浙江大学

课程名称：计算机动画

姓　　名：******

学　　院：计算机科学与技术学院

专　　业：数字媒体技术

学　　号：315010****

指导教师：于金辉

2017　年　10 月　10 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：动画路径控制曲线与样条路径曲线弧长参数化

学生姓名：****** 专业：数字媒体技术 学号：315010****

同组学生姓名：无 指导老师：于金辉

实验地点：曹光彪西 304 实验日期：2017 年 10 月 10 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，实现 Cardinal 样条曲线的生成和弧长参数化。

二、 实验内容和原理

1. 找出 Cardinal 样条曲线的矩阵表示和程序之间的对应关系。
2. 给定若干控制点的位置，计算出控制点之间的插值点，显示出样条曲线。
3. 改变曲线弯曲程度的参数 $\tau \in [0, 1]$大小，观察曲线形状的变化。
4. 实现样条曲线弧长参数化。
5. 在弧长参数化后的路径曲线上对某物体（如汽车）进行运动控制。
6. 设计不同的速度曲线。

三、 实验器材

JetBrains WebStorm 2017.2.4, Google Chrome 61.0, JQuery plugin: JCanvas

四、 实验步骤

1. 找出 Cardinal 样条曲线的矩阵表示和程序之间的对应关系。

程序在创建一条新的 Cardinal 曲线时，首先根据传入的 tension 值构造 M 矩阵：

$$\mathbf{M} = \begin{bmatrix} -\tau & 2-\tau & \tau-2 & \tau \\ 2\tau & \tau-3 & 3-2\tau & -\tau \\ -\tau & 0 & \tau & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

在程序中这个矩阵的构造过程对应方法 makeMatrix()：

```
makeMatrix() {
    msg("Making Matrix...");
    let t = this.tension;
    let m = new Array(16);
    m[0] = -t;      m[1] = 2 - t;   m[2] = t - 2;       m[3] = t;
    m[4] = 2 * t;   m[5] = t - 3;   m[6] = 3 - 2 * t;  m[7] = -t;
    m[8] = -t;      m[9] = 0;       m[10] = t;          m[11] = 0;
    m[12] = 0;      m[13] = 1;      m[14] = 0;          m[15] = 0;
    this.m = m;
}
```

Figure 1 MakeMatrix()

接着，将传入的控制点数组复制首尾两点并存入 points 变量数组中。

最后，对于每一段步长为 1/grain 的插值位置，将其累计步长作为 u，连同自 point 数组中的第一个点起的四个点的 x、y 值，以及构造的 M 矩阵分别代入 P(u)计算式，得到该插值位置的插值点 x 和 y 坐标。将这一系列点存入结果数组 result，即得到了这四个点中居中两点的曲线段上的各个插值点坐标。这一系列计算过程，尤其是 P(u)的矩阵计算，可以在 combineSegments()方法和两个静态方法：makeSegment()和 multiply()中清楚地观察到：

```
combineSegments() {
    msg("Combining segments...");
    let step = [];
    let result = [];
    for (let i = 0;i < this.grain;i++) {
        step.push(i / this.grain);
    }
    for (let i = 1; i < this.points.length - 2; i++) {
        for (let j = 0;j < this.grain;j++) {
            let x = CdnSpline.makeSegment(this.m, [this.points[i - 1].x, this.points[i].x,
                this.points[i + 1].x, this.points[i + 2].x], step[j]);
            let y = CdnSpline.makeSegment(this.m, [this.points[i - 1].y, this.points[i].y,
                this.points[i + 1].y, this.points[i + 2].y], step[j]);
            result.push(new Point(x, y));
        }
    }
    // Push the last control point into spline_points[].
    result.push(this.points[this.points.length - 1]);
    msg("Generating result...");
    this.points = result;
}
```

Figure 2 combineSegments()

```
static makeSegment(m, n, u) {
    let c = new Array(4);
    for (let i = 0;i < 4;i++) {
        c[i] = CdnSpline.multiply(m, n, i * 4);
    }
    return c[3] + u * (c[2] + u * (c[1] + u * c[0]));
}

static multiply(M, n, order) {
    let sum = 0;
    for (let i = 0;i < 4;i++) {
        sum += M[order + i] * n[i];
    }
    return sum;
}
```

Figure 3 makeSegment() 和 multiply()

2. 给定若干控制点的位置，计算出控制点之间的插值点，显示出样条曲线。

计算插值点的过程已在 1 中详述。显示样条曲线的过程由 CdnSpline 的 SplinePoints 类的成员变量 spline_points 负责。它根据传入的点列数组构造出两类 JCanvas 对象：JCanvas 曲线与 JCanvas 点列，由他们将离散的插值点和生成的样条曲线显示在画布上。Class SplinePoints 的结构如图所示：

```
class SplinePoints {
    constructor(points, show_dots, line_width) {
        // Array consists of class Point
        this.points = points;
        // Array consist of class SplineDots
        msg("Drawing dots...");
        this.spline_dots = new SplineDots(points, show_dots);
        // class Line
        msg("Drawing spline...");
        this.spline_points_line = new Line('rgba(255, 50, 50, 0.6)', line_width, this.points,
            'SplinePointsLine');
    }

    removePoints() {...}

    drawDots() {...}

    removeDots() {...}

    getPoints() {...}

    setLineWidth(line_width) {...}
}
```

Figure 4 class SpilePoints

3. 实现样条曲线弧长参数化。

normalizeSpline()是对已完成初步插值的曲线点列进行进一步的统一化的方法。在这个方法中，程序首先调用 makeLengthList()方法，使用简化的 Simpson 算法计算各个初步插值的曲线点与曲线起点间的曲线弧长距离。在这个过程中，需要用到曾经计算过的 M 矩阵计算结果来计算算法所需的五个系数。

```
makeLengthList() {
    let length = 0;
    this.length_list[0] = 0;
    // Duplicate the last point
    this.points.push(this.points[this.points.length - 1]);
    // Duplicate the first point
    this.points.unshift(this.points[0]);
    for (let i = 0;i < this.points.length - 3;i++) {
        this.calculateCoefficient(this.m,
            [this.points[i], this.points[i + 1], this.points[i + 2], this.points[i + 3]]);
        length += this.calculateSimpsonLength(0, 1);
        this.length_list.push(length);
    }
    this.length = length;
}
```

Figure 5 makeLengthList()

在得到曲线的长度索引表（length_list）之后，调用 findPointByLength()

方法寻找想要得到的曲线长度值所在的曲线位置。在这个过程中，首先确定该长度值所在的曲线段，然后使用二分法查找，其精度受阈值 SEARCH_STEP 所限。不断重复这个过程，就能得到较为精确的曲线长度值对应点位置数组。即输入怎样的长度间隔，就能求出对应的点，此时的曲线已被“参数化”，即可如对待直线运动一般对待 Cardinal 样条曲线上的运动。曲线参数化的结果被存入 normalized_spline_points 并显示在画布上。

```
findPointByLength(length) {
    let segment_no = 0;
    let u_max = 1;
    let u_min = 0;
    let u_current;
    let current_length;
    let difference_length;
    while (this.length_list[segment_no] < length) {
        segment_no++;
    }
    segment_no -= 1;
    let target_length = length - this.length_list[segment_no];
    let param_list = this.calculateCoefficient(this.m,
        [this.points[segment_no], this.points[segment_no + 1], this.points[segment_no + 2], this.points[segment_no + 3]]);
    do {
        u_current = (u_max + u_min) / 2;
        current_length = this.calculateSimpsonLength(u_min, u_current);
        difference_length = target_length - current_length;
        if (difference_length > 0) {
            u_min = u_current;
            target_length -= current_length;
        }
        else {
            u_max = u_current;
        }
    } while (Math.abs(difference_length) > SEARCH_STEP);
    return new Point(CdnSpline.calculatePoint(u_current, param_list[0]),
        CdnSpline.calculatePoint(u_current, param_list[1]))
}
```

Figure 6 findPointByLength()

4. 在弧长参数化后的路径曲线上对某物体（如汽车）进行运动控制。

使用预计算的逐帧动画来操作物体（火箭）沿路径曲线的运动，动画控制交由 startRocket()、playAnimation()等一系列方法来实现。其运动帧的位置来源于 3 中的计算结果。

```
startRocket() {
    if (this.normalized_spline_points) {
        this.track_points = this.normalized_spline_points.getPoints();
        this.rocket = new Rocket(ROCKET_SRC, this.track_points[0].x, this.track_points[0].y, 50, 50,
            CdnSpline.calculateAngle(this.track_points[0], this.track_points[2]), 1);
        for (let i = 0;i < this.track_points.length - 2;i++) {
            this.angle_list.push(CdnSpline.calculateAngle(this.track_points[i], this.track_points[i + 2]));
        }
        this.rocket.show();

        this.then = Date.now();
        this.playAnimation();
        return true;
    }
    else {
        msg("Please normalize the spline first!");
        return false;
    }
}

updateRocket() {...}

removeRocket() {...}

playAnimation() {...}

stopAnimation() {...}

pauseAnimation() {...}

stepAnimation() {...}
```

Figure 7 Start Animation

5. 设计不同的速度曲线

控制 3 中曲线距离的取值，即可控制最终的关键帧动画结果。程序预设了四种取值方式，分别为线性速度（匀速）、线性速度（加速）、非线性速度（加速）和非线性速度（加速再减速）。

```
if (method === NORMALIZE_METHOD_TYPE_1) {
    for (let i = 1;i < total_frames;i++) {
        result.push(this.findPointByLength(i * step * this.length));
    }
}
else if (method === NORMALIZE_METHOD_TYPE_2) {
    for (let i = 1;i < total_frames;i++) {
        result.push(this.findPointByLength(Math.pow(i * step, 2) * this.length));
    }
}
else if (method === NORMALIZE_METHOD_TYPE_3) {
    for (let i = 1;i < total_frames;i++) {
        result.push(this.findPointByLength((Math.sin(Math.PI / 2 * (i * step - 1)) + 1) * this.length));
    }
}
else if (method === NORMALIZE_METHOD_TYPE_4) {
    for (let i = 1;i < total_frames;i++) {
        result.push(this.findPointByLength((Math.sin(Math.PI * (i * step - 0.5)) + 1) / 2 * this.length));
    }
}
```

Figure 8 Different velocity graphs

五、 实验结果分析

1. 主界面

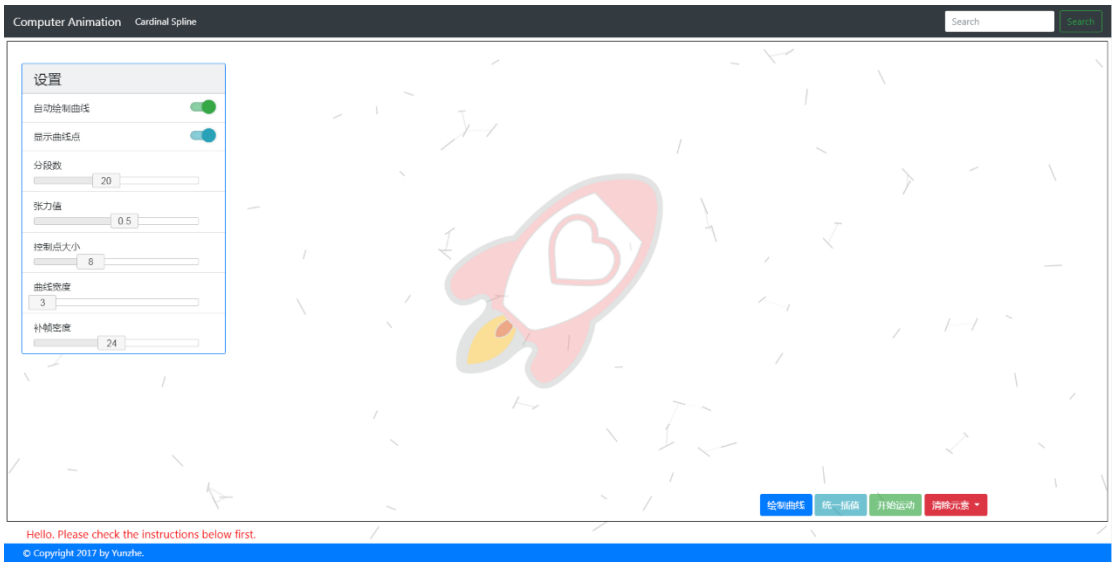

Figure 9 主界面

使用一个现代浏览器打开 spline.html 或者访问 222.205.56.130/spline，浏览器将自动加载相关资源和脚本文件。最大化浏览器窗口以容纳所有视觉元素。

2. 操作指引

将页面滚动至底部，可以看到关于这个应用的操作指引。

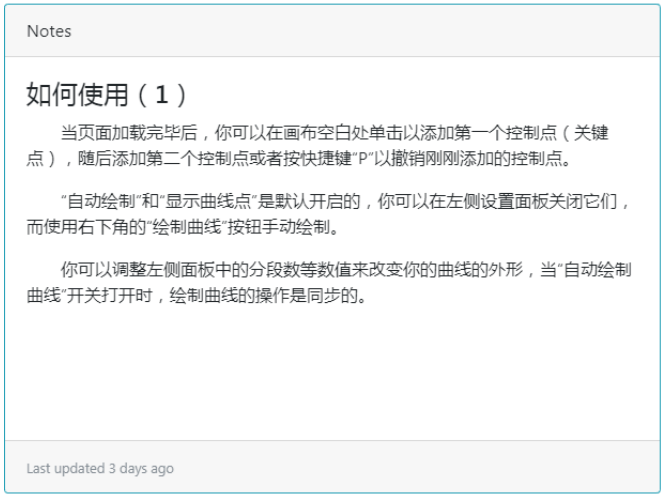

Figure 10 **操作指引** 1

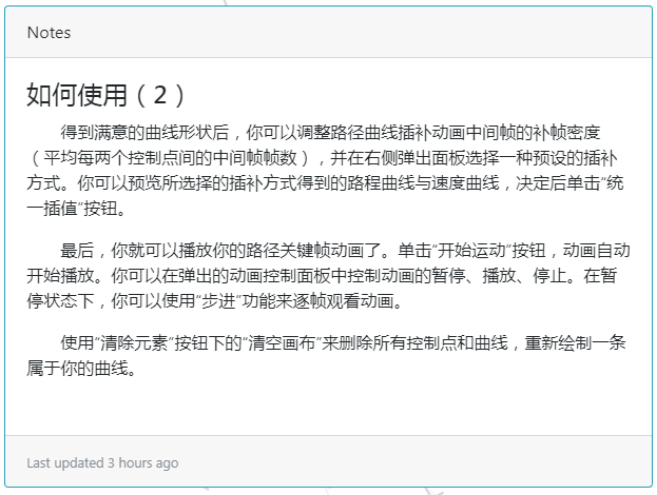

Figure 11 **操作指引** 2

3. 绘制控制点

将页面滚动至顶部，用鼠标指针在画布空白处创建控制点。

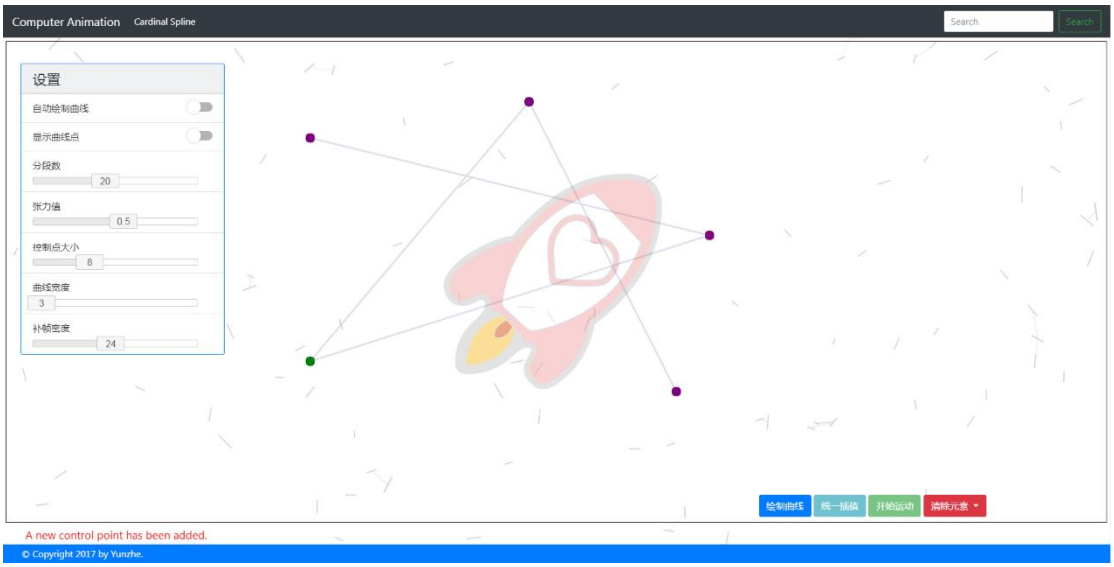

Figure 12 **绘制控制点**

4. 生成曲线

单击右下角的“绘制曲线”按钮，绘制 Cardinal 插值曲线。

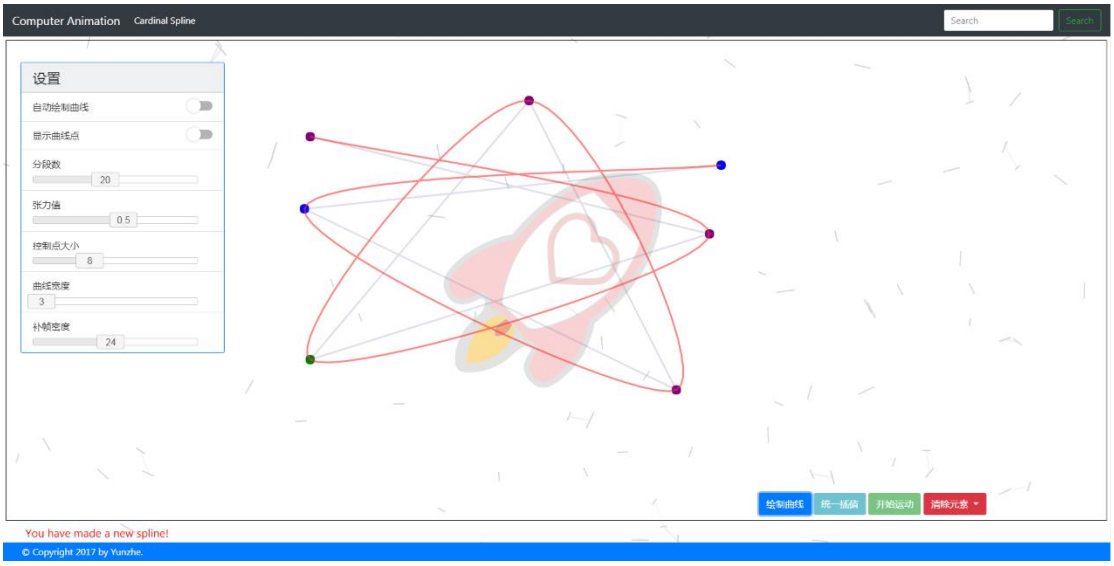

Figure 13 **生成曲线**

5. 改变曲线弯曲程度的参数 $\tau \in [0, 1]$大小，观察曲线形状的变化。

在左侧的设置面板中拖动滑动条来改变“张力值”，再次单击“绘制曲线”按钮。

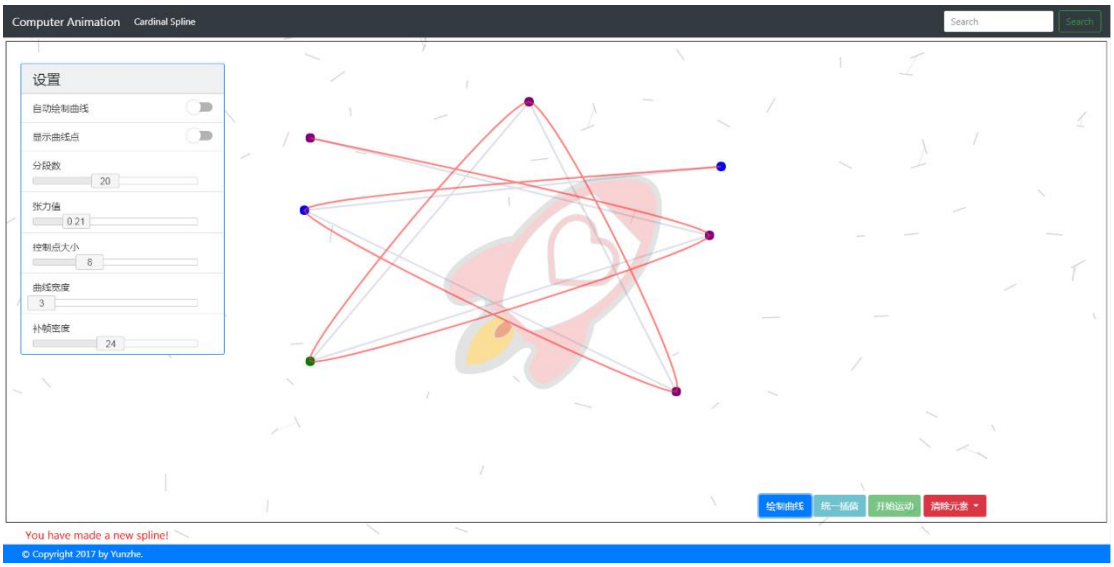

Figure 14 **减小张力值**

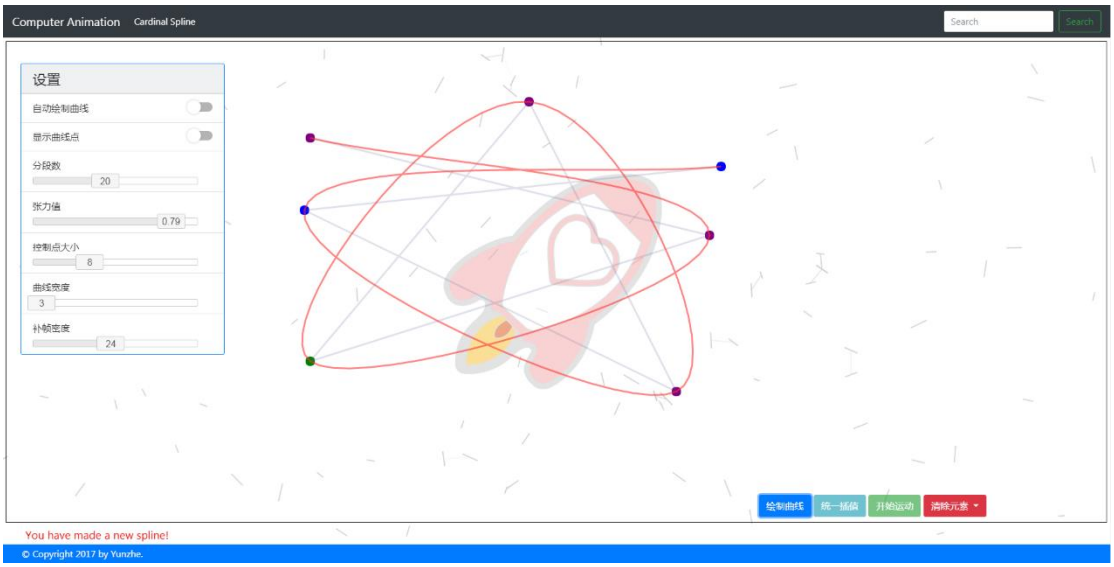

Figure 15 **增大张力值**

6. 显示曲线点

打开“显示曲线点”开关。

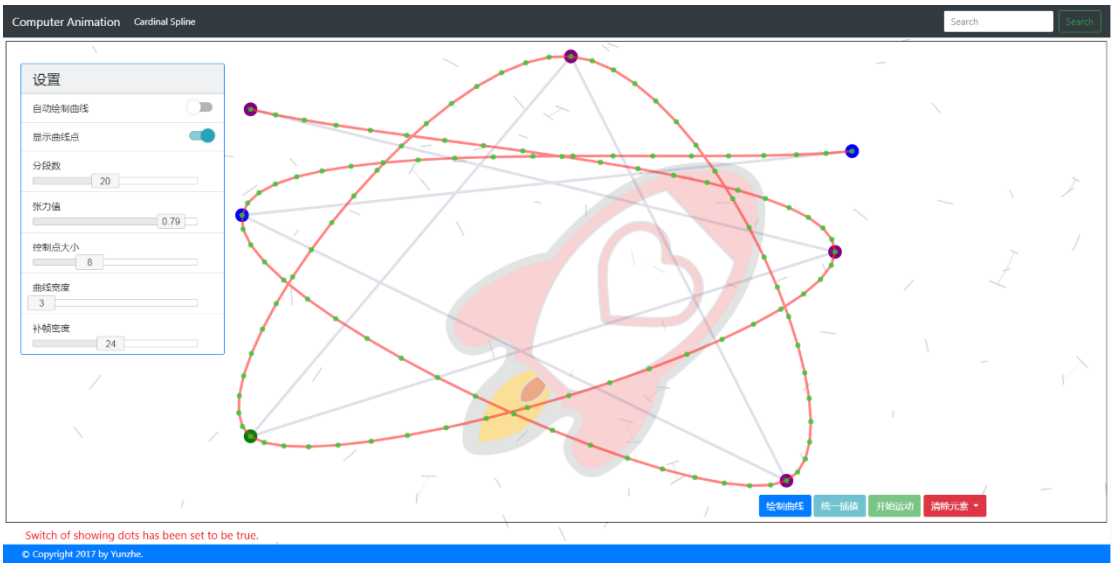

Figure 16 **显示曲线点**

7. 改变 grain

改变“分段数”的值，再次单击“绘制曲线”按钮。“分段数”意味着每两个控制点间的曲线段数。

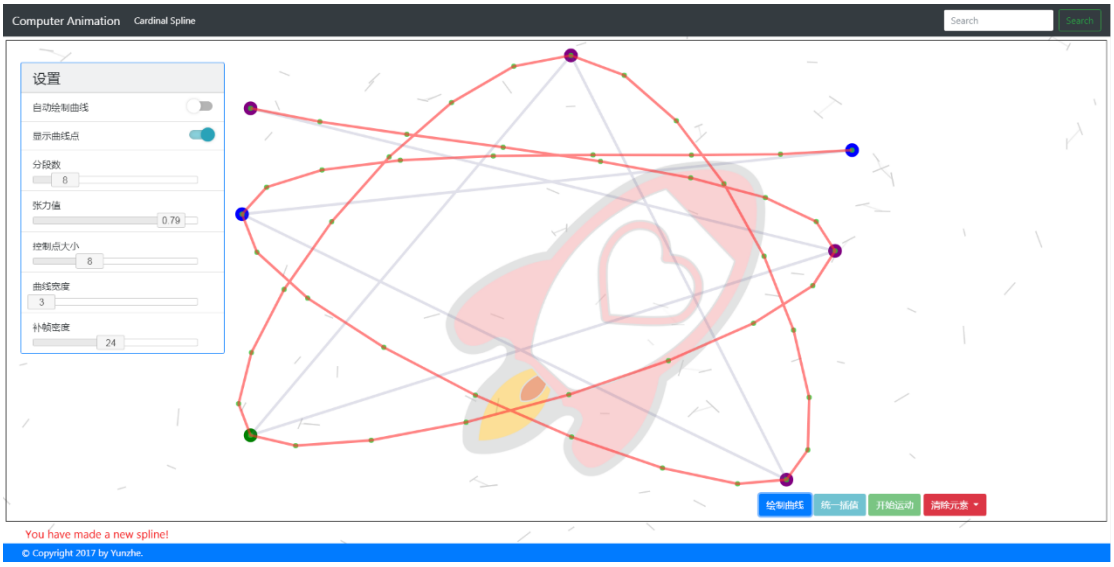

Figure 17 **改变分段数**

8. 插补动画中间帧

单击“统一插值”按钮进行曲线参数化处理。

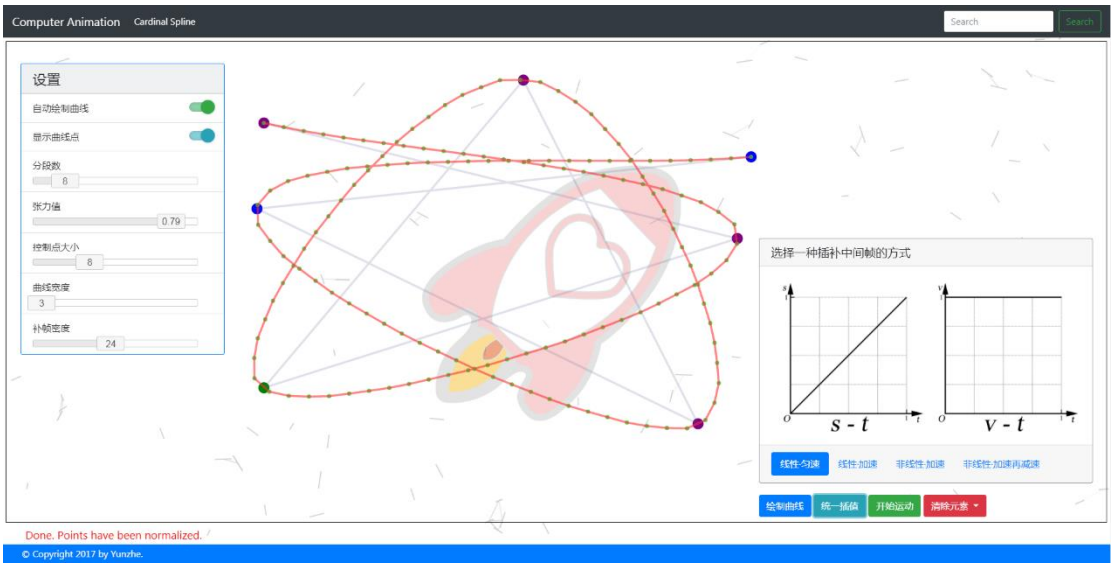

Figure 18 插补中间帧

9. 改变插补方式与补帧密度

改变补中间帧的方式并调整补帧密度，再次单击“统一插值”按钮。

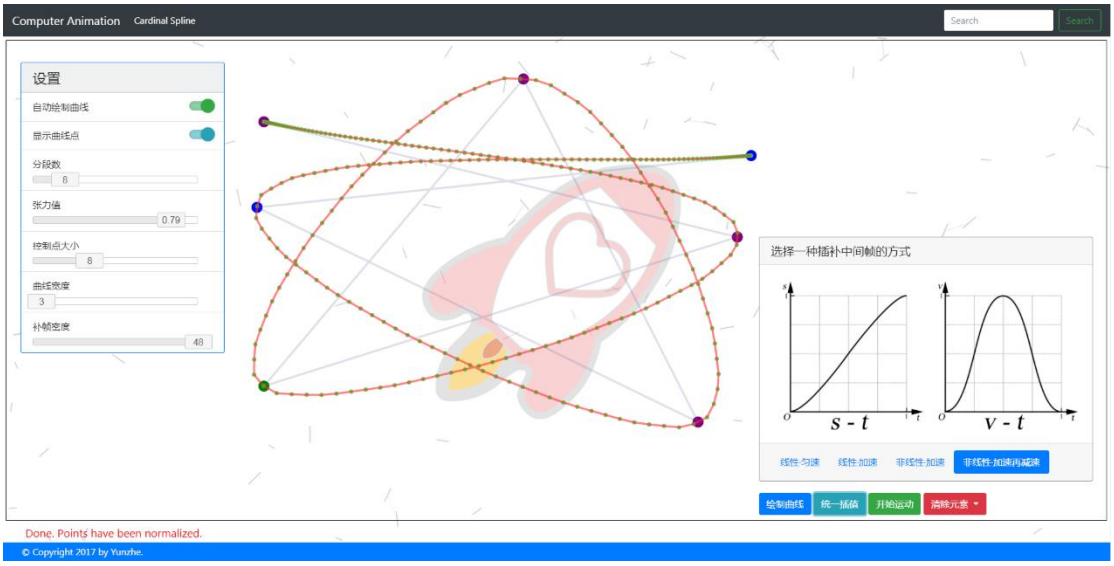

Figure 19 改变插补方式与补帧密度

10. 播放动画

完成插值后，单击“开始运动”按钮播放逐帧动画。在此过程中可以依次单击“暂停”和“步进”按钮来逐帧观察火箭状态。

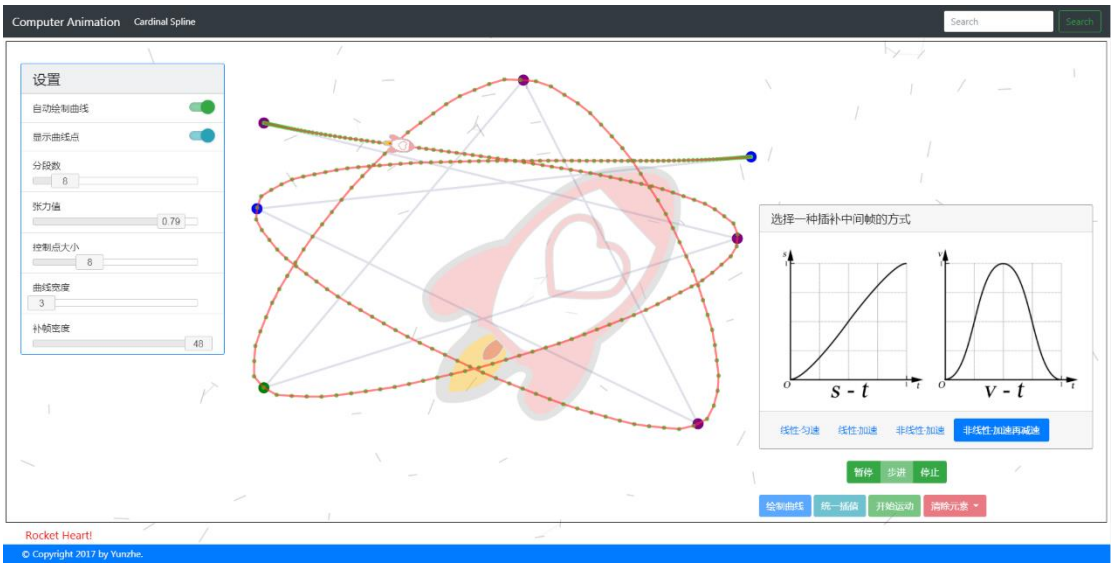

Figure 20 **播放动画**

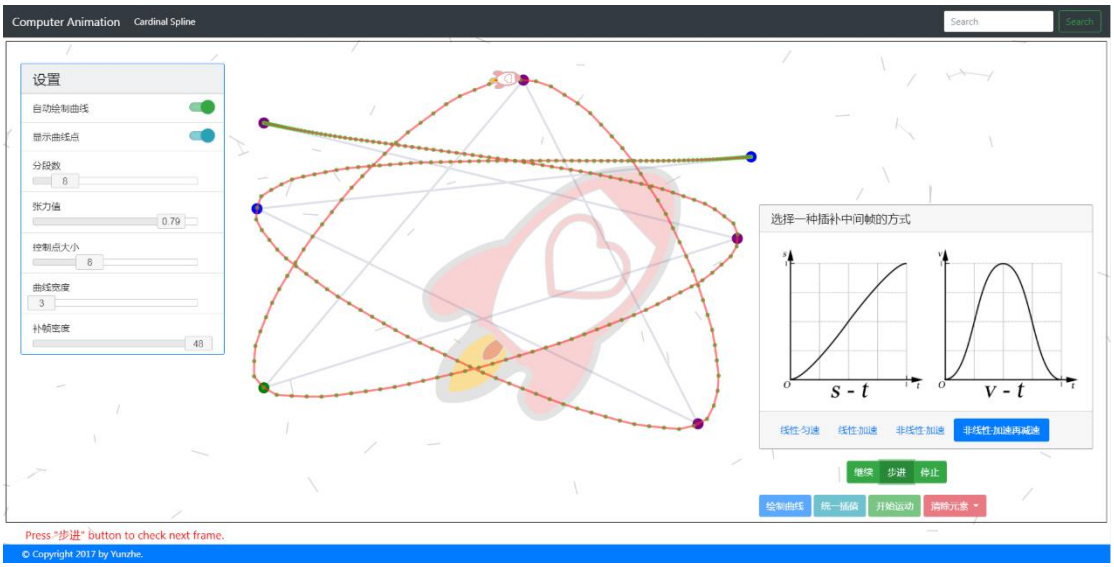

Figure 21 **步进动画**

更多细节请参考源码与演示视频。

六、实验感想

经过本次实验，我对 Cardinal 算法生成样条曲线有了更深的理解，经过实践得来的直接经验，使我能熟练编写生成 Cardinal Spline 的程序并对曲线进行参数化处理，同时对 JavaScript 语言的掌握也更进了一步。处理逐帧动画的过程中，我体会到了传统动画作画的艰辛与不易，但也只有传统动画才能精妙地表现人物的动作映射出的心理状态，这是计算机辅助动画所难以做到的。