PODER EXECUTIVO
MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE RORAIMA
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

COMPUTAÇÃO GRÁFICA

RELATÓRIO DE ALGORITMOS CURVAS DE BÉZIER E DE RECORTES DE POLÍGONO

ALUNOS:
Pedro Aleph Gomes de Souza Vasconcelos – 2016.007150

Novembro de 2019
Boa Vista/Roraima

Introdução

Este tem como objetivo relatar a construção feita e os resultados obtidos dos algoritmos curvas de Bézier(incluindo Casteljau) e de recortes de polígono(Sutherland Hodgman) conforme foi proposta para ser feito, a partir da ideia de cada um apresentado em sala, e pesquisando mais um pouco.

Desenvolvimento

Para desenvolver foi utilizado a biblioteca p5.js do JavaScript [1], ela é focada em uso do canvas, contendo várias funcionalidades para desenhar. No caso foi utilizado o próprio canvas. point(x,y), para o algoritmo de bézier; line(x1,y1, x2, x2), para casteljau; e para o recorte de polígono, foi usado shape(), que cria uma figura a partir dos vértices dados ordenadamente (vertex(x,y)).

Curva de Bézier

Começamos determinando quatro pontos de entrada(P0, P1, P2 e P3) com seus respectivos valores de x e y.

Para melhor visualização da curva a ser gerada, eu tracei as linhas através dos quatro pontos, e coloquei um texto para cada ponto.

E para formar a curva temos a implementação da equação da curva de bézier de quatro pontos, com x e y calculados separadamente. a equação é definida em função de t, que varia de 0 a 1, nesse caso, a precisão é 0,001, menos do que isso o pontos não de conectam.

equação:

$$C(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3$$

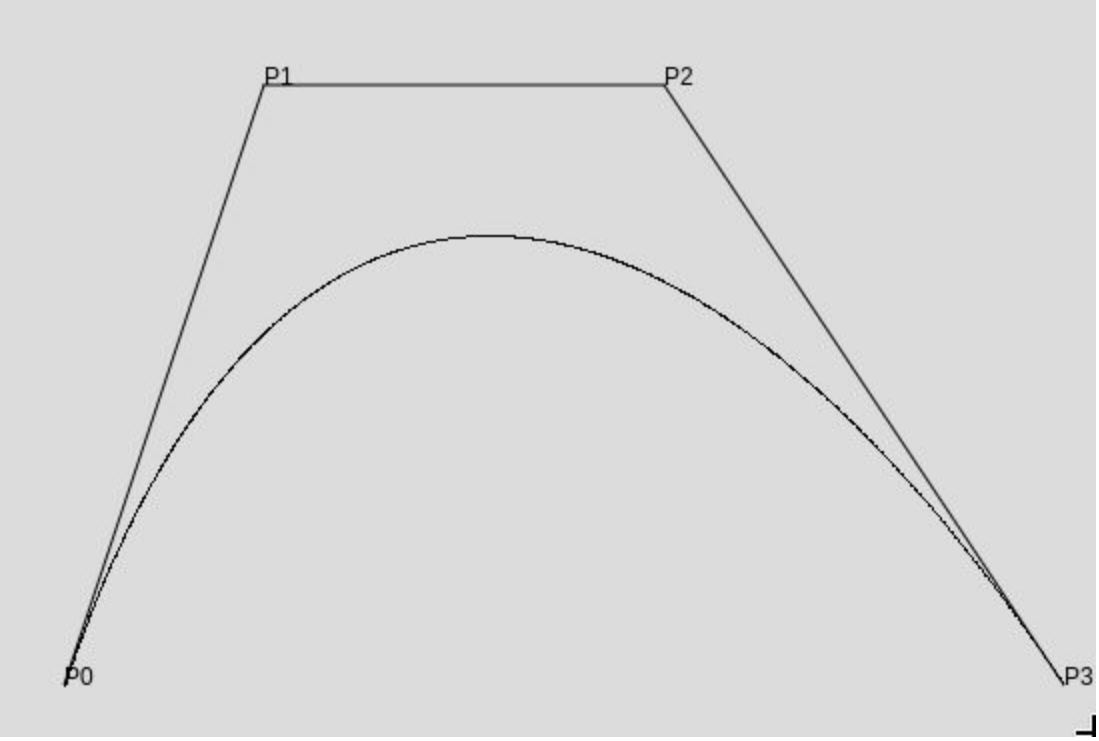
no código:

```
for(var t = 0; t <= 1; t+= 0.001){
  x = pow(1 - t, 3) * x1 + 3 * t * pow(1 - t, 2) * x2
    + 3 * pow(t, 2) * (1 - t) * x3 + pow(t, 3) * x4;
  y = pow(1 - t, 3) * y1 + 3 * t * pow(1 - t, 2) * y2
    + 3 * pow(t, 2) * (1 - t) * y3 + pow(t, 3) * y4;
  point(x, y);
}
```

para os pontos:

```
x1 = 100; y1 = 400;
x2 = 200; y2 = 100;
x3 = 400; y3 = 100;
x4 = 600; y4 = 400;
```

temos:



Algoritmo de Casteljau

Diferente de bézier que utiliza t como parâmetro, este se baseia em recursão de pontos médios obtidos entre os pontos e mais formados entre os próximos até que obtém se o ponto médio entre duas retas : MP01(entre P0 e P1), MP12(entre P1 e P2), MP23; MP012 (entre MP01 e MP12), MP123(entre MP12 e MP23); e finalmente MP0123(entre MP012 e MP123).

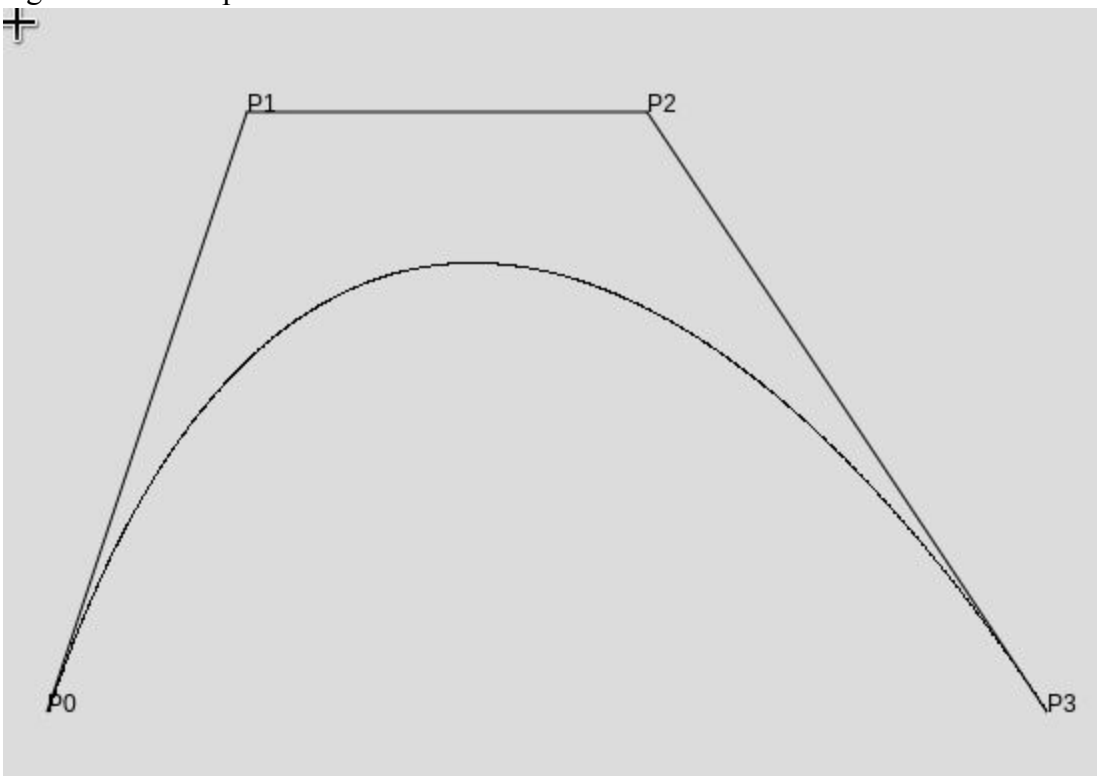
Aposś calcular a primeira vez, tem se que divide para cada metade da curva, e assim o ponto médio x e y é marcado para cada nível de recursão

segue o código:

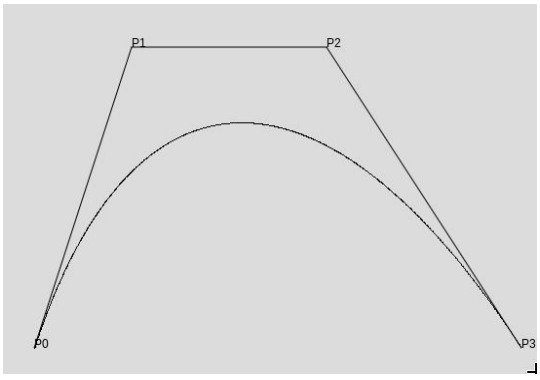
```
function draw_casteljau(x1,  y1, x2, y2, x3, y3, x4, y4, t){
  if (t <= MAX){
    var mx12 = media(x1, x2);
    var my12 = media(y1, y2);
    var mx23 = media(x2, x3);
    var my23 = media(y2, y3);
    var mx34 = media(x3, x4);
    var my34 = media(y3, y4);
    var mx123 = media(mx12, mx23);
    var my123 = media(my12, my23);
    var mx234 = media(mx23, mx34);
    var my234 =  media(my23, my34);
    var x = media(mx123, mx234);
    var y = media(my123, my234);
    draw_casteljau(x1, y1, mx12, my12, mx123, my123, x, y, t + 1);
    draw_casteljau(x, y, mx234, my234, mx34, my34, x4, y4, t + 1);
    point(x, y);
  }
}
```

neste caso foi definido que o nível da recursão deve ser 10(9 já preenche tudo);

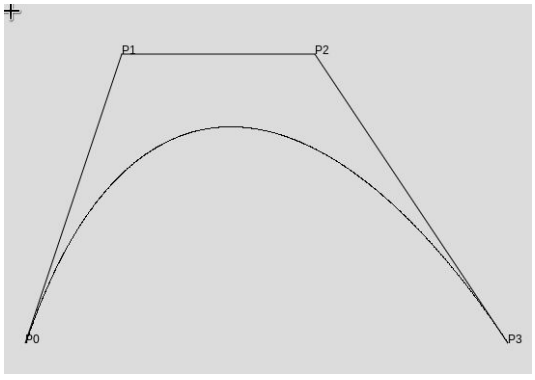
seguindo com os pontos anteriores:



comparando as curvas:



bézier

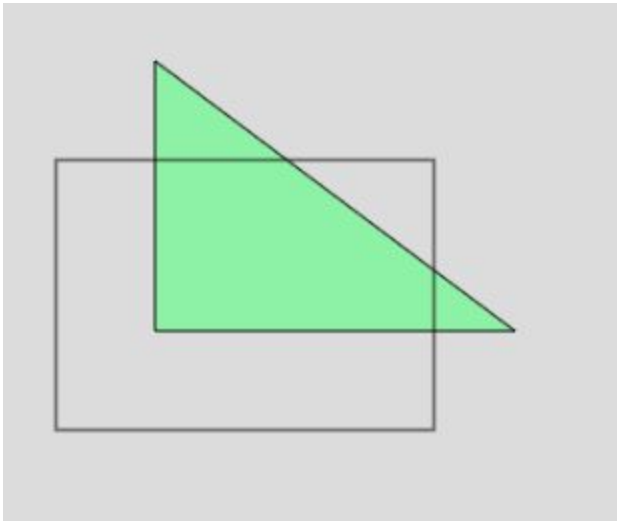


Casteljau

Recorte de Polígonos - Sutherland Hodgman

Este algoritmo tem como objetivo de dado um polígono, traçamos outro polígono sobre este, e cortamos os seus lados que ultrapassem os polígono de recorte, para isso temos que criar um polígono qualquer e um polígono de recorte.

neste caso, temos um triângulo e um quadrilátero como polígono de recorte, criados com referência dos vértices:



este foi criado da seguinte forma:

triângulo

```
let v;
fill('rgba(0,255,0, 0.25)');
beginShape();
v = [100, 50]; vertex(v[0], v[1]); O.push(v);
v = [100, 200]; vertex(v[0], v[1]); O.push(v);
v = [300, 200]; vertex(v[0], v[1]); O.push(v);
endShape(CLOSE);
```

quadrilátero

```
let v;
noFill();
beginShape();
v = [45, 105]; vertex(v[0], v[1]); R.push(v);
v = [45, 255]; vertex(v[0], v[1]); R.push(v);
v = [255, 255]; vertex(v[0], v[1]); R.push(v);
v = [255, 105]; vertex(v[0], v[1]); R.push(v);
endShape(CLOSE);
```

(Observe que os vértices estão em vetores, isso é para facilitar sua utilização no algoritmo)

O algoritmo funciona da seguinte forma [3]: há duas listas, uma de entrada e uma de saída, a lista de saída inicialmente tem todos os vértices do polígono a ser recortado ; para cada aresta do polígono de recorte, a lista de entrada salva os valores da lista de saída no início(e depois é usada para determinar os pontos S e P) e a lista de saída fica vazia, ele então calcula o ponto de interseção(I) para cada arestas do polígono a ser cortado(com as retas estendidas ao infinito), usando a aresta do polígono de recorte atual com a aresta formada pelos pontos S e P(que percorrem os vértices do polígono a ser recortado) ; depois, calcula se com I, se os pontos S e P estão dentro do polígono de recorte, caso os dois estejam dentro, a lista de saída recebe o ponto S, caso um esteja fora do vértice (no caso é P), a lista de saída recebe antes S, o ponto I; caso só o ponto P estiver dentro então só o ponto I é adicionado a lista; e depois do código ter percorrido todas arestas do polígono de

recorte, temos a lista de saída com todos os vértices do polígono recortado, e assim este é formado.
Código:

```
for(var e = 0; e < edge; e++){
  input = output;
  output = [];
  E = findEdge(R, e, edge);
  for(var i = 0; i < input.length; i++){
    S = input[i];
    P = input[(i + input.length - 1) % input.length];
    I = intersection(P, S, E);
    if(inside(S, E) < 0 ){
      if(inside(P, E) >= 0)
        output.push(I);
      output.push(S);
    }
    else if (inside(P, E) < 0)
      output.push(I);
  }
}
```

Funções auxiliares, para encontrar a interseção(com base na fórmula matemática) e se está dentro do polígono(é determinado com 0 como parâmetro, caso seja maior ou igual a zero não entra, caso contrário entra) :

Interseção:

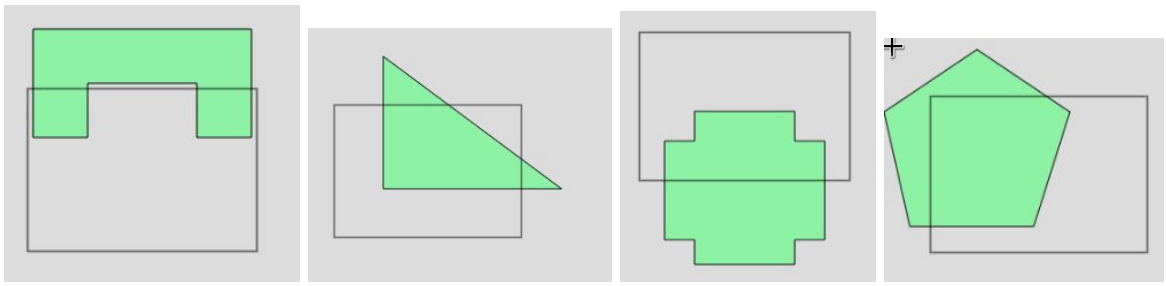
```
function intersection(P, S, E){
  let p = [];
  var x1 = P[0], x2 = S[0], x3 = E[0][0], x4 = E[1][0];
  var y1 = P[1], y2 = S[1], y3 = E[0][1], y4 = E[1][1];
  p.push(((x1 * y2 - y1 * x2) * (x3 - x4)
        - (x1 - x2) * (x3 * y4 - y3 * x4))
        / ((x1 - x2) * (y3 - y4) - (y1 - y2) * (x3 - x4)));
  p.push(((x1 * y2 - y1 * x2) * (y3 - y4)
        - (y1 - y2) * (x3 * y4 - y3 * x4))
        / ((x1 - x2) * (y3 - y4) - (y1 - y2) * (x3 - x4)));
  return p;
}
```

Dentro ou fora:

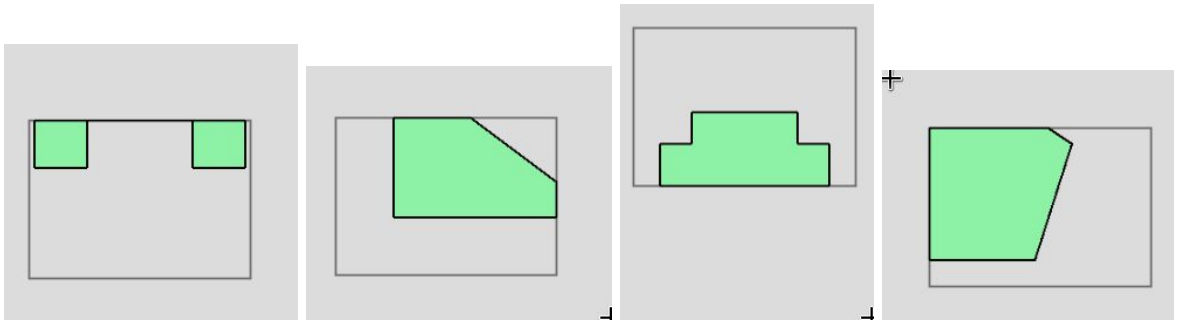
```
function inside( S, E){
  var x = S[0], x1 = E[0][0], x2 = E[1][0];
  var y = S[1], y1 = E[0][1], y2 = E[1][1];
  var p = (x2 - x1) * (y - y1) - (y2 - y1) * (x - x1);
  return p;
}
```

Polígonos usados:

Antes:



Depois:



Considerações finais

Este trabalho apresentou a implementação dos algoritmos de curva e recorte de polígonos, não foi citado uma nova forma de implementação desses algoritmos, somente foi utilizado ideias do mesmo já existentes e estudo do seu uso. o de bézier é o mais simples, apenas uma função matemática; o de casteljau , é particularmente o que mais gostei; e o de sutherland foi complicado de entender, mas todos foram uma boa experiência.

Referências

[1]https://p5js.org/
[2]https://editor.p5js.org/
[3]https://en.wikipedia.org/wiki/Sutherland%E2%80%93Hodgman_algorithm
[4]https://zh.wikipedia.org/wiki/%E5%BE%B7%E5%8D%A1%E6%96%AF%E7%89%B9%E9%87%8C%E5%A5%A5%E7%AE%97%E6%B3%95