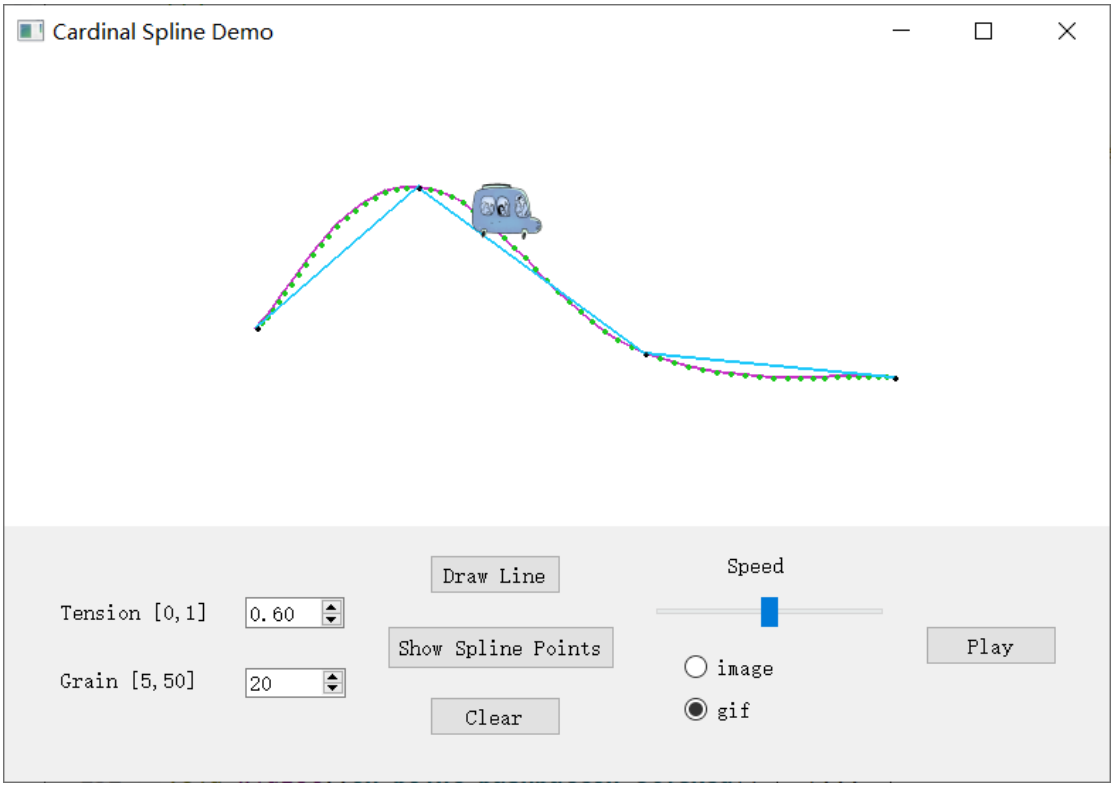


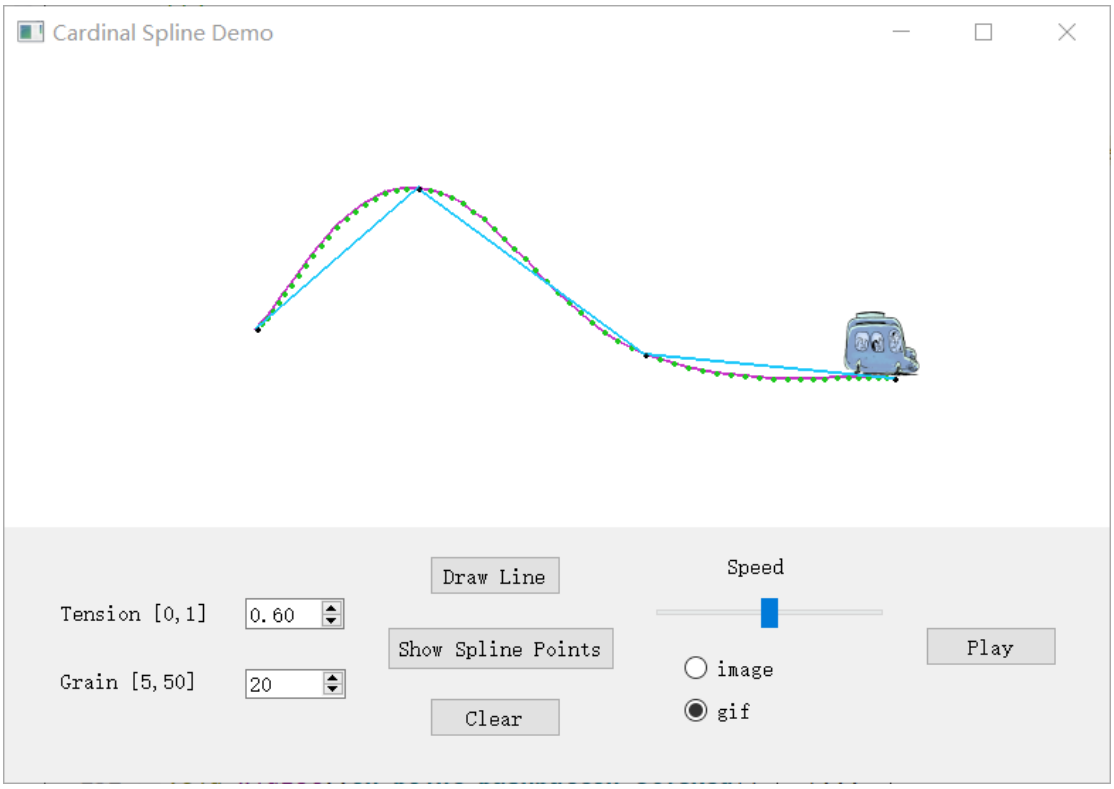


8.小车运动（选取 gif）

由于 gif 在截图中不太明显，详见 video.mp4


Cardinal Spline Demo
Tension [0,1]
0.60
Grain [5,50]
20
Draw Line
Show Spline Points
Clear
Speed
image
gif
Play


Cardinal Spline Demo
Tension [0,1]
0.60
Grain [5,50]
20
Draw Line
Show Spline Points
Clear
Speed
image
gif
Play



9.改变速度

小车改变速度在截图中难以显示，详见 video.mp4。

由以上结果可以看到，本次实验实现了实验的目的和要求，达到了比较不错的结果。

六、 实验感想及分析

在本次实验的过程中，通过实现路径控制曲线，我了解动画动态控制的基本原理和方法，掌握了 Cardinal 样条曲线的表示和算法，了解了控制参数对曲线形状的影响，掌握并实践了 Cardinal 样条曲线的数学表示和程序代码的对应关系。同时，在完成本次实验的过程中，通过查找资料和 debug，我学会了很多 QT 的函数及其用法，学会了对于 ui 界面的操作。

在本次实验中，我遇到的困难主要在处理小车运动方面。第一个困难发生在选取小车运动的图片上，一开始不知道如何载入 gif，通过不断的试验和查找资料，最终解决了这个问题。第二个困难发生在改变小车的角度上，由于一开始选取的是 float atan(float x)函数，将斜率代入后，小车的角度经常会发生错误，后来发现这个函数只能代表第一象限和第四象限，对于小车各种角度的判断方式我一度陷入困境，后面通过查找资料发现了 float atan2(float y, float x)函数，于是我不再计算插值点所在处的斜率，直接代入其

纵坐标之差和横坐标之差，结果取得了很不错的效果。

通过本次实验，我感觉到自己对于动画的理解有了进一步的加深，受益匪浅。同时，钻研问题和解决问题是很快乐和很有成就感的，我感觉很有收获。