

Open in app ↗

Medium

 Search Write 11

★ Get unlimited access to the best of Medium for less than \$1/week. [Become a member](#)



Creating Smooth Curves with Chaikin's Algorithm

Juan Espinoza · [Follow](#)

7 min read · Apr 19, 2024



1



How computers render smooth curves efficiently.

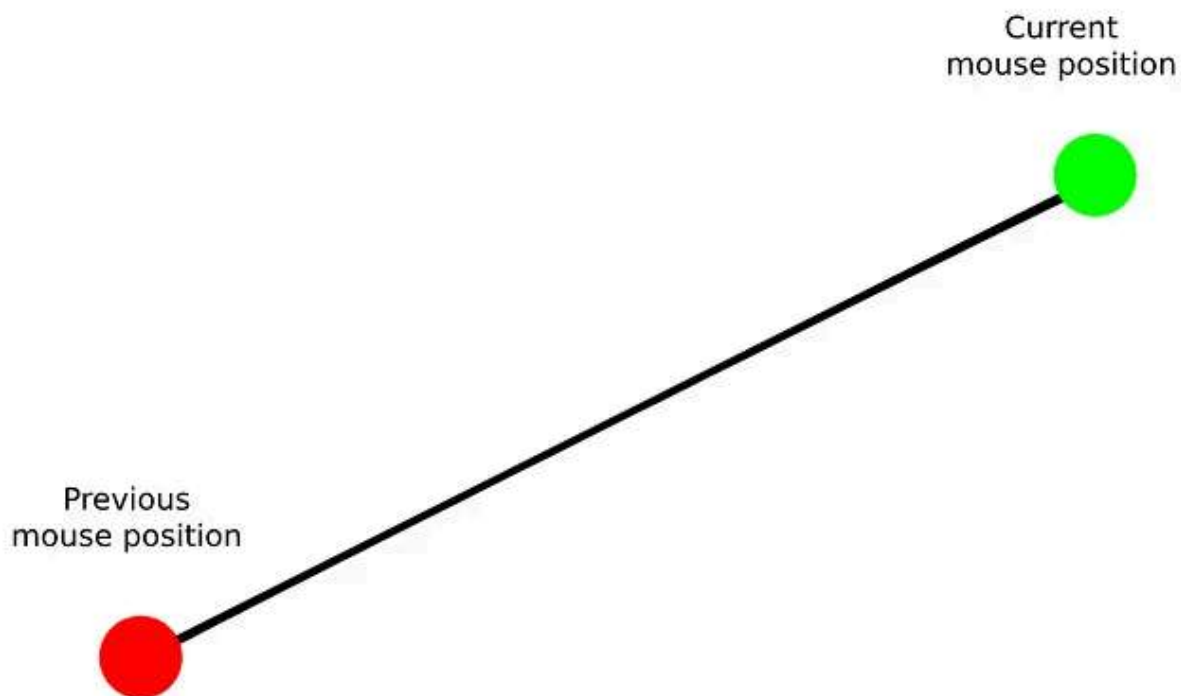
One of the first apps I ever programmed was a MS Paint clone. I was very proud of myself. With only a few lines of code I had created something that resembled a real app.

But one day, while using Adobe Photoshop, I noticed something peculiar. Whenever I drew quickly, the result was different to my clone app. With Photoshop I got smooth curves, but with my clone app, I got these jagged lines.

At that time I didn't know how to solve this problem. Little did I know, the solution existed since 1974.

My Strategy

The way my app worked was rather simple. Every frame, I would store the current mouse position. Then, I drew a line from the previous position to the new position. Simple!



The problem is that when drawing fast, my code couldn't capture mouse movements in between frames. This resulted in loss of points in a curve and

thus the jagged lines.

I tried using other frameworks and even looked into multi-threading in hopes of fixing mouse lags, but in the end, all efforts proved futile.

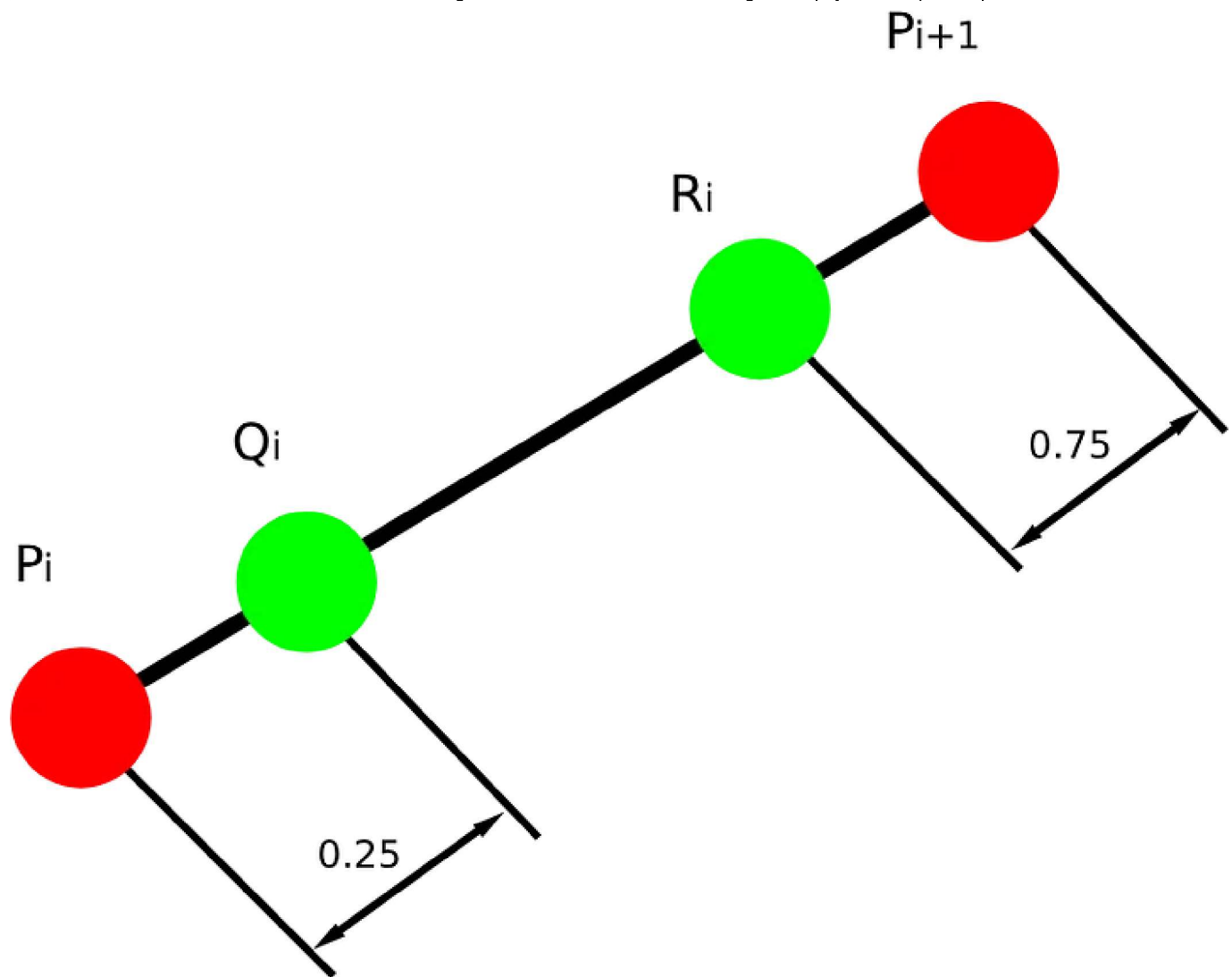
I realized that the problem wasn't in the way I captured the mouse, but in how I connected the points. I needed a way to create smooth curves out of a list of points.

Enter George Chaikin.

An Algorithm for High Speed Curve Generation

In 1974, George Chaikin proposed an algorithm for generating smooth curves with a small number of control points. He published this algorithm in his paper "An algorithm for high speed curve generation".

The algorithm is quite simple: In a list of points, for each pair of neighboring points, we create two new points that are a fraction a along the line segment between the two original points.



Given two neighboring points, P_i and P_{i+1} , We create two new points Q_i , which is $1/4$ of the way, and R_i , which is $3/4$ of the way.

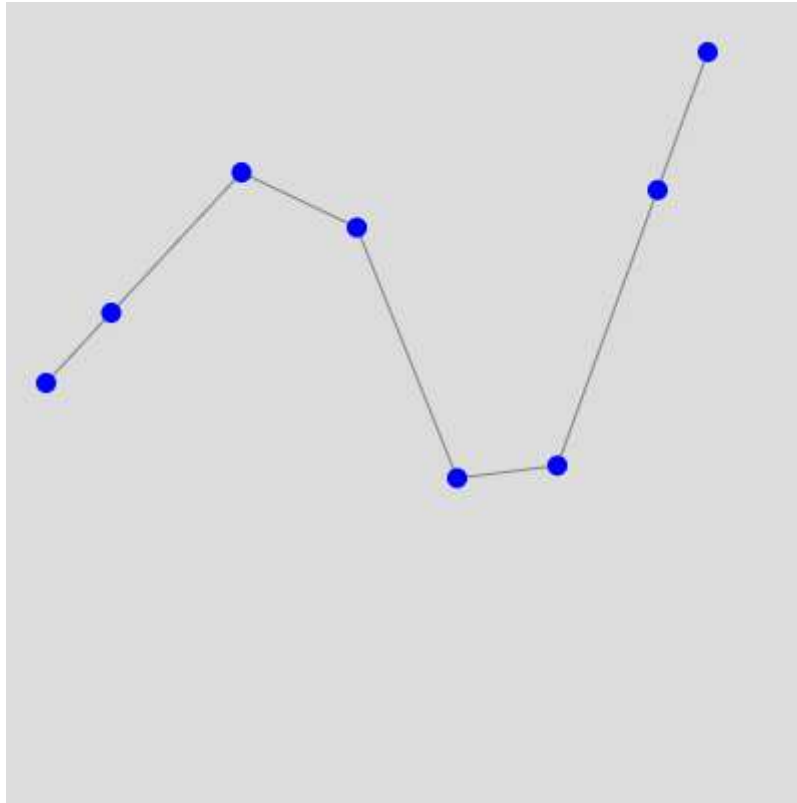
We can use the following formula to calculate the new points:

$$Q = [0.75 * P0x + 0.25 * P1x, 0.75 * P0y + 0.25 * P1y]$$

$$R = [0.25 * P0x + 0.75 * P1x, 0.25 * P0y + 0.75 * P1y]$$

And that's it! We just need to repeat this for every line segment in our list.

Also, If we do multiple iterations our results will be smoother. For example, say we take a list of points and apply the algorithm, we get a new list of points. If we take this new list of points and apply the algorithm again, our result will be an even smoother curve.



Of course, there is a limit to how many iterations you can perform without affecting your performance.

Now that we know the theory behind the algorithm, let's try to implement it in code.

The Starter Code

I'm going to use the p5.js JavaScript library. This is a creative coding library which makes coding easy for designers, artists, and beginners.

You can get the p5.js library by going to the official website (<https://p5js.org/>). You have an option to download the library files and run them locally or you can use the online web editor, which allows you to start coding right in your browser.

Let's take a look at the starter code.

```
let pg;
let strokePoints;

function setup() {
  createCanvas(400, 400);

  pg = createGraphics(width, height); // drawing surface

  strokePoints = [];
}
```

We begin creating two global variables. `pg` will hold an off screen canvas that will serve as our drawing surface. `strokePoints` is an array that will save our mouse position whenever we hold down the mouse.

```
function draw() {
  background(220);

  // draw the points on the screen
  for (let i=0; i<strokePoints.length-1; i++){
    let p0 = strokePoints[i];
    let p1 = strokePoints[i+1];
    line(p0[0], p0[1], p1[0], p1[1]);
  }

  // render the drawing surface
  image(pg, 0, 0);
}
```

```
}
```

In the draw function, we first draw our `pg` surface to the screen. We then proceed to draw the points stored in the `strokePoints` to the screen. Notice that we are not drawing anything in the `pg` surface. We will only transfer the points to the off-screen canvas when we release the mouse button.

```
function mouseDragged(){  
  // add points while mouse drag  
  strokePoints.push([mouseX, mouseY]);  
}
```

In the mouse drag event, we add our current mouse position to our `strokePoints` array.

```
function mouseReleased(){  
  // render points on drawing surface and  
  // clear point list  
  for (let i=0; i<strokePoints.length-1; i++){  
    let p0 = strokePoints[i];  
    let p1 = strokePoints[i+1];  
    pg.line(p0[0], p0[1], p1[0], p1[1]);  
  }  
  strokePoints = [];  
}
```

Finally, we will take the user mouse release as indication that we want to commit the stroke to the `pg` surface. We use a `for` loop to iterate through every pair of points and connect them with lines. Once our `strokePoints`

have been transferred to our `pg` surface, we can clear our `strokePoints` array.

Here you have the complete code.

```
let pg;
let strokePoints;

function setup() {
  createCanvas(400, 400);

  pg = createGraphics(width, height); // drawing surface

  strokePoints = [];
}

function draw() {
  background(220);

  // draw the points on the screen
  for (let i=0; i<strokePoints.length-1; i++){
    let p0 = strokePoints[i];
    let p1 = strokePoints[i+1];
    line(p0[0], p0[1], p1[0], p1[1]);
  }

  // render the drawing surface
  image(pg, 0, 0);
}

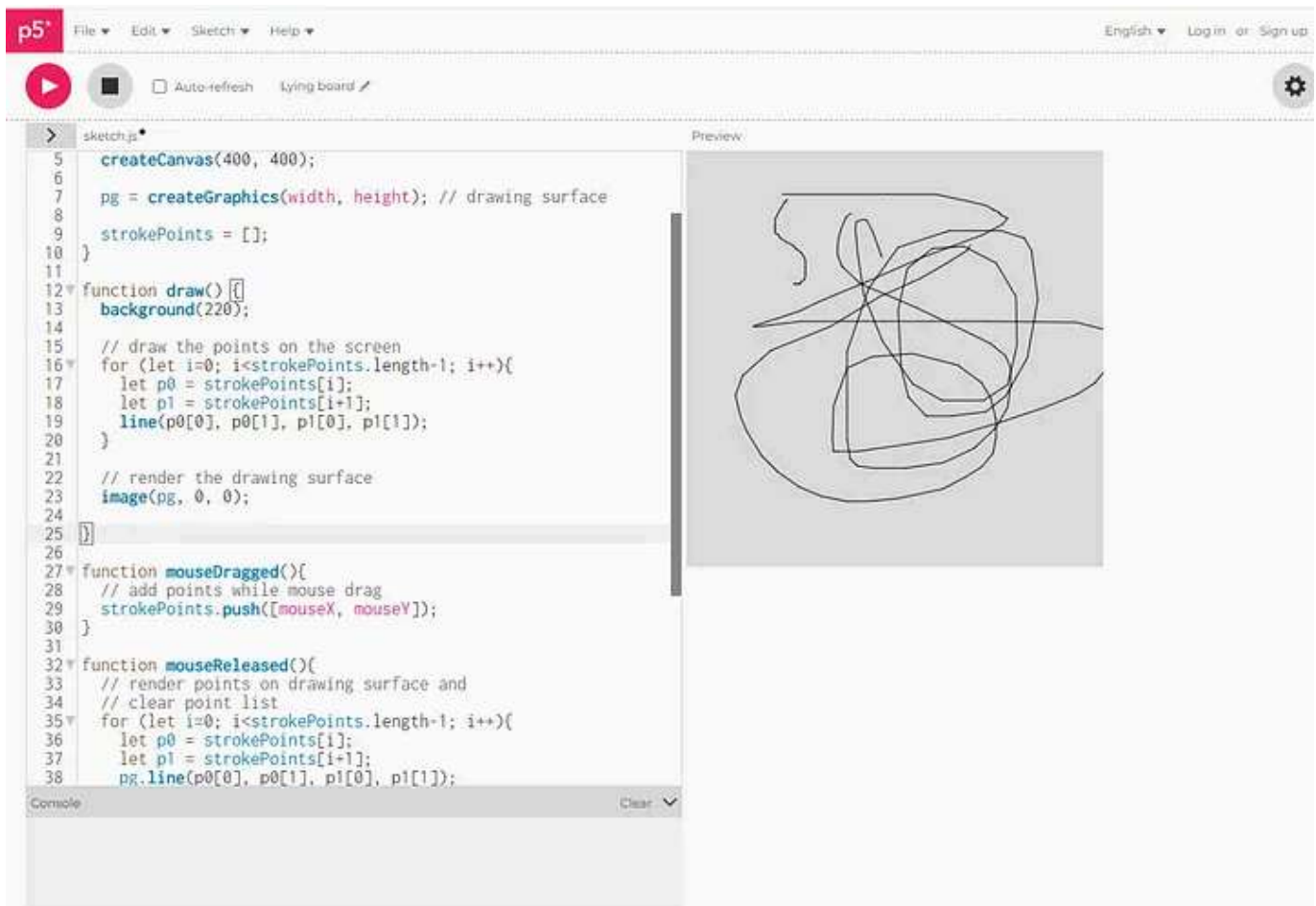
function mouseDragged(){
  // add points while mouse drag
  strokePoints.push([mouseX, mouseY]);
}

function mouseReleased(){
  // render points on drawing surface and
  // clear point list
  for (let i=0; i<strokePoints.length-1; i++){
    let p0 = strokePoints[i];
    let p1 = strokePoints[i+1];
    pg.line(p0[0], p0[1], p1[0], p1[1]);
  }
}
```



```
strokePoints = [];
}
```

Run the sketch by clicking the run button in top left of the editor.



Drag your mouse to add stroke to the canvas.

Now is time to add the Chaikin algorithm to smooth whatever strokes we place on the screen.

Implementing the Algorithm

Let's create a function that we can reuse in our code.

```
function chaikin(inPoints){
  let tmp = [];
  for (let i=0; i<inPoints.length-1; i++){
    let p0 = inPoints[i];
    let p1 = inPoints[i+1];
    let q = [0.75*p0[0]+0.25*p1[0], 0.75*p0[1]+0.25*p1[1]];
    let r = [0.25*p0[0]+0.75*p1[0], 0.25*p0[1]+0.75*p1[1]];
    tmp.push(q);
    tmp.push(r);
  }
  return tmp;
}
```

Start by creating a function and name it `chaikin`. This function takes in an array of points. For each pair of neighboring points, we calculate points Q and R. We then save the new points to a temporary array called `tmp`.

Finally, our function returns the new array of “smooth” points.

Let’s modify our previous code by adding a call to the `chaikin` function before we draw the stroke to the screen.

```
function draw() {
  background(220);

  // smooth points
  let smoothPoints = chaikin(strokePoints);

  // draw the points on the screen
  for (let i=0; i<smoothPoints.length-1; i++){
    let p0 = smoothPoints[i];
    let p1 = smoothPoints[i+1];
    line(p0[0], p0[1], p1[0], p1[1]);
  }

  // render the drawing surface
  image(pg, 0, 0);
}
```

```
}
```

We pass the `strokePoints` array to the `chaikin` function and create a new variable called `smoothPoints` which will hold the result of the `chaikin` function.

Now, instead of drawing the `strokePoints` array, we draw the `smoothPoints`.

If we run the code and test it by dragging our mouse, you should see that our curves are now smooth. However, when you release the mouse, we see the jagged lines being drawn to the off-screen surface.

We need to modify the code, so that it also applies the `chaikin` function before permanently painting the points on the `pg` surface.

```
function mouseReleased(){  
  
    // apply chaikin's algorithm to points  
    let smoothPoints = chaikin(strokePoints);  
  
    // render points on drawing surface and  
    // clear point list  
    for (let i=0; i<smoothPoints.length-1; i++){  
        let p0 = smoothPoints[i];  
        let p1 = smoothPoints[i+1];  
        pg.line(p0[0], p0[1], p1[0], p1[1]);  
    }  
    strokePoints = [];  
}
```

The code is similar to what we wrote in the draw function. We pass the `strokePoints` into the `chaikin` function and save the results into an array called `smoothPoints`. We then draw the `smoothPoints`, but this time, we preface the line drawing function with `pg`.

Here you have the complete code for the final result.

```
let pg;
let strokePoints;

function setup() {
  createCanvas(400, 400);

  pg = createGraphics(width, height); // drawing surface

  strokePoints = [];
}

function draw() {
  background(220);

  // smooth points
  let smoothPoints = chaikin(strokePoints);

  // draw the points on the screen
  for (let i=0; i<smoothPoints.length-1; i++){
    let p0 = smoothPoints[i];
    let p1 = smoothPoints[i+1];
    line(p0[0], p0[1], p1[0], p1[1]);
  }

  // render the drawing surface
  image(pg, 0, 0);
}

function mouseDragged(){
  // add points while mouse drag
  strokePoints.push([mouseX, mouseY]);
}

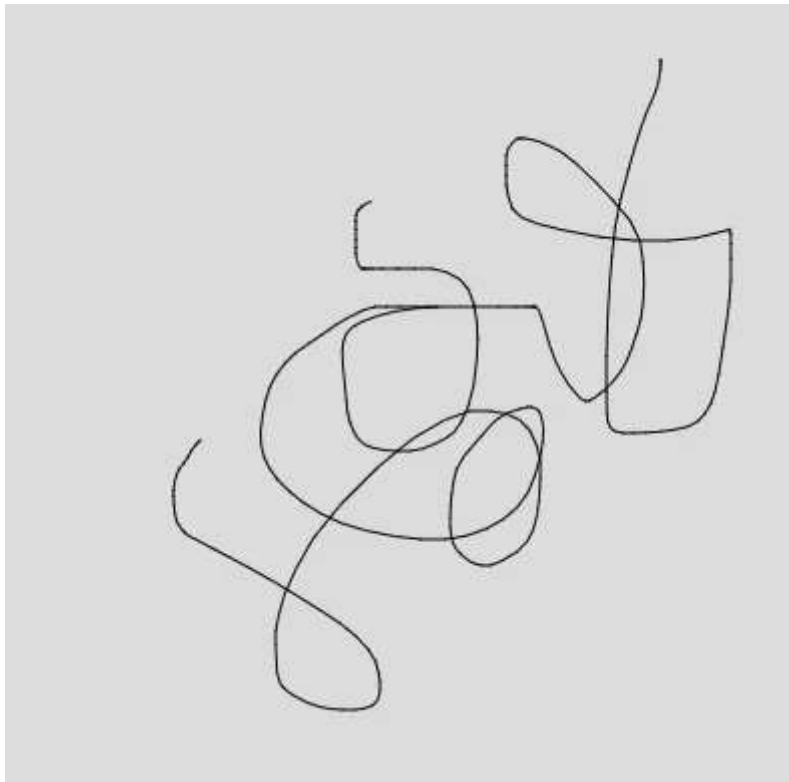
function mouseReleased(){
```

```
// apply chaikin's algorithm to points
let smoothPoints = chaikin(strokePoints);

// render points on drawing surface and
// clear point list
for (let i=0; i<smoothPoints.length-1; i++){
  let p0 = smoothPoints[i];
  let p1 = smoothPoints[i+1];
  pg.line(p0[0], p0[1], p1[0], p1[1]);
}
strokePoints = [];
}

function chaikin(inPoints){
  let tmp = [];
  for (let i=0; i<inPoints.length-1; i++){
    let p0 = inPoints[i];
    let p1 = inPoints[i+1];
    let q = [0.75*p0[0]+0.25*p1[0], 0.75*p0[1]+0.25*p1[1]];
    let r = [0.25*p0[0]+0.75*p1[0], 0.25*p0[1]+0.75*p1[1]];
    tmp.push(q);
    tmp.push(r);
  }
  return tmp;
}
```

Run the sketch and test it by dragging the mouse across the screen.



Our strokes are now smooth whenever we draw curves quickly.

Final Words

Congratulations! You now have implemented one of the most clever algorithms in curve approximation. If you wish to take it further, you can add more iterations to the points array before drawing them.

I'll leave it up to you to implement this changes on your own.

I hope you liked this small tutorial and hopefully you now have an extra tool in your tool belt.

[JavaScript](#)[Computer Graphics](#)[Design](#)[Code](#)



Written by Juan Espinoza

12 Followers · 4 Following

Follow

Passionate about blending engineering with artistic innovation 🧑🏻💻 Teaching Coding | 🧠 Creative Coding | 💻 Computer Graphics | 🌐 Augmented Reality

No responses yet



What are your thoughts?

Respond

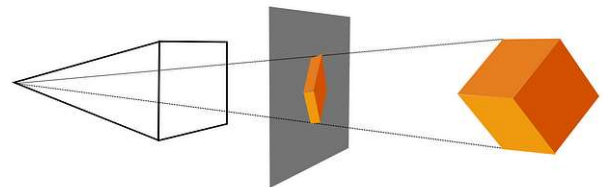
More from Juan Espinoza



Juan Espinoza

3D Graphics: The Perspective Projection

Did you know that at one point in time, painters struggled to realistically portray...



Juan Espinoza

3D Graphics: A Beginners Guide

Nowadays Computer graphics, or CG, are everywhere. From video games to medical...

Feb 28, 2024



3



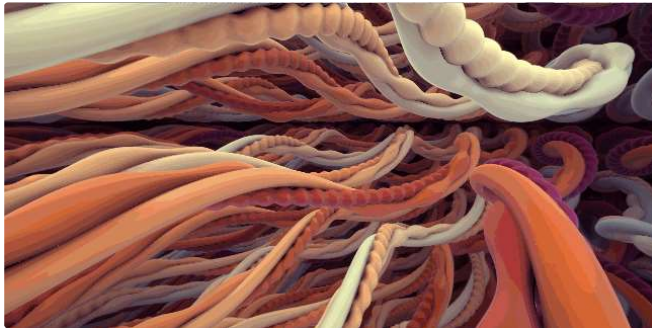
4



Feb 21, 2024



5



Juan Espinoza

3D Graphics: Intro to GLSL

A short introduction to shader programming

Sep 30, 2024



7



1



Juan Espinoza

3D Graphics: Implementing a 3D Renderer

In this article, we will put all of the previously covered theory into practical use by renderi...

Mar 7, 2024



1

[See all from Juan Espinoza](#)

Recommended from Medium

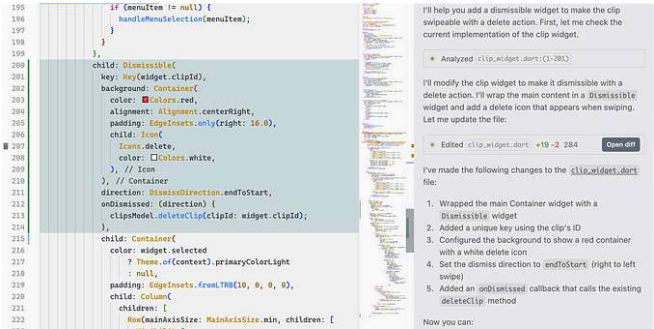



 Martijn Assie

Building Interactive 3D Websites with Kadence, Svelte, and WebGL...

Ever wanted to create an immersive, interactive 3D website? Combining Kadence...

Sep 12, 2024  52  2  



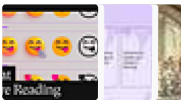
 In Coding Beauty by Tari Ibaba

This new IDE just destroyed VS Code and Copilot without even...

Wow I never thought the day I stop using VS Code would come so soon...

Jan 18  2K  86  

Lists



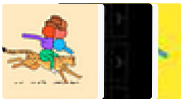
Stories to Help You Grow as a Designer

11 stories · 1088 saves



Interesting Design Topics

258 stories · 948 saves



Figma 101


7 stories · 817 saves

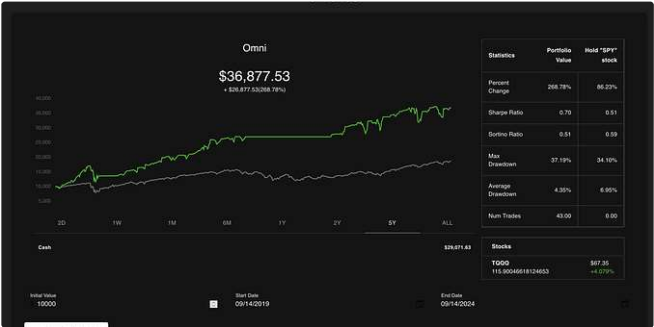



Icon Design

39 stories · 482 saves



 Harendra




 In DataDrivenInvestor by Austin Starks

How I Am Using a Lifetime 100% Free Server

Get a server with 24 GB RAM + 4 CPU + 200 GB Storage + Always Free

★ Oct 26, 2024 🖱 8.7K 💬 133 📌+ ...



 Alberto Romero

DeepSeek Is Chinese But Its AI Models Are From Another Planet

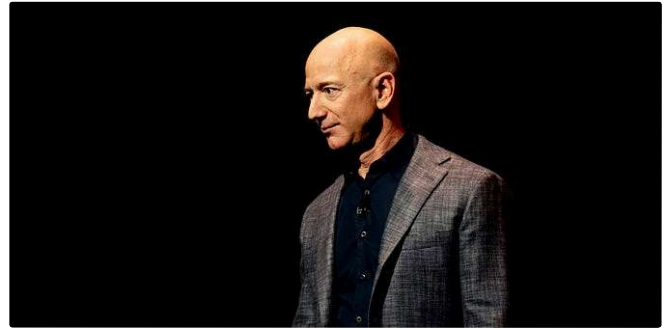
OpenAI and the US are in deep trouble


★ Jan 23 🖱 4.6K 💬 128 📌+ ...

I used OpenAI's o1 model to develop a trading strategy. It is...

It literally took one try. I was shocked.

★ Sep 16, 2024 🖱 8.6K 💬 217 📌+ ...



 Jessica Stillman

Jeff Bezos Says the 1-Hour Rule Makes Him Smarter. New...

Jeff Bezos's morning routine has long included the one-hour rule. New...

★ Oct 30, 2024 🖱 21K 💬 572 📌+ ...

See more recommendations