

一、实验目的和要求

- 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、实验内容和原理

- 选用 Cardinal 样条曲线来表示运动路径以及不规则物体的形状，掌握它的表示 了解控制参数对曲线形状的影响。
- 找出 Cardinal 样条曲线的矩阵表示和程序之间的对应关系。
- 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。
- 在实验报告中简述 Cardinal 样条曲线，附上图形结果并用文字讨论。
- 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、实验平台

HTML5 + CSS + Javascript @Chrome

四、实验步骤

1. 背景

在设计矢量图案的时候，我们常常需要用到曲线来表达物体造型，单纯用鼠标轨迹绘制显然是不能够满足要求的。于是我们希望能够实现这样的方法：通过设计师手工选择控制点，再通过插值得到过控制点（或在附近）的一条平滑曲线。在这样的需求下，样条曲线诞生了。因此我们的目标是：通过给定的关键帧点生成一条希望的直线和曲线。

2. 数学原理

(1) 引入：直线插值

曲线插值前我们先来观察一下直线插值：生成一条直线，给定直线首尾的关键点 P_0, P_1 ，就能确定这条直线的特性，比如 $y = kx + b$ 中的斜率 k 和 y 轴偏移值 b 。通过线性 (P_0, P_1 线性相关) 插值（线性的给中间插上一定数量的点使看起来连续）的方式就能得到我们想要的直线段（图2.1）。

- 最简单的线性插值(**linear interpolation**)，插值变量是u,

$$P(u) = (1 - u) \cdot P_0 + u \cdot P_1$$

这里 $u \in [0,1]$

- 写成代数形式:

$$P(u) = (P_1 - P_0) \cdot u + P_0$$

- 写成矩阵形式:

$$P(u) = \begin{bmatrix} u & 1 \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \end{bmatrix} = U^T M B$$

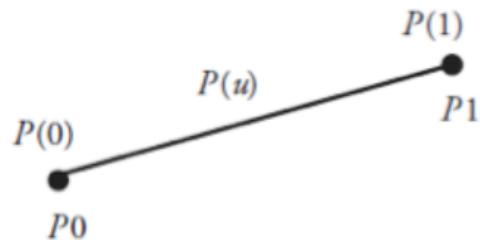


图2.1

(2) 曲线插值

而曲线插值的难点在于：根据所给控制点确定样条曲线的多项式函数。我们将样条曲线进行切分，要确定每一段曲线（相邻两个控制点之间），需要四个参数：

首尾控制点的位置(参数化后的 $P(u)$)以及他们的方向($P(u)$ 一阶导数)，通过插值来对实现中间点的平滑过渡，再将每个小段的曲线进行连接。

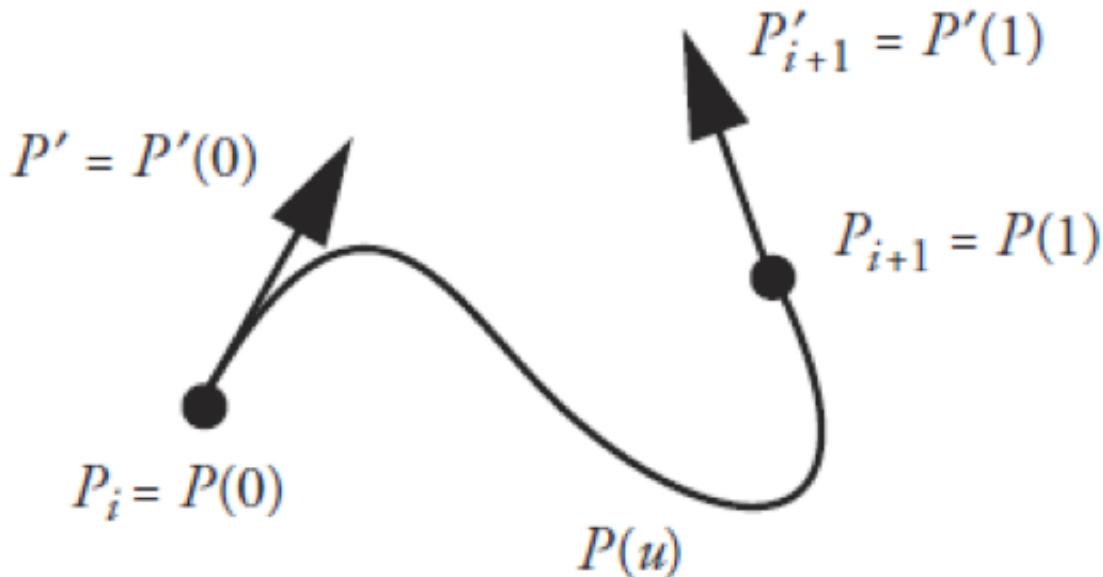


图2.2

总结一下，曲线的生成分为两步：

- 对每两个控制点之间进行插值得到曲线段(segment) (图2.2)
- 连接两两关键帧点的曲线段 (图2.3)

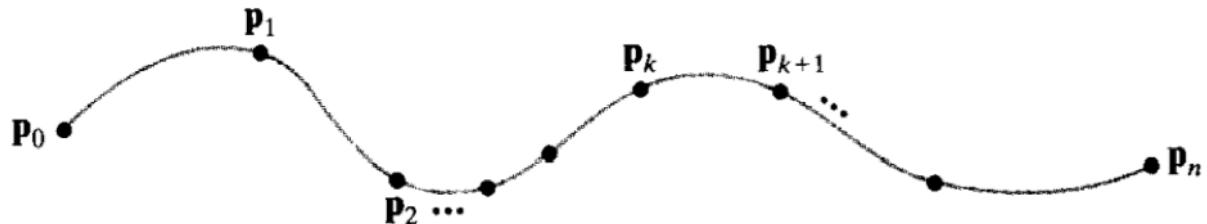


图2.3

(3)方法: 三次多项式插值

对两个控制点进行插值的关键在于得到一个参数化的函数去拟合样条曲线，因此我们需要通过给定条件($P(u)$ 和 $P'(u)$)去确定这个函数的系数，然后根据得到的函数计算插值点的位置得到曲线。这里选择三次多项式的原因在于灵活性和计算效率：

- 与更低次多项式相比，三次样条在模拟任何曲线形状时显得更灵活。（能满足大多数情况下的曲率调整，比如二次抛物线太对称，一次就是直线）
- 与更高次多项式相比，三次样条只需较少的计算与存储空间，并且较为稳定；（只需要一个 4×4 的基函数矩阵）

(4)曲线连续性

为了保证分段参数曲线从一段到另一端平滑过渡，可以在连接点处要求各种连续性（参考微积分中的函数连续性与导数的关系）。

- C^0 连续：有相同的公共点，如图(a)
- C^1 连续：有相同的公共点且公共点处有相同的一阶导数（切线斜率），如图(b)
- C^2 连续：有相同的公共点且有相同的公共点且公共点处有相同的一阶导数和二阶导数（切线斜率的变化率），如图c

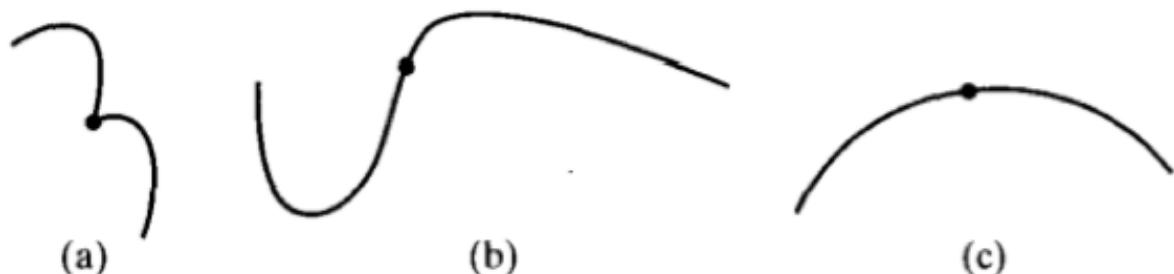


图2.4

(5)寻找合适的插值方案: 自然样条曲线

一种简单的插值方法是自然样条插值，就是对如图2.2的一条有 $n+1$ 个控制点($p_0 \sim p_n$) n 个分段的曲线求三次多项式系数这样就有 $4n$ 个系数要确定（需要列出 $4n$ 个方程才能求解出来），通过曲线内部的 $n-1$ 个点每相邻曲线段公共点的一阶及二阶导数相等且都通过该点，能确定 $4n-4$ [$4*(n-1)$]个方程，再加上两个“隐含”控制点 p_0 和 p_n 的二阶导数为0得到 $4n$ 个方程来求解。缺点是一个点发生了变化其他点都跟着受到影响，难以满足我们需要实时的调整的需求。

(6)Hermite曲线

为了解决自然样条曲线不能局部控制的缺点，Hermite插值对每个控制点的切线进行了限制为D（由用户给出），这样切线就变成了D*Pk（D斜率+经过该点），使每个曲线段仅依赖于端点位置的约束。如对每小段曲线的边界条件表示为：

$$\begin{aligned}P(0) &= P_k \\P(1) &= P_{k+1} \\P'(0) &= D_{P_k} \\P'(1) &= D_{P_{k+1}}\end{aligned}$$

这里P是向量，对于xy分量来说分别都符合这个边界条件。那么如何保证曲线连续的呢？即对于一个连接点Pk-1来说，这段曲线相应的终点为Pk，在下一段曲线里Pk则作为这段曲线的起始点，保证Pk这个点的函数值和导数值相等就可以。现在点的位置都是Pk，导数值都是DPk，二阶导数值都是D。向量方程为：

$$P(u) = au^3 + bu^2 + cu + d, u \in [0, 1]$$

其中P和abcd都是向量，结合公式1-2转换成矩阵形式我们可以得到：

$$\mathbf{P}(u) = [u^3 \quad u^2 \quad u \quad 1] \cdot \mathbf{M}_H \cdot \begin{bmatrix} \mathbf{p}_k \\ \mathbf{p}_{k+1} \\ \mathbf{D}\mathbf{p}_k \\ \mathbf{D}\mathbf{p}_{k+1} \end{bmatrix}$$

其中M*P部分就是通过对Hermite函数推导得到的系数矩阵[a b c d]'，Pk和Pk+1由用户给定，DPk和DPk+1由给定D值计算获得，根据不同的控制方式可以生成有不同曲线，Cardinal曲线就是Hermite曲线的一种变形，通过对切线斜率进行控制来控制曲线的平滑度。

(7)Cardinal曲线

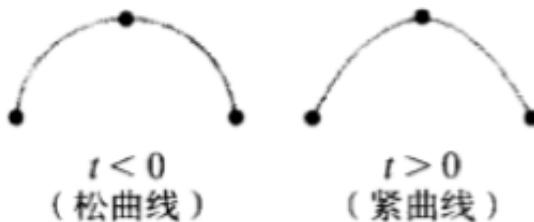
因为在Hermite曲线中斜率D还是需要用户给出，Cardinal样条曲线可以由相邻两控制点坐标来计算斜率从而使曲线完全由控制点Pk来决定，但是在这里引入了一个t作为在每个公共点上尖锐程度的控制值。

Cardinal曲线的边界条件方程：

$$\begin{aligned}\mathbf{P}(0) &= \mathbf{p}_k \\ \mathbf{P}(1) &= \mathbf{p}_{k+1} \\ \mathbf{P}'(0) &= \frac{1}{2}(1-t)(\mathbf{p}_{k+1} - \mathbf{p}_{k-1}) \\ \mathbf{P}'(1) &= \frac{1}{2}(1-t)(\mathbf{p}_{k+2} - \mathbf{p}_k)\end{aligned}$$

这样控制点Pk和Pk+1处的斜率和弦PkPk+2和PkPk+1成正比（理解为表示比较接近的控制点对曲线曲率的影响）

其中t（称为张量）控制Cardinal样条和输入控制点之间的松紧程度，对曲线的影响程度可以通过下图看出来：



通过跟Hermite类似的系数求解方式，将边界条件代入可以得到矩阵表达式：

$$\mathbf{P}(u) = [u^3 \quad u^2 \quad u \quad 1] \cdot \mathbf{M}_C \cdot \begin{bmatrix} \mathbf{p}_{k-1} \\ \mathbf{p}_k \\ \mathbf{p}_{k+1} \\ \mathbf{p}_{k+2} \end{bmatrix}$$

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

3. Cardinal曲线的代码实现

我选择了HTML5 + CSS实现前端操作界面，Javascript实现Cardinal曲线的原理，直接在浏览器中打开即可。

由上一节得到的Cardinal插值的样条曲线的矩阵，对每一段曲线 $P_k P_{k+1}$ 来说，都可以通过拟合一个被参数化为 $P(u)$ 函数 ($0 \leq u \leq 1$) 的图形的形式。

同样， $M \cdot P$ 得到的是系数矩阵 $[a \ b \ c \ d]$ 的转置， s 是引入的表示每个连接点处曲线尖锐程度的变量—张量
所以我们现在的期望是

- 输入：n个控制点的xy轴坐标数组 $x[], y[]$ ，张量tension，每两个控制点之间插入的密度grain
- 输出：插值好的样条曲线上控制点的一个数组

下面是具体代码实现：

核心算法

1) 初始化数据结构，并进行插值计算准备

```

var jd = new Array(100); //用户输入的控制点位置数组，临时变量存储
var n0=points.length; //控制点个数

//添加起始点和终点
for(var i=1; i<=n0; i++){
    jd[i] = points[i - 1]; //内部点
}
jd[0] = points[0]; //补上隐含的整条曲线起始点
jd[n0+1] = points[n0-1]; //补上隐含的整条曲线终止点
var knots = jd;

```

2)计算Mc矩阵

根据输入的平滑度 tension 值计算Mc矩阵

```

function GetCardinalMatrix (a1) {
    var m = new Array(16);
    m[0]=-a1; m[1]=2.0-a1; m[2]=a1-2.; m[3]=a1;
    m[4]=2.*a1; m[5]=a1-3.; m[8]=-a1; m[9]=0.;
    m[12]=0.; m[13]=1.; m[6]=3.-2*a1; m[7]=-a1;
    m[10]=a1; m[11]=0.; m[14]=0.; m[15]=0.;
    return m;
}

```

3) 计算 $P(u)$ 的值

输入4个点 $P_{k-1}, P_k, P_{k+1}, P_{k+2}$ 的值(x分量或者y分量, 以及参数化好的u值), 返回计算得到的 $P(u)$ 的值

```

function Matrix (p0, p1, p2, p3, u, m) {
    //求解系数
    var a, b, c, d;
    a=m[0]*p0+m[1]*p1+m[2]*p2+m[3]*p3;
    b=m[4]*p0+m[5]*p1+m[6]*p2+m[7]*p3;
    c=m[8]*p0+m[9]*p1+m[10]*p2+m[11]*p3;
    d=m[12]*p0+m[13]*p1+m[14]*p2+m[15]*p3;
    return(d+u*(c+u*(b+u*a))); //au^3+bu^2+cu+d
}

```

4) 插值过程

两次循环:

- 第一次对输入的控制点遍历

- 第二次对每两个控制点之间插值，分别计算xy分量上得出的插值后的函数值

```
//计算
var alpha = new Array(50);
var k0, kml, k1, k2;
//获取MC矩阵
var m = GetCardinalMatrix(tension);
//对两个关键点之间的插值点进行参数化到0~1之间
for(var i=0; i<grain; i++){
    alpha[i]=i*1.0/grain;
}
//从最开始的四个点开始，给第一段曲线插值
kml = 0;
k0 = 1;
k1 = 2;
k2 = 3;
//两次循环第一次对输入的控制点遍历，第二次对每两个控制点之间插值，分别计算xy分量上得出的插值后的函数值，k值分别+
1
for(var i =1; i<n0; i++){
    for(var j=0; j<grain; j++){
        var cpx = Matrix(knots[kml][0], knots[k0][0], knots[k1][0], knots[k2][0], alpha[j], m);
        var cpy = Matrix(knots[kml][1], knots[k0][1], knots[k1][1], knots[k2][1], alpha[j], m);
        Spline.push(new Array(cpx, cpy));
    }
    kml++; k0++; k1++; k2++;
}
```

交互实现

在核心算法的基础上，为了提高用户友好度，我们还需要关注输入、输出、交互、绘制等功能。

1) 点击屏幕，获取控制点位置

我们的目标是：用户鼠标点击屏幕，在该处生成控制点并记录位置。

在HTML中声明画布`Canvas`

```
<canvas class="Mycanvas">
<p>One Canvas</p>
</canvas>
```

在js中进行初始化

```
const canvas = document.querySelector(".Mycanvas");
const width = canvas.width = window.innerWidth;
const height = canvas.height = window.innerHeight;
const ctx = canvas.getContext('2d');
```

调用画布监听函数，当用户单击左键时，获取坐标，将坐标放入数组中（为绘制曲线做准备），并绘制单个点



```
//绘制红色单个粗圆点
function DrawRedPoint(x, y) {
    ctx.fillStyle = 'rgb(255,0,0)';
    ctx.fillRect(x-2, y-2, 4, 4);
}

// 监听画布点击事件
canvas.addEventListener("click", function(event) {
    getMousePos(canvas, event);
});

// 获得鼠标位置并绘制单个圆点
function getMousePos(canvas, event) {
    //1
    var rect = canvas.getBoundingClientRect();
    //2
    var x = event.clientX - rect.left * (canvas.width / rect.width);
    var y = event.clientY - rect.top * (canvas.height / rect.height);
    var pos = [x,y];
    points.push(pos);
    DrawRedPoint(x,y);
}
```

2)点击按钮，生成曲线

在HTML中声明按钮*Button*

```
<button class = "button" id = "PathButton"><p>生成曲线</p></button>
```

在js中进行监听，当用户点击时，调用函数计算插值点（Tension和Gain参数通过滑块Slider实时控制，后续详细介绍），并进行绘制

```

//监听按钮
document.getElementById("PathButton").addEventListener("click", CardinalGenerate);
//生成cardinal曲线
function CardinalGenerate() {
    ctx.clearRect(0,0,width,height);
    Spline = [];
    CubicSpline(document.getElementById("GrainRange").value,
    document.getElementById("TensionRange").value/100.0);
}
//根据插值点数组绘制曲线
function DrawCardinal() {
    for (var point of points) {
        DrawRedPoint(point[0], point[1]);
    }
    ctx.beginPath();
    ctx.moveTo(Spline[0][0], Spline[0][1]);
    for (var i = 1; i < Spline.length; i++) {
        ctx.lineTo(Spline[i][0], Spline[i][1]);
    }
    ctx.stroke();
}

```

3) 调节Tension/Grain滑块， 实时控制曲线的张量

在HTML中声明滑块Slider

```

<input type="range" min="0" max="100" value="50" class="slider"
id="TensionRange">

```

在js中进行监听，每当滑块的值改变就清空画布，重新计算插值点并绘制曲线，达到实时调整的效果

```

//设置Grain滑块
var Gslider = document.getElementById("GrainRange");
var Goutput = document.getElementById("GrainShow");
Goutput.innerHTML = "Grain:" + Gslider.value;
Gslider.oninput = function() {
    Goutput.innerHTML = "Grain:" + this.value;
    ctx.clearRect(0,0,width, height);
    Spline = [];
    CubicSpline(this.value,Tslider.value / 100.0);
    if (point_show) ShowPoints();
}

```

Grain和Tension的设置完全一致，不再赘述

小车运动

使小车沿着曲线路径运动，速度可调，自动调整方向

具体步骤是：

- 导入小车png图片
- 移动到第一个控制点
- 根据速度计算每一帧移动距离
- 根据相邻两插值点的斜率计算当前小车的方向
- 调用动画函数`requestAnimationFrame/animate`



```
function LineCompute(n) {
    return (Spline[n+1][1] - Spline[n][1])/(Spline[n+1][0] - Spline[n][0]);
}
function CarMove() {
    var imgTag = new Image();
    var pos_now = 0;
    imgTag.onload = animate;
    imgTag.src = "https://s1.ax1x.com/2020/09/28/0VYJE9.md.png"; // load image
    var num = Math.ceil(document.getElementById("SpeedRange").value / 10);
    function animate() {
        ctx.clearRect(0, 0, width, height); // clear canvas
        DrawCardinal();
        ctx.save();
        var x = Spline[pos_now][0];
        var y = Spline[pos_now][1];
        var w = 100;
        var h = 100;
        ctx.translate(x+w/2, y+h/2);
        ctx.rotate(LineCompute(pos_now)*20*Math.PI/180.0);
        ctx.translate(-x-w/2, -y-h/2);
        ctx.drawImage(imgTag, x-w/2, y-h/2, w, h); // draw image at
        current position
        ctx.restore();
        pos_now += num; //use point nums to control speed
        if (pos_now < Spline.length && Spline[pos_now][0] < width)
            requestAnimationFrame(animate); // loop
    }
}
```

其他

1) 清空画布和控制点数组

```
//清空画布和坐标数组
function Clear() {
    ctx.clearRect(0,0,width, height);
    points = [];
    Spline = [];
}
```

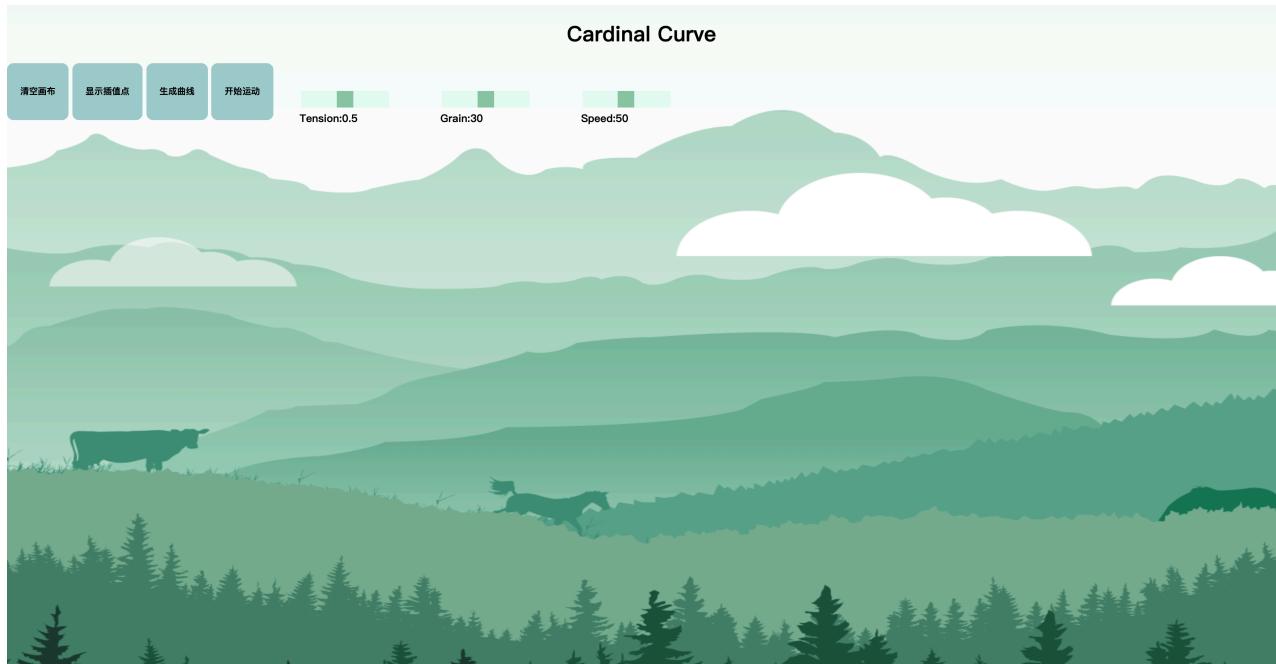
2) 显示插值点

```
function ShowPoints() {
    point_show = true;
    for (let point of Spline) {
        DrawPoint(point[0], point[1]);
    }
}
```

3) 使用CSS进行了界面美化，但不属于实验核心内容，这里不再赘述。

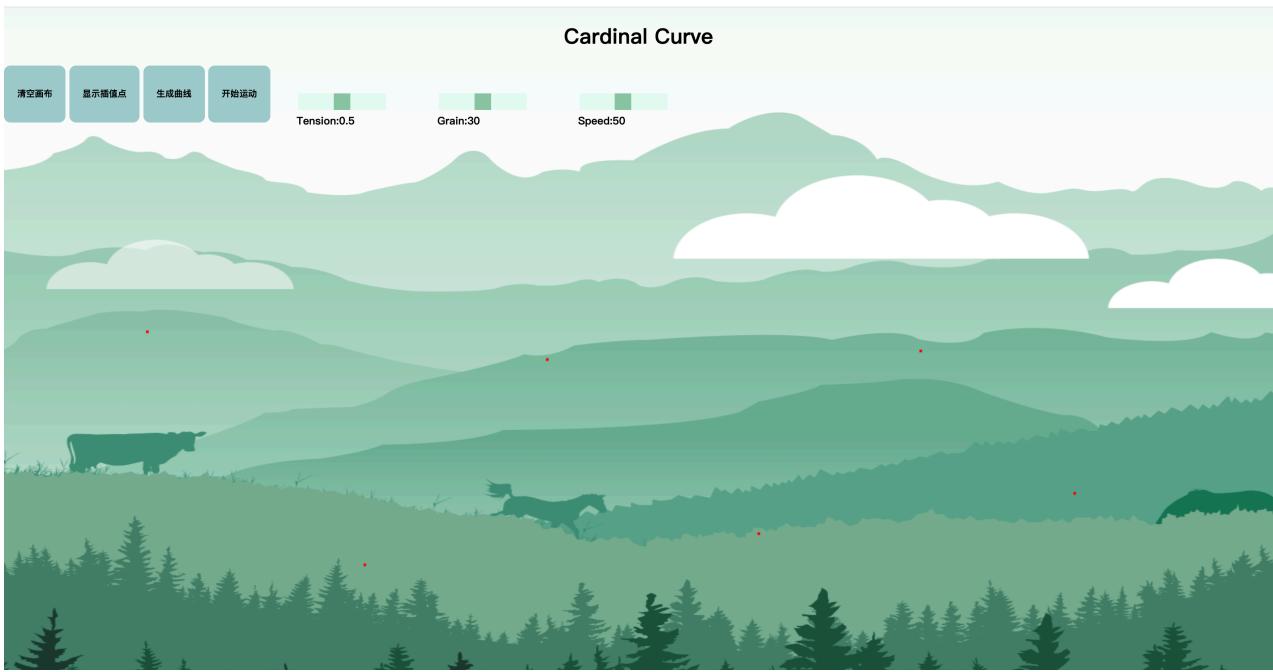
五、实验结果分析

1.主界面

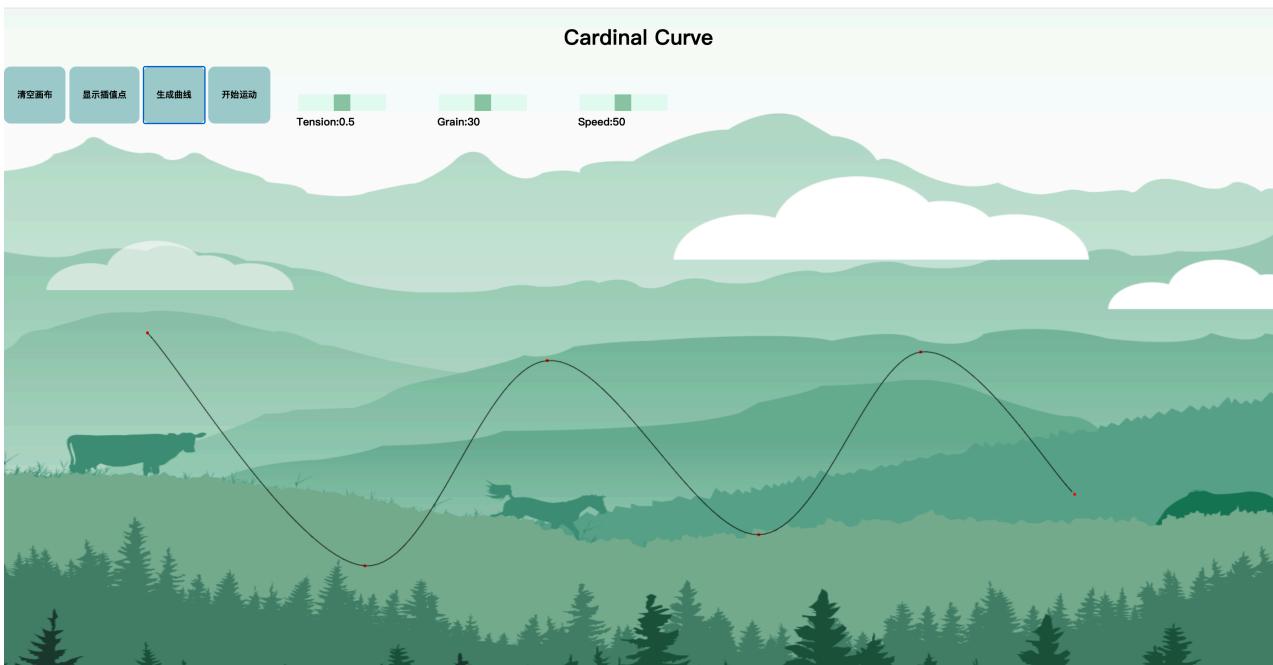


2.绘制Cardinal曲线

2.1选取控制点

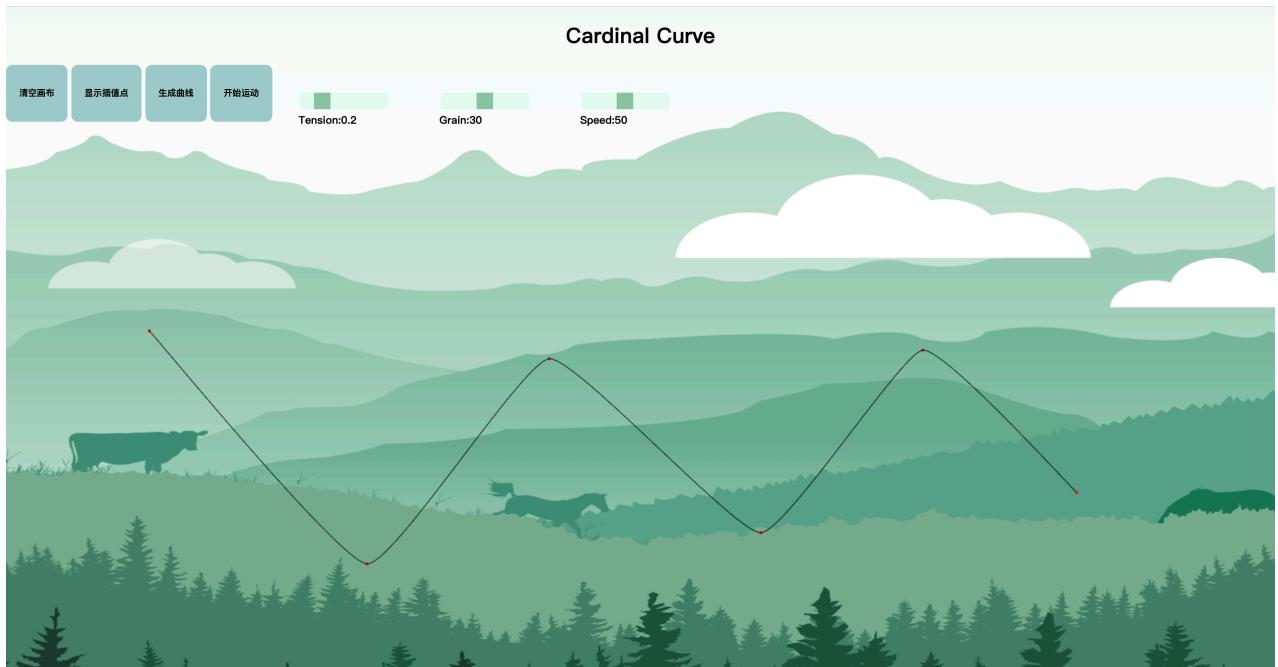


2.2生成曲线

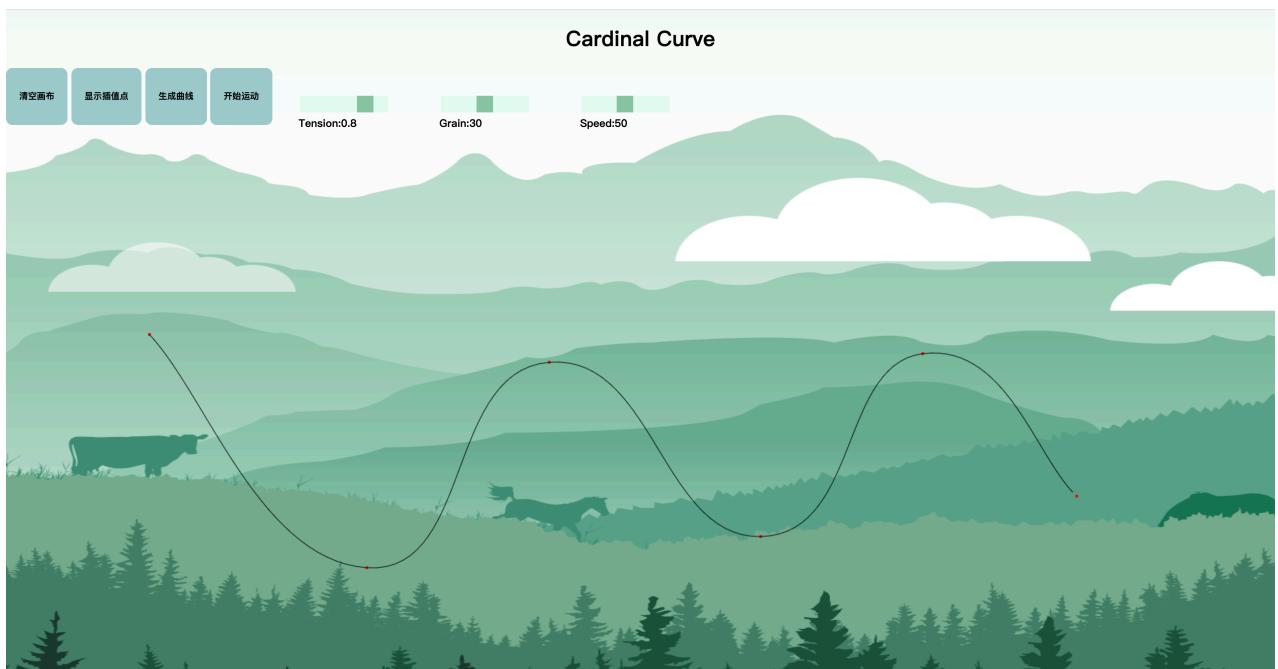


2.3修改Tension值（可使用滑块实时调整）

Tension:0.2

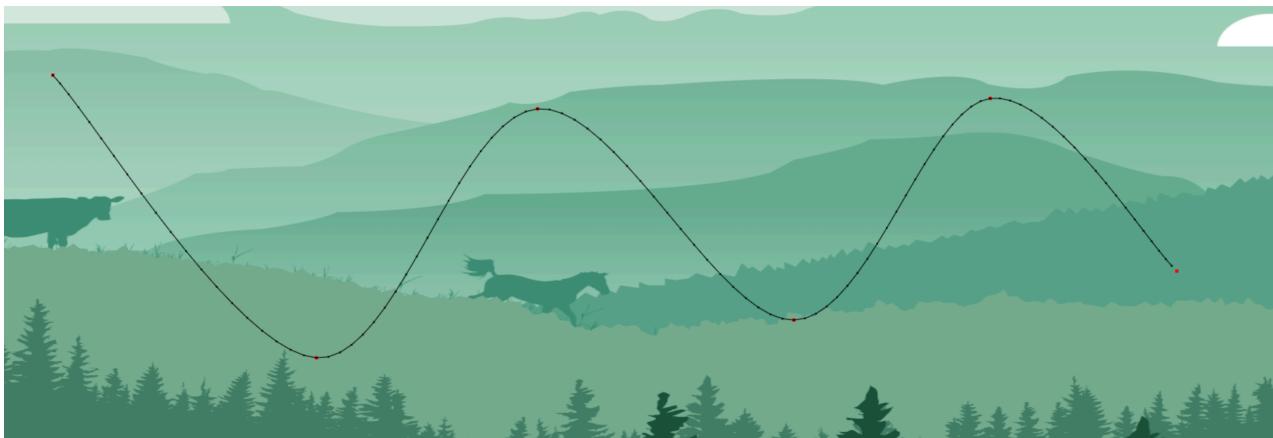


Tension:0.8



2.4 修改Grain值（通过显示插值点按钮观察）

Grain: 20



Grain:45



3. 小车动画



更多细节见演示视频

六、实验感悟

通过这次试验，我对 Cardinal 样条曲线有了进一步的了解，也对于物体沿曲线运动的算法有了更深入的了解和亲自实践。另外，借着这次机会，我自学了前端界面设计和实现，这是一项非常实用的技能。虽然目前我掌握的程度还很有限，但这为我以后更深入地学习前端和计算机动画打下了基础。

