# 一维质点运动学

曾经在2011年研究过一阵子物理引擎，但半途而废了。当时用的编程语言是C#，对应平台是已被微软放弃的Silverlight。现在准备重振旗鼓再冲击一次，用的语言是TypeScript，这样就可以在浏览器中使用这个引擎，通用性更好。

## 物理知识

质点是一个理想化物理模型，无需考虑它的大小形状，在高中物理中大多数情况中都可以把物体看成一个有质量的点，而且点也没有旋转的概念。

运动学是对物体运动的研究，并不考虑物体上的作用力，我们只关心质点的位移、速度和加速度随时间变化的规律。

在初中物理中我们就已经学过匀速直线运动，加速度为0，位移与速度关系为：

*x*t＝*x*0＋*vt*

式中*x*t表示*t*秒末的位移，*x*0表示初始位移，*v*为速度，它是一个有方向的矢量，表征物体运动的快慢，从另一个角度理解，它也是物体**单位时间中位移的改变量**，例如*v*＝5m/s向右，即表示物体每秒钟向右移动5 m的距离。

到了高中，我们又学习了匀变速直线运动，此运动中速度与加速度的关系为：

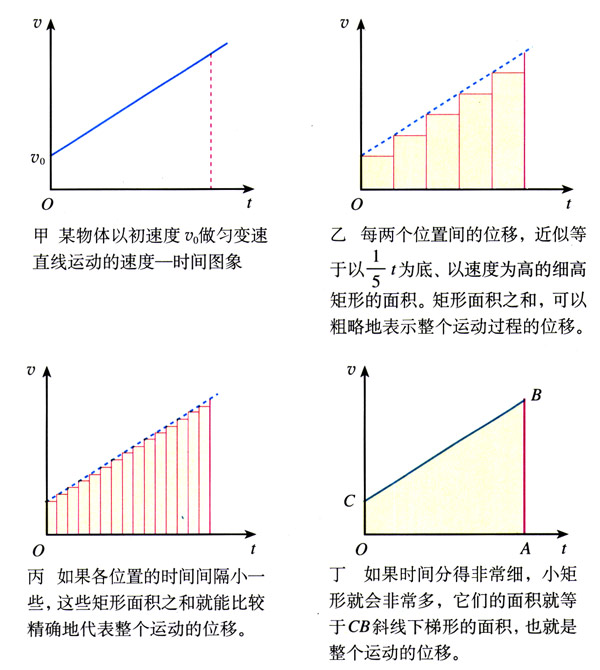
*v*t＝*v*0＋*at*

加速度*a*也是矢量，表示速度变化的快慢，从另一个角度理解，它也是**物体单位时间中速度的改变量**，例如*a*＝5 m/s2向右，表示物体的速度每秒钟在向右方向增加5 m/s。

而位移、初速度和加速度之间的关系为：

*x*＝*x*0＋*v*0*t*＋*at*2

那么上面那个公式究竟是如何得出的？高中物理教材使用的都是微积分的思想，如下图所示为人教版的推导示意图。



对于上图，不妨对应这么一个匀加速过程：*v*0=1 m/s，*a*=2m/s2，求*t*=1 s时的*x*。

对应甲图，即得到*x*的精确解，为2 m。

对应乙图，将1 s分成5个Δ*t*，且Δ*t*=0.2 s。将这五段都看出匀速直线运动，速度为该段的初速度，则物体的位移为这五段匀速运动位移之和。下面的表格显示了计算值和精确值的比较：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 步数 | *t* | *x*的计算值 | *x*的精确值 | 差值 |
| 1 | 0.20 | 0.2 | 0.24 | 0.04 |
| 2 | 0.40 | 0.48 | 0.56 | 0.08 |
| 3 | 0.60 | 0.84 | 0.96 | 0.12 |
| 4 | 0.80 | 1.28 | 1.44 | 0.16 |
| 5 | 1.00 | 1.8 | 2 | 0.2 |

在这个例子中，计算值比精确值要小，且随着时间的增加，误差越来越大。

那将Δ*t*（在数值分析这门学科中，Δ*t*又叫做步长，用*h*表示）减小会如何呢？对应丙图，这次取Δ*t*=0.02，相应的表格如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 步数 | *t* | *x*的计算值 | *x*的精确值 | 差值 |
| 1 | 0.02 | 0.02 | 0.0204 | 0.0004 |
| … | … | … | … | … |
| 20 | 0.40 | 0.552 | 0.56 | 0.008 |
| … | … | … | … | … |
| 30 | 0.60 | 0.948 | 0.96 | 0.012 |
| … | … | … | … | … |
| 40 | 0.80 | 1.424 | 1.44 | 0.016 |
| … | … | … | … | … |
| 50 | 1.00 | 1.98 | 2 | 0.02 |

这次误差小得多，可以想象，若Δ*t*→0，那么计算出的值就应该等于精确值，如图丁所示。

在计算机上不可能做到Δ*t*→0，通常网页的刷新频率为1/60s，物理引擎的刷新率设置为1/30s通常就能满足要求了。

总结下来，在电脑上进行运动的仿真计算，**需要将时间分割成长度为Δ*t*一小段，将这每一小段都看成匀速直线运动，由*x*i=*v*iΔ*t*计算出每一小段的位移，最后将这些位移相加**即可，伪代码如下：

void Update(dt)

{

v+=a\*dt;

x+=v\*dt;

}

物体的运动情况归根结底是由它的初始状态（初始位移和初速度）和加速度（严格地说，应该是由受力情况决定的，这会在后面的质点动力学中进行讨论）决定的，因此只需设置物体的初始位置、初速度和加速度，计算机程序就会自动算出任意时刻物体的位置了。

## 实现StunPhysics引擎

理解了背后的物理知识，下面我们就开始实现一个史上最简陋的2D物理引擎，我把它命名为StunPhysics。主要参考了[Box2D.ts](https://github.com/flyover/box2d.ts)，[matter.js](http://brm.io/matter-js/)和[p2.js](https://github.com/schteppe/p2.js)。使用的编辑器为Visual Studio Code。

如下图所示，可以把物理引擎对应现实世界的台球桌，即我们的物理世界（World），而台球上的桌球就是处于这个世界的物体（Body），我们只要对物体施加“第一推动”，接下去物理引擎就可以像拉普拉斯魔那样预测出物体的未来。



World（世界）

Body（物体）

*x*

*y*

*O*

因此首先创建我们的世界——World类，代码如下：

import { Body } from "./objects/Body";

export class World {

bodies: Array<Body>;

constructor() {

this.bodies = new Array<Body>();

}

addBody(body: Body) {

this.bodies.push(body);

}

step(dt: number) {

for (let i: number=0; i < this.bodies.length; i++){

this.bodies[i].Integrate(dt);

}

}

}

这个类管理一个Body的集合，在它的step方法中更新集合中所有Body的速度和位移。

Body类表示一个物体，目前表示一个质点。这个类保存了物体的位置、速度和加速度。最关键的代码位于积分运算的Integrate方法中：

import { World } from "../World";

export class Body {

x: number = 0;

velocity: number = 0;

acceleration: number = 0;

Integrate(dt: number) {

this.velocity += this.acceleration \* dt;

this.x += this.velocity \* dt;

}

}

好了，至此，史上最简陋的物理引擎制作完毕，物理引擎就相当于一个黑箱，输入初位置、初速度，目前还需要输入加速度，它就可以计算出下一帧的新状态，如下图所示：

物理引擎

*v*old

*v*new

*v*new

*s*old

*s*old

*s*new

*v*new

*a*

由加速度求得新的速度

由新的速度求得新的位置

最后为了调试方便，还添加了一个绘图类Render（对物理引擎来说它不是必须的），它可以在屏幕上绘制圆形代表Body。代码如下：

import { World } from "../World";

import { Body } from "../objects/Body";

export class Render {

ctx: CanvasRenderingContext2D;

constructor(ctx: CanvasRenderingContext2D) {

this.ctx = ctx;

}

draw(world: World) {

this.ctx.clearRect(0, 0, this.ctx.canvas.width, this.ctx.canvas.height);

this.ctx.save();

for (let i: number=0; i < world.bodies.length; i++) {

const body: Body = world.bodies[i];

this.drawSolidCircle(body.x, 40, 20);

}

this.ctx.restore();

}

private drawSolidCircle(x: number, y: number, radius: number) {

const ctx: CanvasRenderingContext2D = this.ctx;

if (ctx) {

const cx: number = x;

const cy: number = y;

ctx.beginPath();

ctx.arc(cx, cy, radius, 0, Math.PI \* 2, true);

ctx.moveTo(cx, cy);

ctx.lineTo((cx + radius), cy);

ctx.fillStyle = "rgba(255,0,0,0.5)";

ctx.fill();

ctx.strokeStyle = "rgb(255,0,0)";

ctx.stroke();

}

};

}

## 应用

下面我们使用这个引擎模拟一下几个常见的物理情境。

### 1、匀变速直线运动

根据物理知识，首先需要设定物体的初始条件：*x*0、*v*0，在下面的例子中，小球被放置在画布偏左侧，坐标值为100 m（以1个像素为1 m），初速度设置为25 m/s，匀变速直线运动的加速度是个恒量，可以设置为25 m/s2。

现在可以算一下小球位移600 m所需的时间，由运动学公式：

*x*＝*x*0＋*v*0*t*＋*at*2

可求得*t*=6 s。

下面用物理引擎来计算一下这个时间，代码如下：

import { Render } from "../../src/render/Render";

import { World } from "../../src/World";

import { Body } from "../../src/objects/Body";

export class test {

world: World;

circleBody: Body

render: Render;

canvas: HTMLCanvasElement;

public constructor() {

this.canvas = <HTMLCanvasElement>document.getElementById('canvas');

this.render = new Render(this.canvas.getContext("2d"));

// 初始化物理引擎

this.world = new World();

// 初始化物体，并设置它的位置、速度和加速度

this.circleBody = new Body();

this.circleBody.x = 100;

this.circleBody.velocity = 25;

this.circleBody.acceleration = 25;

this.world.addBody(this.circleBody);

this.Update();

}

private previousTime: number; // 上一帧的开始时刻

private elapsedTime: number; // 每帧流逝的时间（毫秒）

private totalTime: number = 0; // 程序运行的总时间

Update() {

requestAnimationFrame(() => this.Update());

const time: number = performance.now();

this.elapsedTime = this.previousTime ? (time - this.previousTime) / 1000 : 0;

this.previousTime = time;

if (this.circleBody.x > 700) {

return;

}

if (this.elapsedTime > 0) {

this.world.step(this.elapsedTime);

};

this.render.draw(this.world);

};

}

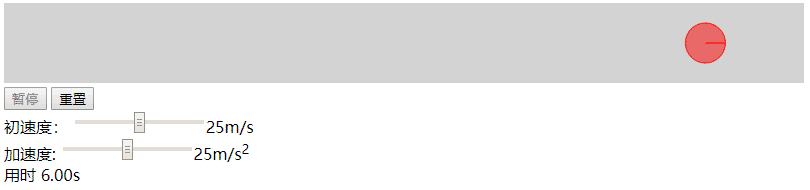
window.onload = () => {

var main: test = new test();

main.start();

}

程序截图如下（由于又增加了一些UI控件，因此其背后的代码比上面的要复杂，详细的代码可参见文章底部的链接）：



### 2、简谐振动

首先还是需要设定初始条件*x*0、*v*0，这次将小球放置在画布中央，坐标值为400 m，初速度设置为200 m/s。

简谐振动应满足：*F*回=-*kx*，再根据牛顿第二定律，简谐振动的加速度表达式为：

*a*=－*x*

式中*k*为弹簧的劲度系数，*m*为振子的质量，在此程序中设置为：*k*=2 N/m，*m*=2 kg。由物理知识可知，此弹簧振子的周期*T*=2π≈6.28 s，振幅*A*=*v*0=200 m。

可以通过下面的程序验证一下：

import { Render } from "../../src/render/Render";

import { World } from "../../src/World";

import { Body } from "../../src/objects/Body";

export class test {

world: World;

circleBody: Body

render: Render;

canvas: HTMLCanvasElement;

// 设置弹簧劲度系数和振子质量

k: number = 2;

m: number = 2;

public constructor() {

this.canvas = <HTMLCanvasElement>document.getElementById('canvas');

this.render = new Render(this.canvas.getContext("2d"));

// 初始化引擎

this.world = new World();

// 设置物体的初位置、初速度

this.circleBody = new Body();

this.circleBody.x = 400;

this.circleBody.velocity = 200;

this.world.addBody(this.circleBody);

this.Update();

};

private previousTime: number; // 上一帧的开始时刻

private elapsedTime: number; // 每帧流逝的时间（毫秒）

Update() {

requestAnimationFrame(() => this.Update());

const time: number = performance.now();

this.elapsedTime = this.previousTime ? (time - this.previousTime) / 1000 : 0;

this.previousTime = time;

if (this.elapsedTime > 0) {

// 设置物体的加速度

this.circleBody.acceleration = -(this.k / this.m) \* (this.circleBody.x - 400);

this.world.step(this.elapsedTime);

};

this.render.draw(this.world);

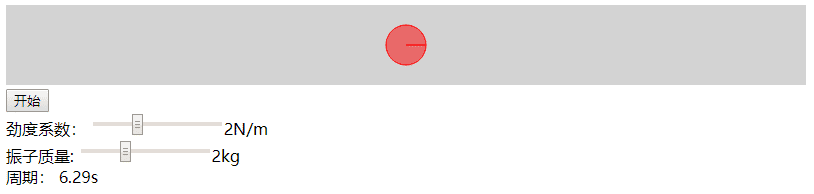
};

}

window.onload = () => {

var main: test = new test();

}



### 3、变加速直线运动

下面的这道题目为2011年上海高考题，原题如下：

电阻可忽略的光滑平行金属导轨长*s*＝1.15 m，两导轨间距*L*＝0.75 m，导轨倾角为30°，导轨上端ab接一阻值*R*＝1.5 Ω的电阻，磁感应强度*B*＝0.8 T的匀强磁场垂直轨道平面向上。阻值*r*＝0.5 Ω，质量*m*＝0.2 kg的金属棒与轨道垂直且接触良好，从轨道上端ab处由静止开始下滑至底端，在此过程中金属棒产生的焦耳热*Q*r＝0.1 J。（取*g*＝10m/s2）求：

a

*B*

30°

b

*R*

（3）金属棒下滑的最大速度*v*m。

【讨论】这里关注的是第（3）小问，最大速度*v*m的答案为2.74 m/s。限于高中生的知识局限，只能用动能定理解决，并需要告知金属棒产生的热量。

【解】*Q*＝*Q*R＋*Q*r＝0.4 J

*mgs*sin30°－*Q*＝*mv*m2

*v*m＝＝ m/s＝2.74 m/s

而如果运用微积分知识是可以直接求得速度大小的。

若只需求得*s*与*v*的函数关系，解法如下：

由牛顿第二定律可知：

*mg*sin*θ*－＝*ma*

代入数据可得*v*和*a*的关系式：

*a*＝5－0.9*v*

对应的微分方程为：

＝＝

使用分离变量法解此方程：

*ds*＝*dv*，初始条件为*s*|*v*=0＝0

*s*＝（－0.9*v*－5ln（－0.9*v*＋5）＋5ln5）

将*s*＝1.15 m代入以上函数可解得：*v*＝2.74 m/s。

若还需要求时间*t*，可以写出以下微分方程

＝*a*＝5－0.9*v*，*v*|*t*＝0＝0

解得

*t*＝[ln5－ln(-0.9*v*+5)]

或

*v*＝（1－*e*-0.9*t*）

将*v*＝2.74 m/s代入以上函数，可求得*t*＝0.75515 s。

进一步利用以下微分方程，还可以求出*s*与*t*的函数关系：

＝*v*＝（1－*e*-0.9*t*），*s*|*t*＝0＝0

解得

*s*＝

将*s*＝1.15 m代入以上函数，可求得*t*＝0.75484 s。

下面用物理引擎模拟这个过程并验证结果，代码如下：

import { Render } from "../../src/render/Render";

import { World } from "../../src/World";

import { Body } from "../../src/objects/Body";

export class test {

world: World;

circleBody: Body;

public constructor() {

this.world = new World();

this.circleBody = new Body();

this.world.addBody(this.circleBody);

this.Update();

};

private previousTime: number; // 上一帧的开始时刻

private elapsedTime: number; // 每帧流逝的时间（毫秒）

Update() {

requestAnimationFrame(() => this.Update());

const time: number = performance.now();

this.elapsedTime = this.previousTime ? (time - this.previousTime) / 1000 : 0;

this.previousTime = time;

if (this.circleBody.x >= 1.15) {

return;

}

if (this.elapsedTime > 0) {

this.circleBody.acceleration = 5 - 0.9 \* this.circleBody.velocity;

this.world.step(this.elapsedTime);

};

};

}

window.onload = () => {

var main: test = new test();

}

模拟的结果*t*≈0.750 s，*v*≈2.70 m/s，都要比精确值来得小。

最后，以上代码都托管在：<https://github.com/fjphysics/StunPhysics/>。