JS Reverse Engineering for Fun

by Nicu Micleușanu

WHAT?

Reverse Engineering - (from Wikipedia) process of extracting knowledge or design information from anything man-made and reproducing it or reproducing anything based on the extracted information.

The process often involves disassembling something and analyzing its components and workings in detail.

WHY?

Learning Purpose. (Learn something new)

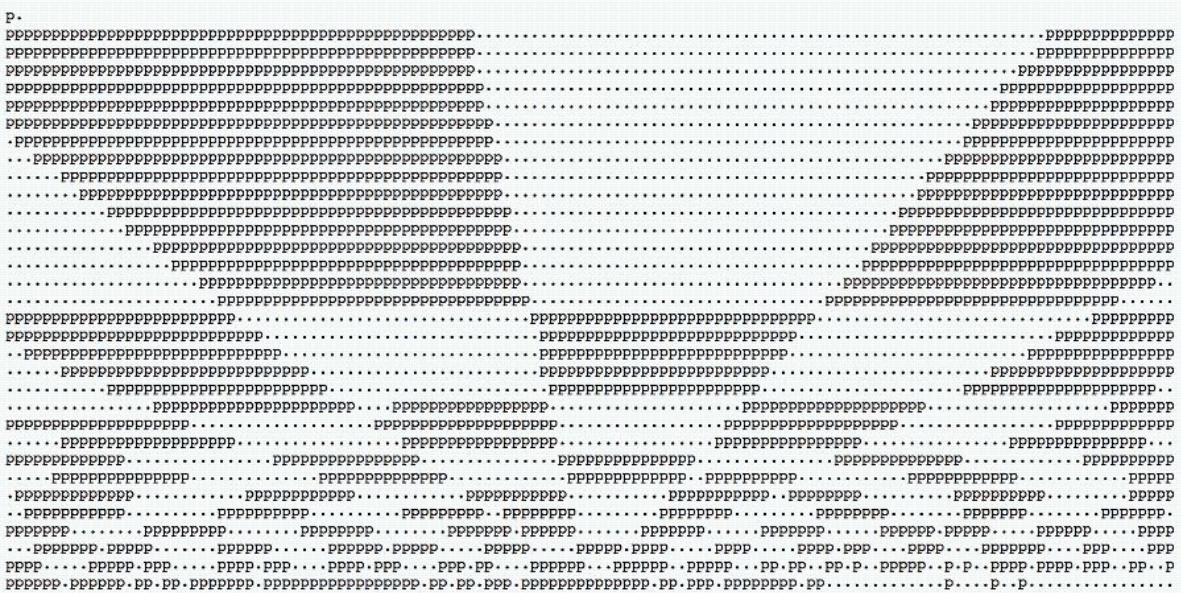
Cloning. (Create a duplicate with some new features)

Code update. (Maybe somebody lost the source)

Bug Fixing. (Fix issues in some legacy code)

Fun. (I just enjoy it)

Reverse Engineering one line of code



http://www.p01.org/

My Story

Programming Puzzles & Code Golf Questions Tags Us

Sprocket Science: Animating a Chain Drive System

87

24

The goal of this challenge is to produce an animation of a **chain drive** system, comprised of a set of **sprocket gears** connected together by a **chain**.

General Requirements

Your program will be given a **list of sprockets**, specified as `(x, y, radius)` triplets. The **resulting chain drive system** is comprised of these sprockets, connected together by a **closed taut chain** passing over each of them, **in order**. Your goal is to produce an **infinitely looping animation**, showing the system in motion. For example, given the input

```
(0, 0, 16),  (100, 0, 16),  (100, 100, 12),  (50, 50, 24),  (0, 100, 12)
```

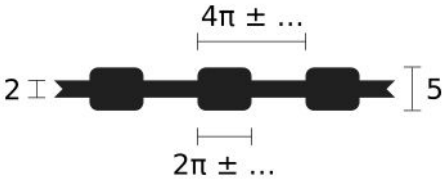
, the output should look something like

https://codegolf.stackexchange.com/questions/64764/sprocket-science-animating-a-chain-drive-system

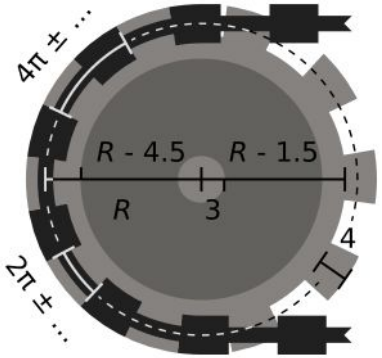
Challenge requirements

Visual Requirements

The chain should consist of a series of **links** of alternating widths. The width of the **narrow link** should be about 2, and the width of the **wide link** should be about 5. The **length** of both types of links should be about equal. The **period** of the chain, that is, the total length of a wide/narrow pair of links, should be the closest number to 4π that fits an integer number of times in the length of the chain. For example, if the length of the chain is 1,000, then its period should be 12.5, which is the closest number to 4π (12.566...) that fits an integer number of times (80) in 1,000. It's important for the period to fit an integer number of times in the chain's length, so that there are no artifacts at the point where the chain wraps around.



A sprocket of radius ***R*** should consist of three concentric parts: a central **axle**, which should be a circle of radius about 3; the **sprocket's body**, around the axle, which should be a circle of radius about *R* - 4.5; and the **sprocket's rim**, around the body, which should be a circle of radius about *R* - 1.5. The rim should also contain the **sprocket's teeth**, which should have a width of about 4; the teeth's size and spacing should match the the sizes of the chain links, so that they enmesh neatly.



My Target

2 Answers

active oldest votes

38

JavaScript (ES6), ~~2557~~ ~~1915~~ ~~1897~~ 1681 bytes

This isn't super-duper *golfed* really; it's minified - partly by hand - but that's nothing special. It could no doubt be shorter if I'd golfed it more before minifying, but I've spent (more than) enough time on this already.

Edit: Ok, so I spent more time on it and golfed the code more before minifying (very manually this time). The code's still using the same approach and overall structure, but even so I still ended up saving 642 bytes. Not too shabby, if I do say so myself. Probably missed some byte-saving opportunities, but at this point even I'm not sure how it works anymore. The only thing that's different in terms of output, is that it now uses slightly different colors that could be written more tersely.

Edit 2 (much later): Saved 18 bytes. Thanks to ConorO'Brien in the comments for pointing out the blindingly obvious that I'd totally missed.

Edit 3: So, I figured I'd reverse engineer my own code, because, frankly, I couldn't remember how I'd done it, and I lost the ungolfed versions. So I went through, and lo and behold found another 316 bytes to save by restructuring and doing some micro-golfing.

```
R=g=>{with(Math){V=(x,y,o)=>o={x,y,l:sqrt(x*x+y*y),a:v=>V(x+v.x,y+v.y),s:v=>o.a(v.
```

The function above appends an SVG element (including animations) to the document. E.g. to display the 2nd test case:

```
R([[100, 100, 60],  [220, 100, 14]]);
```

Seems to work a treat - at least here in Chrome.

Warning!

A Lot of Code Ahead

Let the game begin

```
R=g=>{with(Math){V=(x,y,o)=>o={x,y,l:sqrt(x*x+y*y),a:v=>V(x+v.x,y+v.y),s:v=>o.a(v.m(-1)),
m:f=>V(x*f,y*f),t:r=>V(x*cos(r)-y*sin(r),x*sin(r)+y*cos(r)),c:v=>x*v.y-y*v.x,toString:_=>x+','+y};
a='appendChild',b='setAttribute';S=(e,a)=>Object.keys(a).map(n=>e[b](n,a[n]))&&e;
T=(t,a)=>S(k.createElementNS('http://www.w3.org/2000/svg',t),a);C=(e,a)=>S(e.cloneNode(),
a);P=a=>T('path',(a.fill='none',a));w=h=-(x=y=1/0);G=g.map((a,g)=>(g=V(...a))&&(u=(g.r=a[2])+5,
x=min(x,g.x-u),y=min(y,g.y-u),w=max(w,g.x+u),h=max(h,g.y+u))&&g);k=document;I=k[a].bind(k.body[a](T('svg',
{width:w-x,height:h-y}))[a](T('g',{transform:`translate(${-x},${h})scale(1,-1)`})));
L=(c)=>(h=G.length)&&G.map((g,i)=>c(G[i],G[i?i-1:h-1],G[(i+1)%h]))&&L;l='';L((g,
p,n)=>g.f=p.s(g).c(n.s(g))>0)((g,a,n)=>{d=g.s(n),y=x=1/d.l;g.f!=n.f?(a=asin((g.r+n.r)*x),
g.f?(x=-x,a=-a):(y=-y)):(a=asin((g.r-n.r)*x),g.f&&(x=y=-x,a=-a));t=d.t(a+PI/2);
g.o=t.m(x*g.r).a(g);n.i=t.m(y*n.r).a(n)})((g,p,n)=>{z='#888';d=(l,s,e)=>`A${g.r},${g.r} 0 ${1*l},${1*s} ${e}`;
e=(f,r)=>T('circle',{cx:g.x,cy:g.y,r,fill:f});g.k=p.o.s(n.i).l<g.i.s(g.o).l;w=d(g.k,
!g.f,g.o);g.j=`${w}L${n.i}`;l+=g.j;I(e(z,g.r-1.5));g.g=I(P({d:`M${g.i}${w}${d(!g.k,!g.f,g.i)}`,
stroke:z,'stroke-width':5}));g.h=I(C(g.g,{d:`M${g.i}${g.j}`,stroke:'#222'}));I(e('#666',
g.r-4.5));I(e(z,3))});t=e=>e.getTotalLength(),u='stroke-dasharray',v='stroke-dashoffset',
f=G[0];l=I(C(f.h,{d:'M'+f.i+l,'stroke-width':2}));s=f.w=t(l)/round(t(l)/(4*PI))/2;
X=8*s;Y=f.v=0;L((g,p)=>{g.g[b](u,s);g.h[b](u,s);g==f||(g.w=p.w+t(p.h),g.v=p.v+t(p.h));
g.g[b](v,g.w);g.h[b](v,g.v);g.h[a](C(g.g[a](T('animate',{attributeName:v,from:g.w+X,
to:g.w+Y,repeatCount:'indefinite',dur:'1s'})),{from:g.v+X,to:g.v+Y}))})}}
```

Beautify it

```
R = g => {
    with(Math) {
        V = (x, y, o) => o = {
            x,
            y,
            l: sqrt(x * x + y * y),
            a: v => V(x + v.x, y + v.y),
            s: v => o.a(v.m(-1)),
            m: f => V(x * f, y * f),
            t: r => V(x * cos(r) - y * sin(r), x * sin(r) + y * cos(r)),
            c: v => x * v.y - y * v.x,
            toString: _ => x + ',' + y
        };
        a = 'appendChild', b = 'setAttribute';
        S = (e, a) => Object.keys(a).map(n => e[b](n, a[n])) && e;
        T = (t, a) => S(k.createElementNS('http://www.w3.org/2000/svg', t), a);
        C = (e, a) => S(e.cloneNode(), a);
        P = a => T('path', (a.fill = 'none', a));
        w = h = -(x = y = 1 / 0);
        G = g.map((a, g) => (g = V(...a)) && (u = (g.r = a[2]) + 5, x = min(x, g.x - u),
            y = min(y, g.y - u), w = max(w, g.x + u), h = max(h, g.y + u)) && g);
        k = document;
        I = k[a].bind(k.body[a](T('svg', {
            width: w - x,
            height: h - y
        }))[a](T('g', {
            transform: `translate(${-x},${h})scale(1,-1)`
        })));
        L = (c) => (h = G.length) && G.map((g, i) => c(G[i], G[i ? i - 1 : h - 1], G[(i + 1) % h])) && L;
        l = '';
```

http://jsbeautifier.org/

index.html

```
<!doctype html>
<html>
    <head>
        <title>Chain Drive System</title>
    </head>
    <body>
        <script src="script.js"></script>
        <script>
            R([[100, 100, 60],  [220, 100, 14]]);
        </script>
    </body>
</html>
```

Remove "with" statement

```
R = g => {
    with(Math) {
        V = (x, y, o) => o = {
            x,
            y,
            l: sqrt(x * x + y * y), // Math.sqrt()
            a: v => V(x + v.x, y + v.y),
            s: v => o.a(v.m(-1)),
            m: f => V(x * f, y * f),
            t: r => V(x * cos(r) - y * sin(r), x * sin(r) + y * cos(r)), // Math.cos(), Math.sin()
            c: v => x * v.y - y * v.x,
            toString: _ => x + ',' + y
        };
```

Replace shortcuts

```
k = document;
a = 'appendChild';
b = 'setAttribute';

S = (e, a) => Object.keys(a).map(n => e[b](n, a[n])) && e;
T = (t, a) => S(k.createElementNS('http://www.w3.org/2000/svg', t), a);
C = (e, a) => S(e.cloneNode(), a);
P = a => T('path', (a.fill = 'none', a));
```

setAttributes()

```
S = (e, a) => Object.keys(a).map(n => e[b](n, a[n])) && e;

////////////////////////////////////////////////////////////

const setAttributes = (element, attributes) => {

    Object.keys(attributes).forEach((attribute) => {
        element.setAttribute(attribute, attributes[attribute]);
    });
};
```

createElement()
cloneElement()

```
T = (t, a) => S(k.createElementNS('http://www.w3.org/2000/svg', t), a);

////////////////////////////////////////////////////////////

const createElement = (tagName, attributes) => {

    const element = document.createElementNS('http://www.w3.org/2000/svg', tagName);

    setAttributes(element, attributes);

    return element;
};
```

```
C = (e, a) => S(e.cloneNode(), a);

////////////////////////////////////////////////////////////

const cloneElement = (element, attributes) => {

    const clone = element.cloneNode();

    setAttributes(clone, attributes);

    return clone;
};
```

createPath()

```
P = a => T('path', (a.fill = 'none', a));

////////////////////////////////////////////////////////////

const createPath = (attributes) => {

    return createElement('path', {
        ...attributes,
        fill: 'none'
    });
};
```

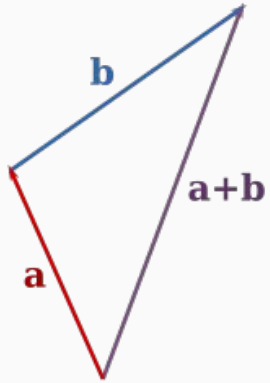
Guess what's this :D

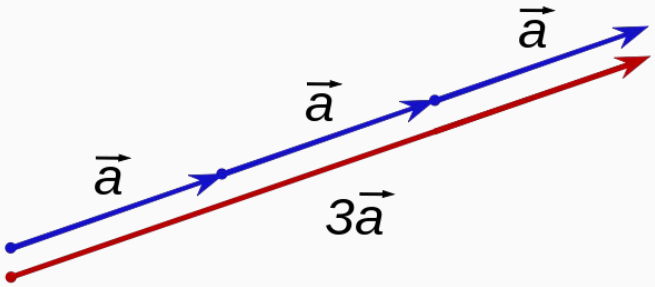
```
V = (x, y, o) => o = {
    x,
    y,
    l: Math.sqrt(x * x + y * y),
    a: v => V(x + v.x, y + v.y),
    s: v => o.a(v.m(-1)),
    m: f => V(x * f, y * f),
    t: r => V(x * Math.cos(r) - y * Math.sin(r), x * Math.sin(r) + y * Math.cos(r)),
    c: v => x * v.y - y * v.x,
    toString: _ => x + ',' + y
};
```

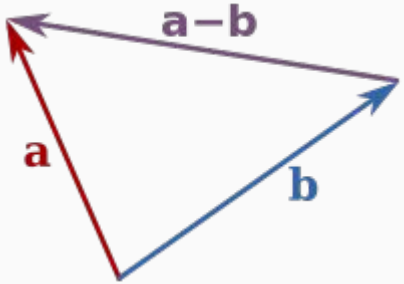
Vector Class

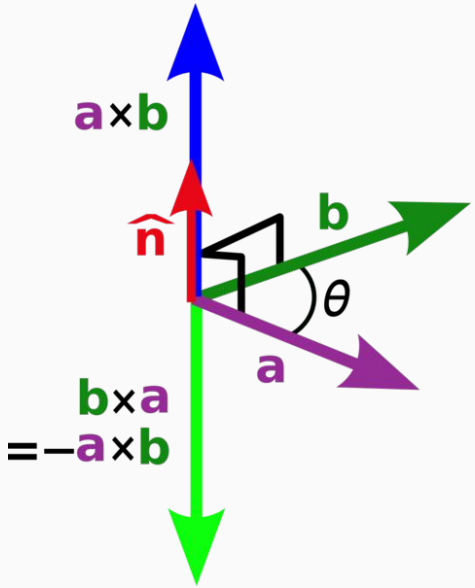
```
class Vector {

    constructor(x, y) {
        this.x = x;
        this.y = y;
        this.length = Math.sqrt(x * x + y * y);
    }

    add(vector) {

        return new Vector(this.x + vector.x, this.y + vector.y);
    }

    subtract(vector) {

        return new Vector(this.x - vector.x, this.y - vector.y);
    }

    multiply(scalar) {

        return new Vector(this.x * scalar, this.y * scalar);
    }

    rotate(radians) {

        const cos = Math.cos(radians);
        const sin = Math.sin(radians);

        return new Vector(this.x * cos - this.y * sin, this.x * sin + this.y * cos);
    }

    cross(vector) {

        return this.x * vector.y - this.y * vector.x;
    }

    toString() {

        return `${this.x},${this.y}`;
    }
}
```

Some vector algebra









Let's prepare viewport

```
w = h = -(x = y = 1 / 0);

G = data.map((a, g) => (g = new Vector(...a)) &&
    (u = (g.r = a[2]) + 5,
        x = Math.min(x, g.x - u),
        y = Math.min(y, g.y - u),
        w = Math.max(w, g.x + u),
        h = Math.max(h, g.y + u)) &&
    g);

////////////////////////////////////////////////////////

let x = Infinity;
let y = Infinity;
let w = -Infinity;
let h = -Infinity;

const gears = data.map((params) => {

    const gear = new Vector(...params);

    gear.r = params[2];

    const unit = params[2] + 5;

    x = Math.min(x, gear.x - unit);
    y = Math.min(y, gear.y - unit);
    w = Math.max(w, gear.x + unit);
    h = Math.max(h, gear.y + unit);

    return gear;
});
```

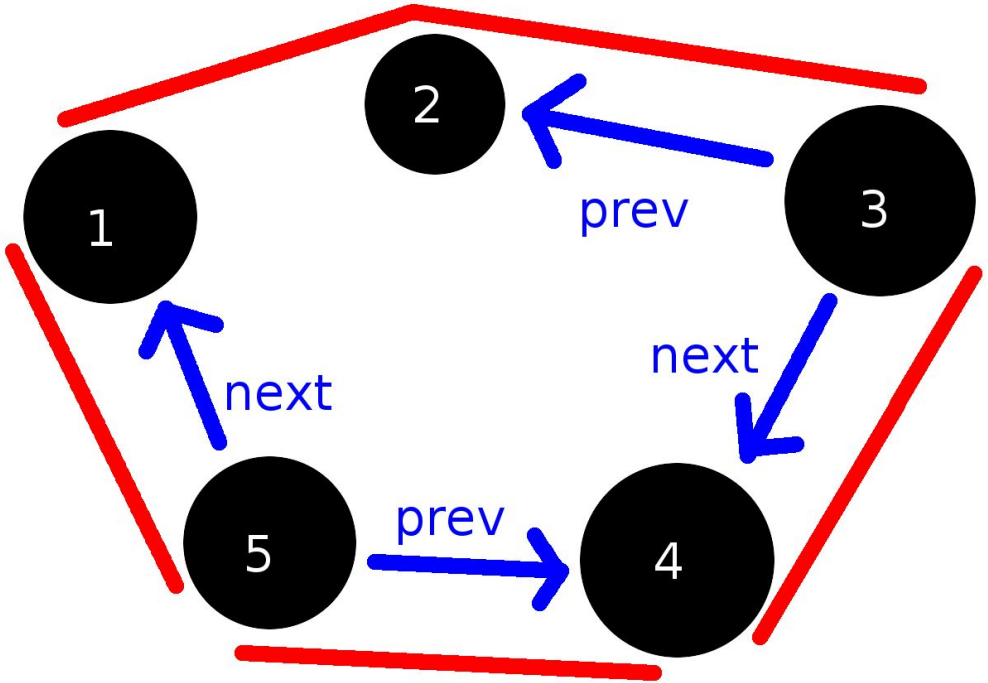
Set viewport

```
I = document.appendChild.bind(document.body.appendChild(createElement('svg', {
    width: w - x,
    height: h - y
})).appendChild(createElement('g', {
    transform: `translate(${-x},${h})scale(1,-1)`
})));

////////////////////////////////////////////////////////

const svg = createElement('svg', {
    width: w,
    height: h,
    viewBox: `${x} ${y} ${w} ${h}`,
    transform: `scale(1,-1)`
});

document.body.appendChild(svg);

// I = svg.appendChild.bind(svg);
```

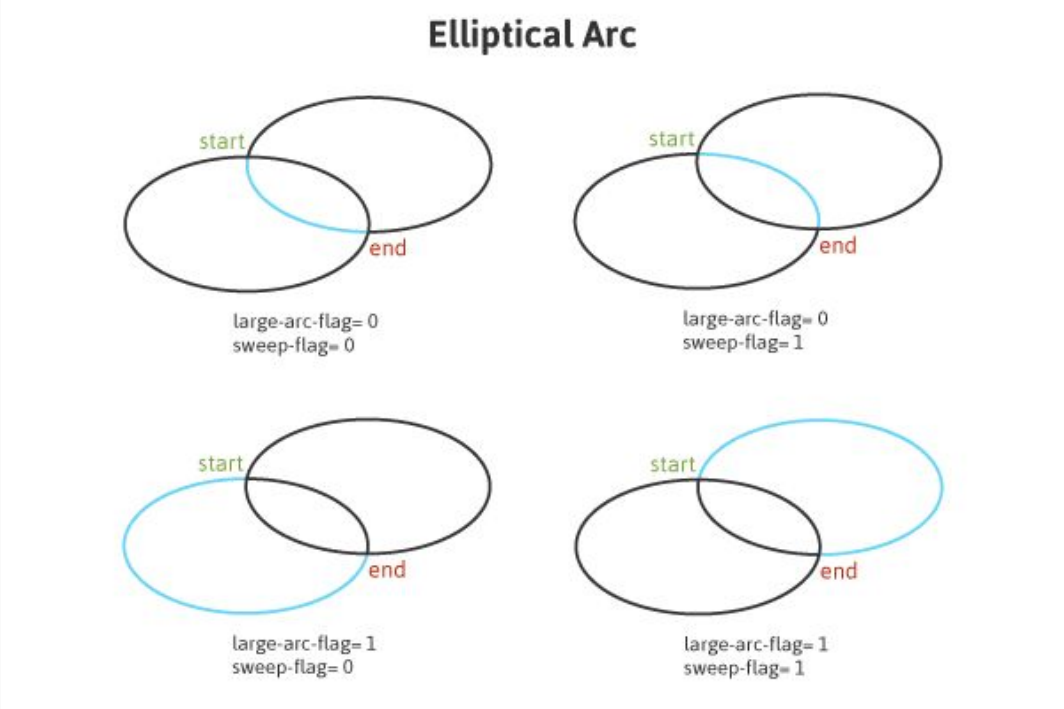
The Loop

```
L = (c) => (h = gears.length) &&
    gears.map((g, i) => c(gears[i], gears[i ? i - 1 : h - 1], gears[(i + 1) % h])) &&
    L;

//////////////////////////////////////////////////////////

const loop = (callback) => {

    const length = gears.length;

    gears.forEach((gear, index) => {

        const prevGear = gears[(length + index - 1) % length];
        const nextGear = gears[(index + 1) % length];

        callback(gear, prevGear, nextGear);
    });
};
```

Why this kind of loop?



SVG arc drawing



https://css-tricks.com/svg-path-syntax-illustrated-guide/

Arc description

```
const arcDescription = (radius, largeArcFlag, sweepFlag, endVector) => {

    return `A${radius} ${radius} 0 ${+largeArcFlag} ${+sweepFlag} ${endVector}`;
};
```

Gear Class

```
class Gear {

    constructor(x, y, radius) {
        this.x = x;
        this.y = y;
        this.radius = radius;
    }

    getVector() {

        return new Vector(this.x, this.y);
    }
}
```

Calculate gears sweep

```
loopGears(gears, (gear, prevGear, nextGear) => {

    const gearVector = gear.getVector();
    const prevGearVector = prevGear.getVector().subtract(gearVector);
    const nextGearVector = nextGear.getVector().subtract(gearVector);

    gear.sweep = (prevGearVector.cross(nextGearVector) > 0);
});
```

Calculate gears chain input and output

```
loopGears(gears, (gear, prevGear, nextGear) => {

    const diffVector = gear.getVector().subtract(nextGear.getVector());

    let angle = 0;
    let x = 1 / diffVector.length;
    let y = x;

    if (gear.sweep === nextGear.sweep) {

        angle = Math.asin((gear.radius - nextGear.radius) * x);

        if (gear.sweep) {
            x = -x;
            y = -y;
            angle = -angle;
        }
    } else {

        angle = Math.asin((gear.radius + nextGear.radius) * x);

        if (gear.sweep) {
            x = -x;
            angle = -angle;
        } else {
            y = -y;
        }
    }

    const perpendicularVector = diffVector.rotate(angle + Math.PI / 2);

    gear.out = perpendicularVector.multiply(x * gear.radius).add(gear.getVector());
    nextGear.in = perpendicularVector.multiply(y * nextGear.radius).add(nextGear.getVector());
});
```

Draw gears

```
loopGears(gears, (gear, prevGear, nextGear) => {

    const largeArcFlag = (prevGear.out.subtract(nextGear.in).length < gear.in.subtract(gear.out).length);
    const arcPath = arcDescription(gear.radius, largeArcFlag, !gear.sweep, gear.out);

    const gearExterior = createCircle(gear.x, gear.y, gear.radius - 1.5, '#888');
    const gearInterior = createCircle(gear.x, gear.y, gear.radius - 4.5, '#666');
    const gearCenter = createCircle(gear.x, gear.y, 3, '#888');

    const gearTeeth = createPath({
        d: `M${gear.in}${arcPath}${arcDescription(gear.radius, !largeArcFlag, !gear.sweep, gear.in)}`,
        stroke: '#888',
        'stroke-width': 5
    });

    const chainParts = cloneElement(gearTeeth, {
        d: `M${gear.in}${arcPath}L${nextGear.in}`,
        stroke: '#222'
    });

    gear.teeth = gearTeeth;
    gear.chainParts = chainParts;

    chainPath += `${arcPath}L${nextGear.in}`;

    svg.appendChild(gearExterior);
    svg.appendChild(gearInterior);
    svg.appendChild(gearCenter);
    svg.appendChild(gearTeeth);
    svg.appendChild(chainParts);
});
```

Draw chain

```
const chain = cloneElement(firstGear.chainParts, {
    d: 'M' + firstGear.in + chainPath,
    'stroke-width': 2
});

const chainLength = chain.getTotalLength();
const chainUnit = chainLength / Math.round(chainLength / (4 * Math.PI)) / 2;
const animationOffset = 8 * chainUnit;
```

Animate gears and chain

```
loopGears(gears, (gear, prevGear) => {

    if (gear === firstGear) {
        gear.teethOffset = chainUnit;
        gear.chainOffset = 0;
    } else {
        gear.teethOffset = prevGear.teethOffset + prevGear.chainParts.getTotalLength();
        gear.chainOffset = prevGear.chainOffset + prevGear.chainParts.getTotalLength();
    }

    setAttributes(gear.teeth, {
        'stroke-dasharray': chainUnit,
        'stroke-dashoffset': gear.teethOffset
    });

    setAttributes(gear.chainParts, {
        'stroke-dasharray': chainUnit,
        'stroke-dashoffset': gear.chainOffset
    });

    const animate = createElement('animate', {
        attributeName: 'stroke-dashoffset',
        from: gear.teethOffset + animationOffset,
        to: gear.teethOffset,
        repeatCount: 'indefinite',
        dur: '1s'
    });

    const cloneAnimate = cloneElement(animate, {
        from: gear.chainOffset + animationOffset,
        to: gear.chainOffset
    });

    gear.teeth.appendChild(animate);
    gear.chainParts.appendChild(cloneAnimate);
});
```

Minify it back

EVIL LAUGHT

1681 -> 5896 -> 1665

THE END

Thank you for listening