浙江大学

课程名称：计算机动画

姓　　名：李沛瑶

学　　院：计算机学院

专　　业：数字媒体技术

学　　号：3180101940

指导教师：于金辉

2020　年　　10　月　　3　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：李沛瑶 专业：数字媒体技术 学号：3180101940

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 27 日

一、 实验目的和要求

1. 设计并实现一个路径曲线，，通过不同参数控制曲线状态，并实现对物体沿生成路线运动的控制。
2. 通过上述实验内容，了解动画动态控制的基本原理何方法，提高动画编程能力。

二、 实验内容和原理

1. 选用 Cardinal 曲线表示运动路径，掌握它的表示和算法，了解不同控制参数对曲线形状和状态的影响。
2. 编写代码实现 Cardinal 曲线算法，对照 cardinal 样条曲线的数学表示和程序之间的对应关系。
3. 给定若干关键控制点的位置（这些控制点可以大致描述某个运动路径的形状），用上述程序计算出控制点之间的插值点，显示出样条曲线。
4. 改变曲线弯曲程度的参数 $\tau \in [0, 1]$大小和控制插值点数目的参数 grain，观察曲线形状的变化。
5. 在路径曲线上放置一小汽车，使其在沿生成的 cardinal 曲线运动，汽车速度和加速度可以调节。

三、 实验平台

Qt 5.14.2 @ Windows

四、 实验步骤

1. 首先，对照 Cardinal 样条曲线的数学表达和程序中计算代码的对应关系。

   Cardinal 样条曲线矩阵表示：

$$P(u) = U^T MB$$

其中，u 是幂次最高为 3 的插值变量，且 u∈[0,1]， M 是 Hermite 多项式矩阵，B 是曲线中，用户指定的关键点数据。

其矩阵展开表示。

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

其中，P_{i-1}，P_i，P_{i+1}，P_{i+2}，是用户指定的控制点控制点，参数 τ 控制曲线的弯曲程度。

为了实现 Cardinal 样条曲线计算，创建 spline 类。

```
class spline
{
private:
    double *ax,*bx,*cx,*dx;//P(u)系数
    double *ay,*by,*cy,*dy;//P(u)系数
    double *A,*B,*C,*D,*E;//计算弧长所用系数
    double* matrix[4];//计算矩阵
    double tension;//参数τ
    int num;//关键点个数
    int grain;//每两个关键点之间插值点的个数（含关键点）
    bool create_flag=false;//是否已经为指针分配空间（判断是否需要delete）
    vector<QPoint> all_points;//所有点

public:
    spline();
     //生成CubicSpline曲线
    void set_Spline(vector<QPoint>& vec,int _grain,double _tension);
    //计算生成的三次样条曲线上所有插值点
    void CubicSpline(vector<QPoint>& vec);
    double calc_Total_length();  //计算曲线总长度

    void init_Matrix(); //初始化矩阵
    void init_spline_Coefficient(vector<QPoint>& vec);//计算P(u)系数

    QPoint calc_Interpolation(int i,double u);//计算内部插值点
    point calc_double_Interpolation(int i,double u);//计算内部插值点（坐标为double类型）

    vector<QPoint>& get_all_points();  //返回储存所有插值点的vector

    void init_length_Coefficient(int _num);//初始化长度计算参数
    double f(int i,double u); //f函数
    double simpson(int i,double a,double b);//求样条曲线长度
    double calc_U(double s,int i,double u1,double u2);//根据长度计算参数u的值
    void clear();//清除数据

    ~spline(){}
};
```

其中 set_Spline 函数根据关键点数组 vec、插值点数目_grain、和控制曲线弯曲程度的参数_tension 生成样条曲线所需要的计算数据（如矩阵数值，P(u)系数等），具体算法如下：

```
//构造CubicSpline曲线
void spline::set_Spline(vector<QPoint>& vec,int _grain,double _tension)
{
    //设置参数
    int _num=vec.size();
    this->grain=_grain;
    this->tension=_tension;
    this->num=vec.size();

    //为每段曲线的中间计算值分配空间
    ax=new double[_num];
    bx=new double[_num];
    cx=new double[_num];
    dx=new double[_num];

    ay=new double[_num];
    by=new double[_num];
    cy=new double[_num];
    dy=new double[_num];

    create_flag=true;

    //根据参数τ初始化计算矩阵
    init_Matrix();
    //计算曲线参数
    init_spline_Coefficient(vec);
}
```

其中 init_Matrix 函数计算矩阵中的数值，即为 P(u)公式中的矩阵 M。init_spline_Coefficient 函数计算不同曲线段中的 P(u)多项式系数，即为公式中的 M*B（分 x，y 两个方向计算）。具体算法如下：

```
//初始化计算矩阵
void spline::init_Matrix()
{
    matrix[0][0]=-tension;
    matrix[0][1]=2-tension;
    matrix[0][2]=-2+tension;
    matrix[0][3]=tension;
    matrix[1][0]=2*tension;
    matrix[1][1]=-3+tension;
    matrix[1][2]=3-2*tension;
    matrix[1][3]=-tension;
    matrix[2][0]=-tension;
    matrix[2][1]=0;
    matrix[2][2]=tension;
    matrix[2][3]=0;
    matrix[3][0]=0;
    matrix[3][1]=1;
    matrix[3][2]=0;
    matrix[3][3]=0;
}
```

```
//根据每段的关键点坐标以及计算矩阵计算参数方程的参数ax等
void spline::init_spline_Coefficient(vector<QPoint>& vec)
{
    vector<QPoint> points=vec;
    QPoint p1,p2,p3,p4;

    for(int i=0;i<points.size()-1;i++)
    {
        if(i==0)//第一段（虚拟点）
        {
            p1=points[0];
            p2=points[0];
            p3=points[1];
            p4=points[2];
        }
        else if(i==points.size()-2)
        {
            p1=points[points.size()-3];
            p2=points[points.size()-2];
            p3=points[points.size()-1];
            p4=points[points.size()-1];
        }
        else//最后一段（虚拟点）
        {
            p1=points[i-1];
            p2=points[i];
            p3=points[i+1];
            p4=points[i+2];
        }
        ax[i]=matrix[0][0]*p1.x()+matrix[0][1]*p2.x()+matrix[0][2]*p3.x()+matrix[0][3]*p4.x();
        bx[i]=matrix[1][0]*p1.x()+matrix[1][1]*p2.x()+matrix[1][2]*p3.x()+matrix[1][3]*p4.x();
        cx[i]=matrix[2][0]*p1.x()+matrix[2][1]*p2.x()+matrix[2][2]*p3.x()+matrix[2][3]*p4.x();
        dx[i]=matrix[3][0]*p1.x()+matrix[3][1]*p2.x()+matrix[3][2]*p3.x()+matrix[3][3]*p4.x();
        ay[i]=matrix[0][0]*p1.y()+matrix[0][1]*p2.y()+matrix[0][2]*p3.y()+matrix[0][3]*p4.y();
        by[i]=matrix[1][0]*p1.y()+matrix[1][1]*p2.y()+matrix[1][2]*p3.y()+matrix[1][3]*p4.y();
        cy[i]=matrix[2][0]*p1.y()+matrix[2][1]*p2.y()+matrix[2][2]*p3.y()+matrix[2][3]*p4.y();
        dy[i]=matrix[3][0]*p1.y()+matrix[3][1]*p2.y()+matrix[3][2]*p3.y()+matrix[3][3]*p4.y();
    }
}
```

2. CubicSpline 函数，根据_grain 值生成不同的 u 值，并计算曲线上所有插值点的坐标，储存在名为 all_points 的 vector 中。

```
//计算曲线上所有插值点
void spline::CubicSpline(vector<QPoint>& vec)
{
    //当没清除就再次点击生成曲线时，不清除之前的插值点，插入间隔点
    QPoint temp(0,0);
    all_points.push_back(temp);

    //根据设置的插值点个数参数，计算对应的u值
    int num=vec.size();
    double* u = new double[grain];
    for (int i = 0; i<grain; i++) {
        u[i] = ((double)i) / grain;  //u [0,1]
    }

    //根据u值和曲线参数计算插值点坐标
    for (int i = 0; i<num-1; i++) {
        QPoint p1=vec[i];
        //加入关键点
        all_points.push_back(p1);
        for (int j = 1; j<grain; j++) {
            QPoint temp=calc_Interpolation(i,u[j]);
            all_points.push_back(temp);
        }
    }
    //加入关键点
    QPoint p1=vec[num-1];
    all_points.push_back(p1);

    delete []u;
}
```

其中，calc_Interpolation 函数根据曲线段序号 i，和参数 u 的不同值，计算具体一个插值点的坐标，具体实现如下：

```
//根据曲线段序号i和参数u计算插值点坐标
QPoint spline::calc_Interpolation(int i,double u)    //计算插值点的xy值
{
    QPoint p;

    double x=(dx[i] + u*(cx[i] + u*(bx[i] + u*ax[i])));
    double y=(dy[i] + u*(cy[i] + u*(by[i] + u*ay[i])));

    p.setX(x);
    p.setY(y);

    return p;
}
```

3. 编写 paintWindow 类作为画板，继承自 QWidget 类。在 paintWindow 类中编写鼠标回调函数 mousePressEvent，记录通过鼠标交互选定的关键点。以及绘制函数 paintEvent，在每次 update（）时调用。paintWindow 类定义具体如下：

```
class paintWindow : public QWidget
{
    Q_OBJECT
private:
    spline* sp;
    int grain;  //每个曲线区间有多少个插值点（包括两端关键点）
    double tension;  //参数，控制曲线弯曲程度

    bool ifDrawInpoint=false;  //是否显示插值点

    int time;//时间
    QTimer* timer;  //计时器
    QPixmap* car[5];  //储存小车位图信息

    int car_index;//显示第几种小车
    int pen_index=0;//使用第几种笔刷

    double speed;  //小车速度
    double accelerate;  //小车加速度

    point now_point;  //运动当前点
    point next_point;  //运动下一个点
    double ratio;  //旋转角度

    vector<QPoint> points;//储存所有关键点

    bool endflag=false;  //小车是否运动到曲线末端
public:
    paintWindow();

    void paintEvent(QPaintEvent *);  //绘制函数
    void mousePressEvent(QMouseEvent *e);  //鼠标回调函数

    void create_Spline(int _grain, double _tension);  //生成并显示cubicspline曲线
    void start_Move(double _speed,double _accelarate);  //小车开始运动
    void stop_Move();  //小车暂停运动
    void continue_Move();  //小车继续运动

    int numbers();  //关键点个数
    double total_length();  //曲线总长度
    double now_length();  //小车当前走过的路线长度
    int get_spline_index(double now_len);  //获取小车当前在哪一段曲线
    double get_Ratio();  //获取当前曲线斜率

    void change_car();  //改变小车
    void change_pen(QPainter& paint);  //改变笔刷
    void change_DrawInPoint();  //改变是否绘制插值点的控制变量
    QPixmap* now_car();  //当前小车位图指针

    void clear();
private slots:
    void changeState();  //连接计时器，改变小车坐标，旋转角度等信息
};
```

4. 鼠标回调函数 mousePressEvent，记录通过鼠标交互选定的关键点。绘制函数 paintEvent，根据数据变化，绘制所有的关键点，曲线，已经选择是否绘制插值点。

```
//鼠标回调函数，鼠标点击，加入关键点
void paintWindow::mousePressEvent(QMouseEvent *e)
{
    points.push_back(e->pos());
    update();
}

//绘制函数
void paintWindow::paintEvent(QPaintEvent *)
{
    QPainter paint(this);
    if(numbers()<=0)return;   //没有关键点

    //有关键点且计时器时钟不为零，则绘制小车
    if(numbers()>=1&&time!=0){
        //平移
        paint.translate(now_point.X(),now_point.Y());
        //获取斜率角度并旋转
        ratio=get_Ratio();
        paint.rotate(ratio);
        paint.drawPixmap(-35,-40,60,40,*car[car_index]);
        //复原平移和旋转
        paint.rotate(-ratio);
        paint.translate(-now_point.X(),-now_point.Y());
    }


    //设置笔刷样式，绘制关键点
    paint.setPen(QPen(Qt::black,5,Qt::DashDotLine,Qt::RoundCap));
    for(int i=0;i<numbers();i++)
    {
        paint.drawEllipse(points[i],1,1);
    }
```

```
    //获取储存所有插值点的vector
    vector<QPoint> a_p=sp->get_all_points();
    paint.setPen(QPen(Qt::blue,3,Qt::SolidLine,Qt::RoundCap));
    //如果已经生成了插值点，则在每两个插值点之间绘制线条
    if(a_p.size()>0)
    {
        for(unsigned int i=0;i<a_p.size()-1;i++)
        {
            QPoint p1=a_p[i];
            QPoint p2=a_p[i+1];
            if(((p1.x()==0&&p1.y()==0)||(p2.x()==0&&p2.y()==0)))
            {
                change_pen(paint);
            }
            else
            {
               paint.drawLine(p1,p2);
            }

        }
    }
```

```
    //根据控制变量ifDrawInpoint的值，决定是否绘制出中间插值点
    if(ifDrawInpoint)
    {
        if(a_p.size()>0)
        {
            paint.setPen(QPen(Qt::white,2,Qt::SolidLine,Qt::RoundCap));
            for(unsigned int i=0;i<a_p.size()-1;i++)
            {
                paint.drawEllipse(a_p[i],1,1);
            }
        }
    }

}
```

5. 引入 QTimer 类作为计时器，每隔一段时间调用 changestate 函数，改变小车坐标及旋转角度等。其中 now_point 是当前小车位置，next_point 是下一个小车位置。get_Ratio 函数计算当下

曲线的斜率，以及小车旋转角度。

```
//计时器关联槽函数，改变小车坐标及角度
void paintWindow::changeState()
{
    //endflag为true，停止计时器，时间恢复为0
    if(endflag)
    {
        time=0;
        timer->stop();
        return;
    }

    //time等于0时，先计算now_point
    if(time==0){
        now_point=sp->calc_double_Interpolation(0,0);
    }

    time++;

    //当前长度超过曲线总长度，保证计时器停止时小车在路线末尾
    if(now_length()>=total_length()-1)
    {
        next_point=sp->calc_double_Interpolation(points.size()-2,1);
        now_point=sp->calc_double_Interpolation(points.size()-2,0.99);
        endflag=true;
        update();
        return;
    }

    //通过路线长度计算小车当前所在曲线段
    double len=now_length();
    int spline_index=get_spline_index(now_length());
    for(int i=0;i<spline_index;i++){
        len -= sp->simpson(i,0,1);
    }

    //通过不同曲线段的参数和小车在当前曲线段走过的路程长度计算小车坐标
    if(spline_index<points.size()-1)      ⚠ comparison of integers o
    {
        if(time>1)now_point=next_point;  //将下一个点赋值为当前点
        //计算下一个点的坐标
        double _u=sp->calc_U(len,spline_index,0,1);
        next_point=sp->calc_double_Interpolation(spline_index,_u);
    }

    update();
}
```

```cpp
//获取当前斜率（小车旋转角度）
double paintWindow::get_Ratio()
{
    const float pi = 3.14159;

    double _x0=now_point.X();
    double _y0=now_point.Y();
    double _x1=next_point.X();
    double _y1=next_point.Y();

    if(_x1-_x0==0&&_y1-_y0==0)return ratio;  //两条曲线之间的间隔点
    else if(_x1-_x0==0)return 0;  //如果分母为零

    double tan=(_y1-_y0)/(_x1-_x0);
    double theta=atan(tan);
    double ratio=theta/(2*pi)*360;

    return ratio;
}
```

6. MainWindow 设计及按钮槽函数

MainWindow 窗口设计如下：

MainWindow 类设计如下：

```
class MainWindow : public QMainWindow
{
    Q_OBJECT
private:
    paintWindow* p_w;

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:
    void on_create_clicked();//绘制曲线并显示
    void on_clear_clicked();//清屏
    void on_start_clicked();//开始运动按钮槽函数
    void on_show_clicked();//展示插值点按钮槽函数
    void on_stop_move_clicked();//停止运动按钮槽函数
    void on_continue_move_clicked();//继续运动按钮槽函数


    void update_numbers();//更新关键点数目
    void on_change_clicked();

private:
    Ui::MainWindow *ui;
};
```

各个按钮的槽函数设计如下：

```
//清屏槽函数
void MainWindow::on_clear_clicked()
{
    //paintWindow清屏
    p_w->clear();

    //清除关键点个数和曲线总长度
    int _temp=0;
    QString temp=QString::number(_temp);
    ui->number2->setText(temp);
    ui->length->setText(temp);

    //开始运动按钮不可用
    ui->start->setDisabled(true);
    ui->stop_move->setDisabled(true);
    ui->continue_move->setDisabled(true);
}
```

```
//生成并绘制曲线
void MainWindow::on_create_clicked()
{
    int num=p_w->numbers();
    if(num==0)return;  //没有指定关键点

    QString temp="  "+QString::number(num);
    ui->number2->setText(temp);
    ui->start->setDisabled(false);

    //读取曲线参数
    double _tension=ui->Slider->value();
    _tension=(_tension+1)/100.0;
    int _grain=ui->spinBox_grain->value();

    //画出曲线
    p_w->create_Spline(_grain,_tension);

    //计算曲线总长度并显示
    double total_len=p_w->total_length();
    temp=QString::number(total_len);
    ui->length->setText(temp);

    //刷新
    update();
}
```

```
//小车开始运动
void MainWindow::on_start_clicked()
{
    double _speed=ui->spinBox_speed->value();
    double _accelarate=ui->spinBox_acce->value();
    p_w->start_Move(_speed,_accelarate);

    //设置按钮是否可用
    ui->continue_move->setDisabled(true);
    ui->stop_move->setDisabled(false);
}
```

```
//修改参数，决定是否展示插值点
void MainWindow::on_show_clicked()
{
    p_w->change_DrawInPoint();
}
```

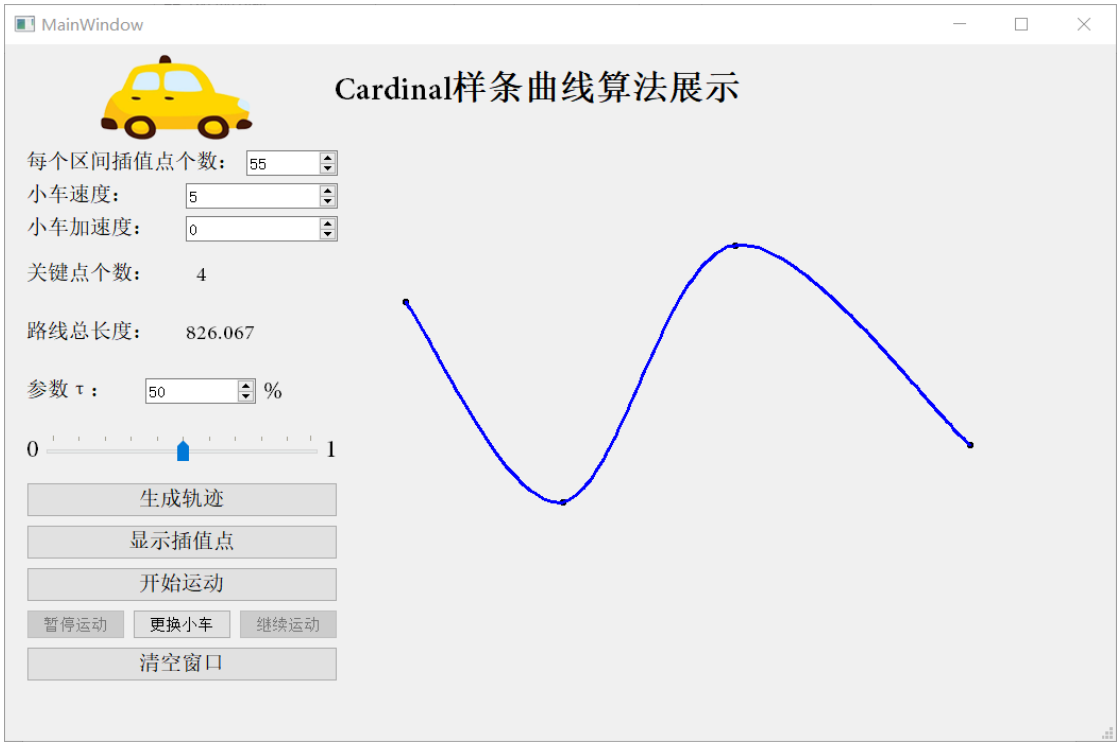
```
//改变控制变量，决定是否绘制插值点
void paintWindow::change_DrawInPoint()
{
    if(ifDrawInpoint)ifDrawInpoint=false;
    else ifDrawInpoint=true;
    update();
}

//小车暂停运动
void MainWindow::on_stop_move_clicked()
{
    p_w->stop_Move();
    ui->continue_move->setDisabled(false);
    ui->stop_move->setDisabled(true);
}

//小车继续运动
void MainWindow::on_continue_move_clicked()
{
    p_w->continue_Move();
    ui->continue_move->setDisabled(true);
    ui->stop_move->setDisabled(false);
}
```

五、 实验结果分析

主界面：

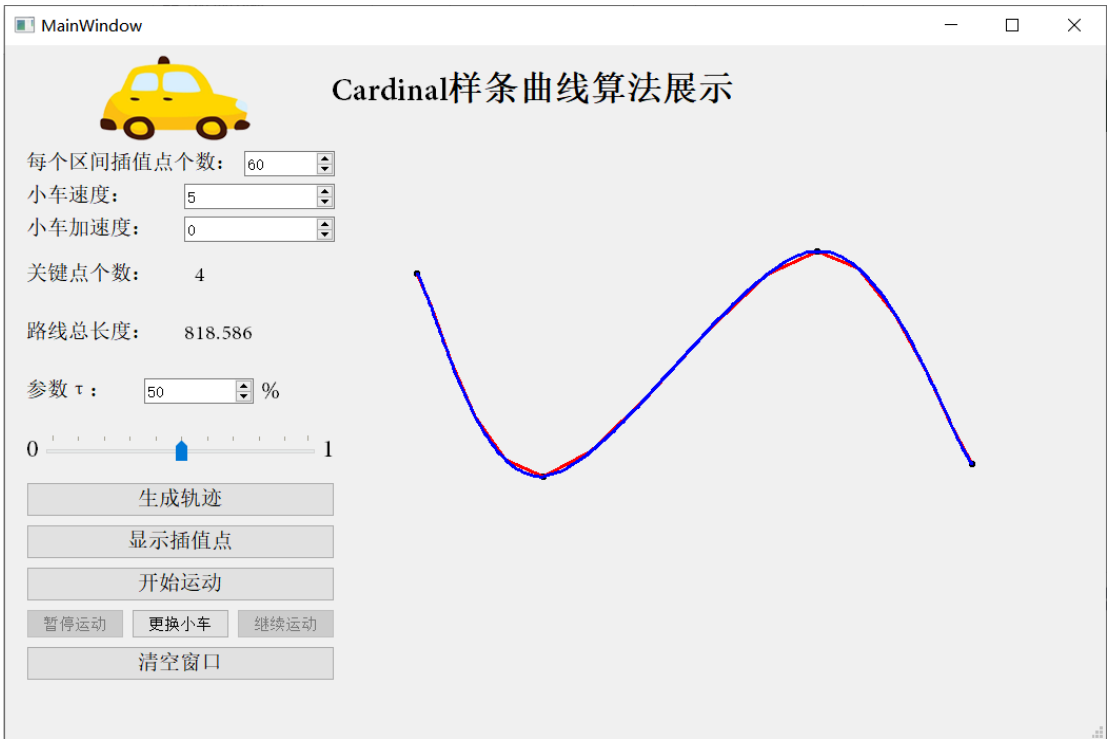
选取控制点：

绘制曲线：同时显示关键点个数和路线总长度。



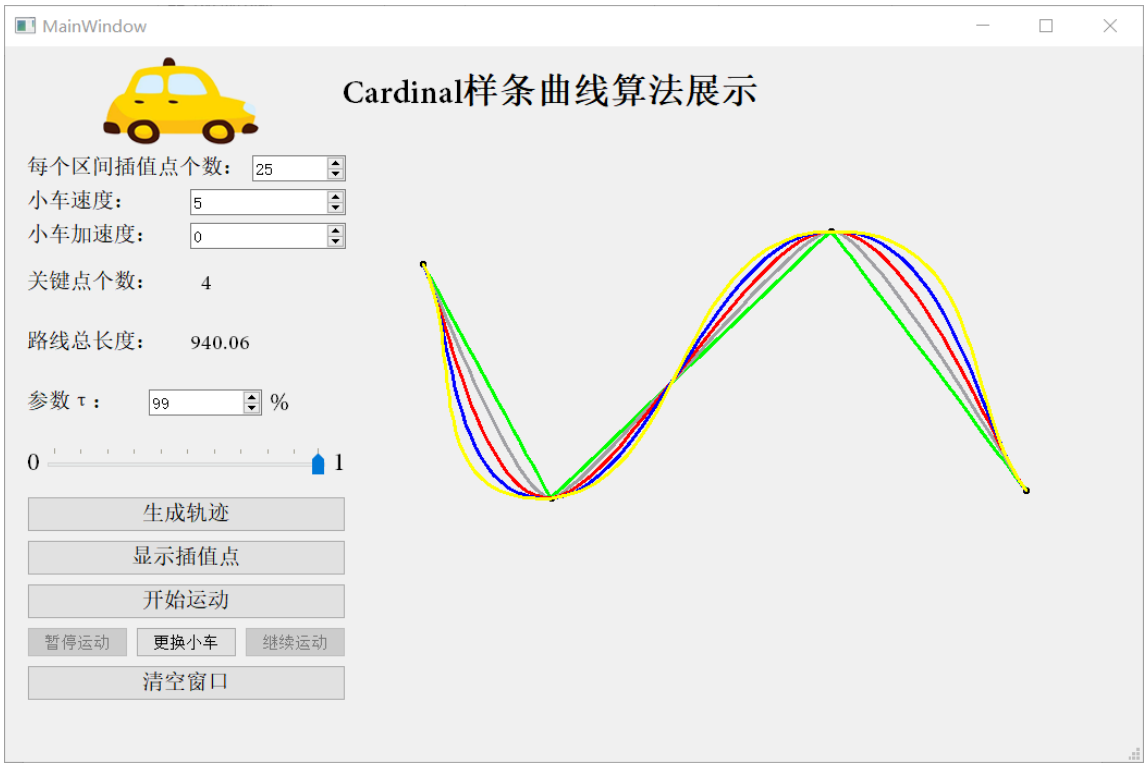
不同参数对曲线值的影响：

1、不同的 grain：grain=5（红色）和 grain=60（蓝色）

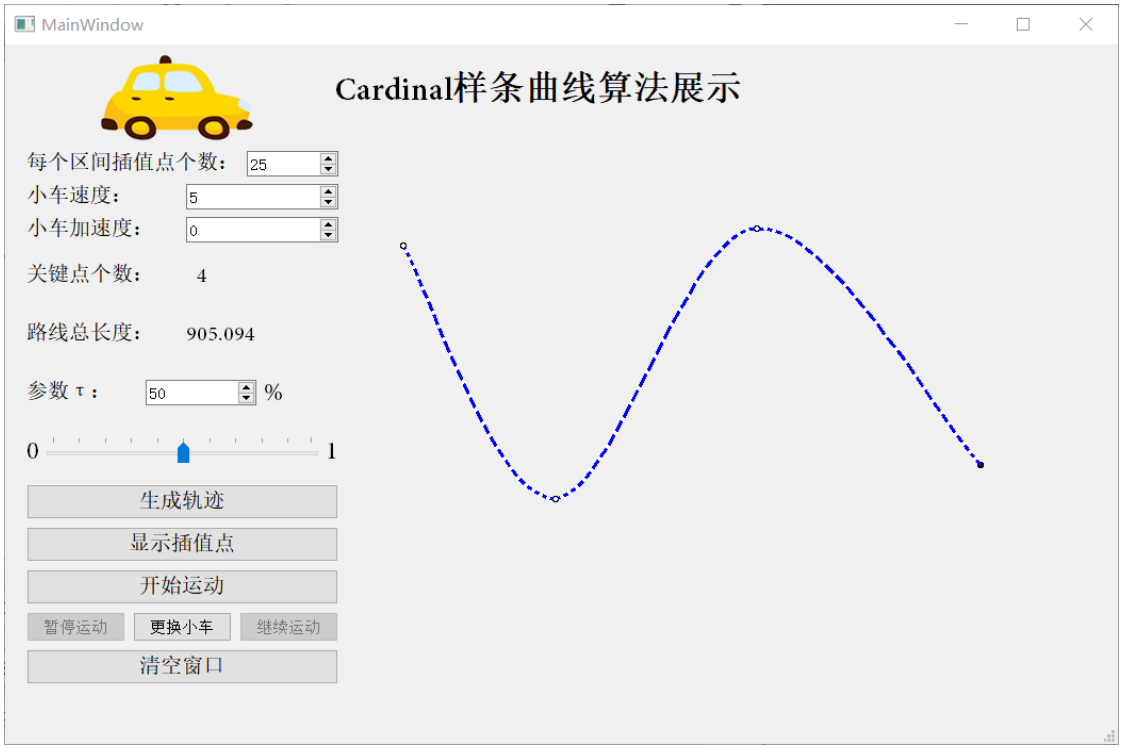


2、不同的 tension（τ）：分别为：

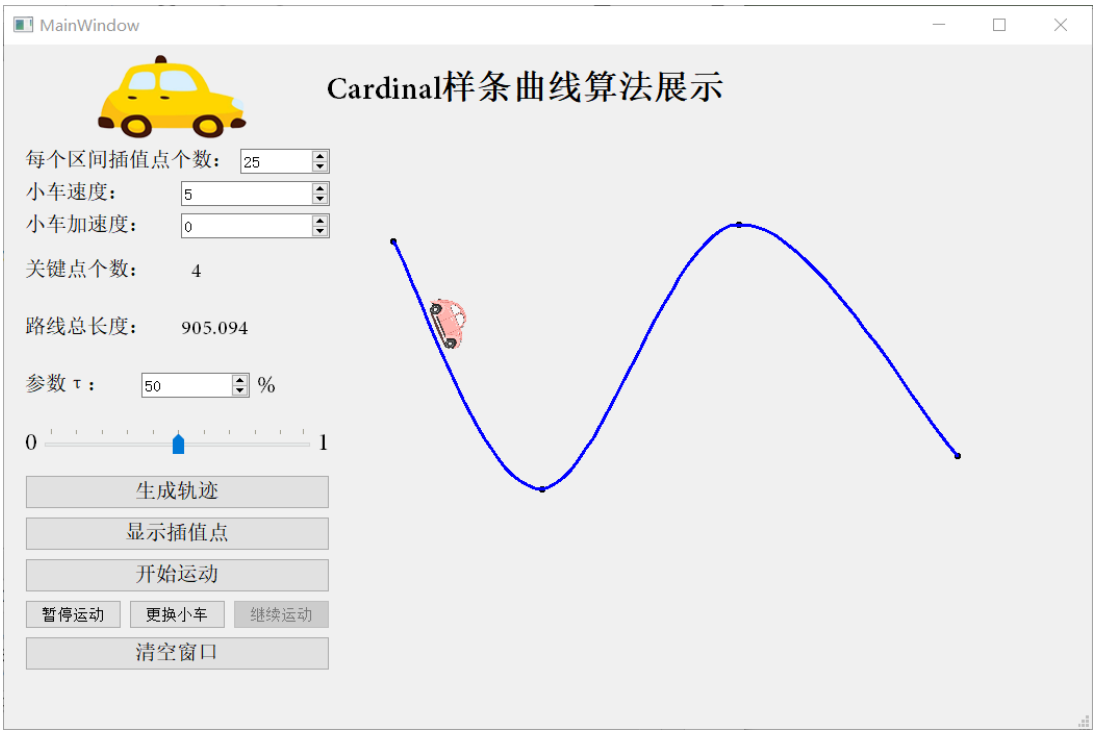
0（绿色），0.25（灰色），0.5（红色），0.75（蓝色），1.0（黄色）



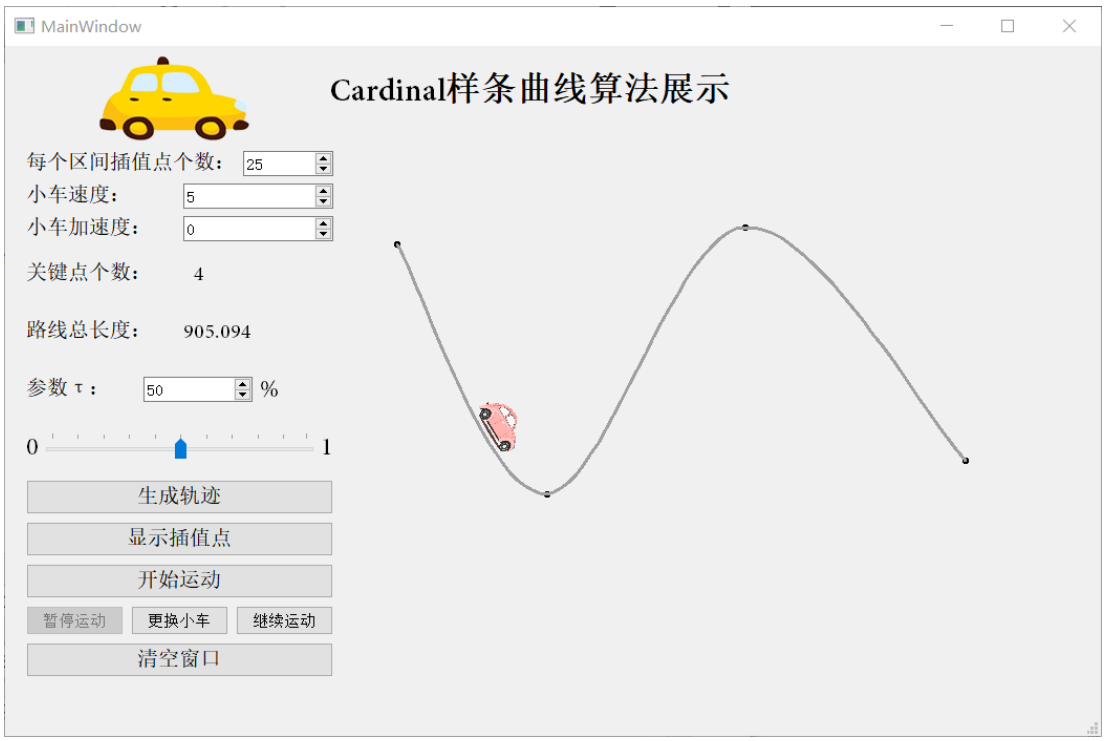
显示插值点（白色为内部插值点）：



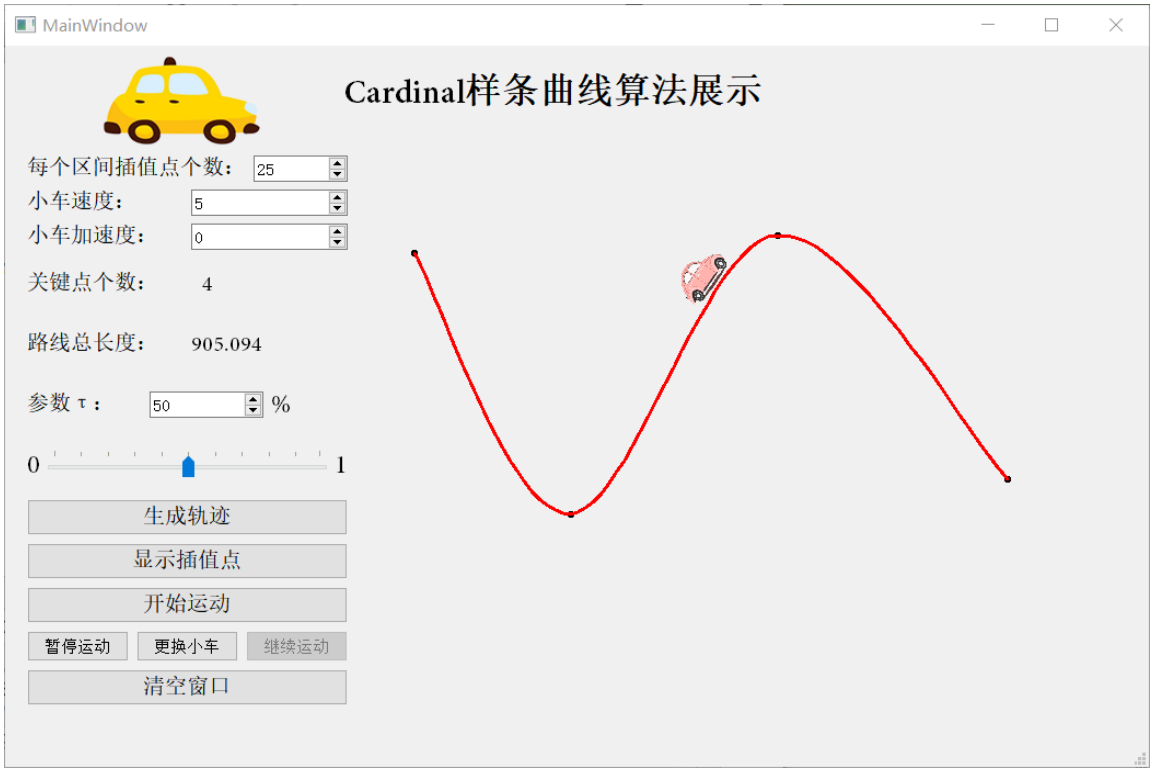
小车开始运动：


MainWindow
Cardinal样条曲线算法展示
每个区间插值点个数：
25
小车速度：
5
小车加速度：
0
关键点个数：
4
路线总长度：
905.094
参数 τ：
50
%
0
1
生成轨迹
显示插值点
开始运动
暂停运动
更换小车
继续运动
清空窗口

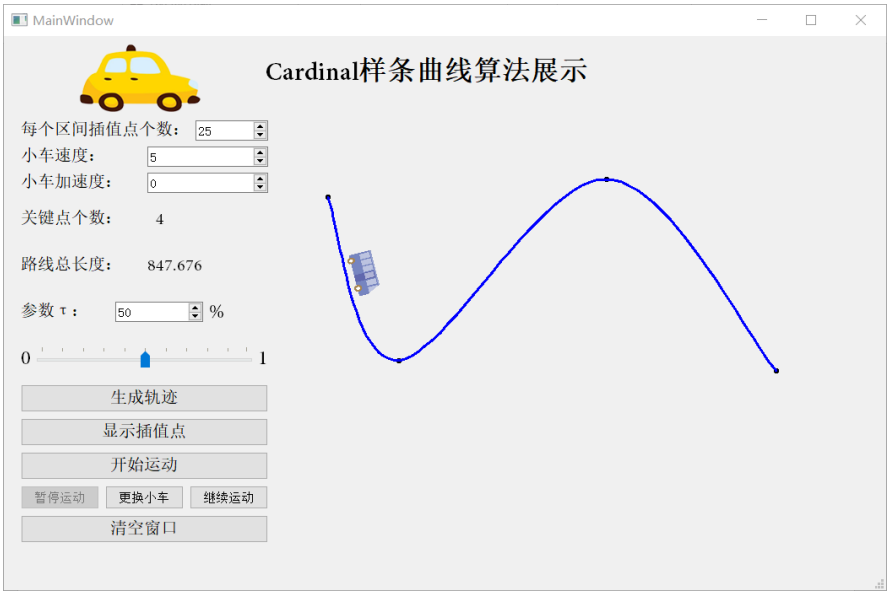
暂停运动：

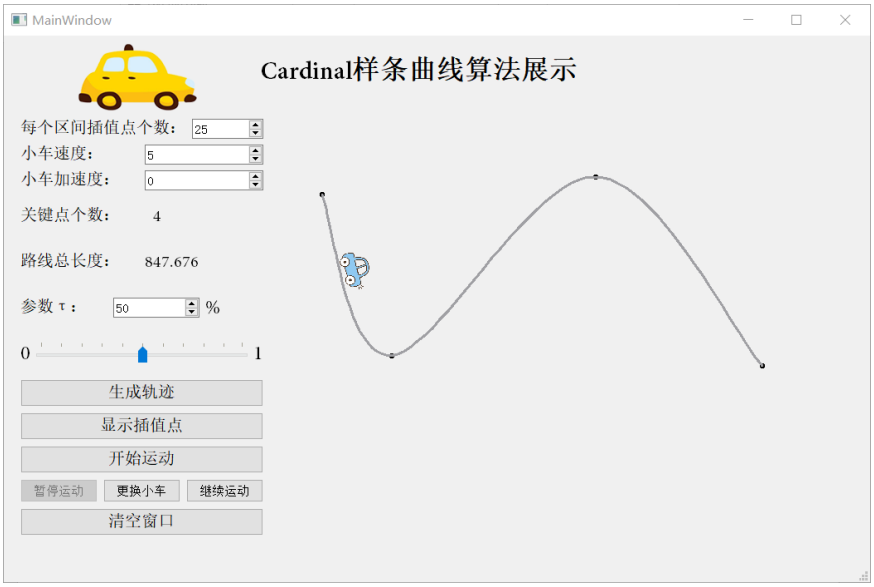

MainWindow
Cardinal样条曲线算法展示
每个区间插值点个数：
25
小车速度：
5
小车加速度：
0
关键点个数：
4
路线总长度：
905.094
参数 τ：
50
%
0
1
生成轨迹
显示插值点
开始运动
暂停运动
更换小车
继续运动
清空窗口

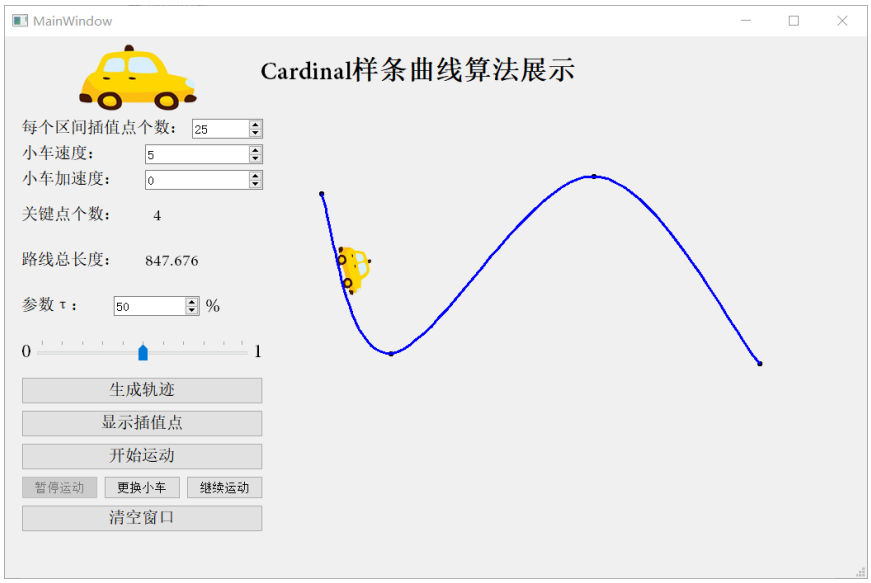
继续运动：

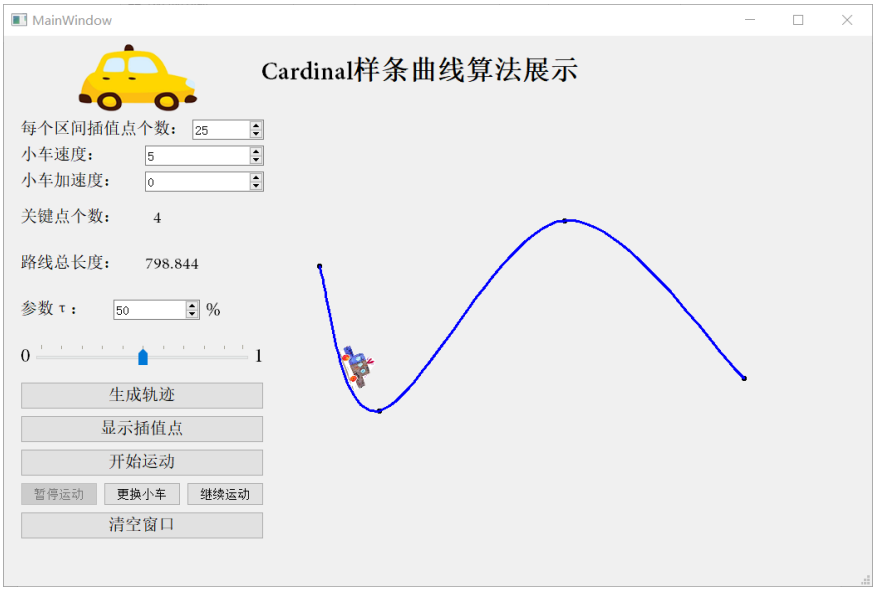

MainWindow
Cardinal样条曲线算法展示
每个区间插值点个数：
25
小车速度：
5
小车加速度：
0
关键点个数：
4
路线总长度：
905.094
参数 τ：
50
%
0
1
生成轨迹
显示插值点
开始运动
暂停运动
更换小车
继续运动
清空窗口

更换小车：


MainWindow
Cardinal样条曲线算法展示
每个区间插值点个数：
25
小车速度：
5
小车加速度：
0
关键点个数：
4
路线总长度：
847.676
参数 τ：
50
%
0
1
生成轨迹
显示插值点
开始运动
暂停运动
更换小车
继续运动
清空窗口


MainWindow
Cardinal样条曲线算法展示
每个区间插值点个数：
25
小车速度：
5
小车加速度：
0
关键点个数：
4
路线总长度：
847.676
参数 τ：
50
%
0
1
生成轨迹
显示插值点
开始运动
暂停运动
更换小车
继续运动
清空窗口


MainWindow
Cardinal样条曲线算法展示
每个区间插值点个数：
25
小车速度：
5
小车加速度：
0
关键点个数：
4
路线总长度：
847.676
参数 τ：
50
%
0
1
生成轨迹
显示插值点
开始运动
暂停运动
更换小车
继续运动
清空窗口


MainWindow
Cardinal样条曲线算法展示
每个区间插值点个数：
25
小车速度：
5
小车加速度：
0
关键点个数：
4
路线总长度：
798.844
参数 τ：
50
%
0
1
生成轨迹
显示插值点
开始运动
暂停运动
更换小车
继续运动
清空窗口

六、 实验感悟及问题

通过本次实验，我对 Cardinal 样条曲线算法有了更深刻的认识和了解，也亲身实践了操作物体平移、旋转、运动等的算法，并且学习和实践了 Qt 的使用，并利用 Qt 实现图形化界面编程，设计和实现界面。