MAC0331 - Geometria Computacional

Felipe Blassioli

January 15, 2015

1 Introdução

O código-fonte do projeto está disponível no github e consiste na modificação do projeto do alexis.

2 Instruções de Uso

2.0.1 Execução

```
$ python tkgeocomp.py
```

2.1 Projeto do Alexis

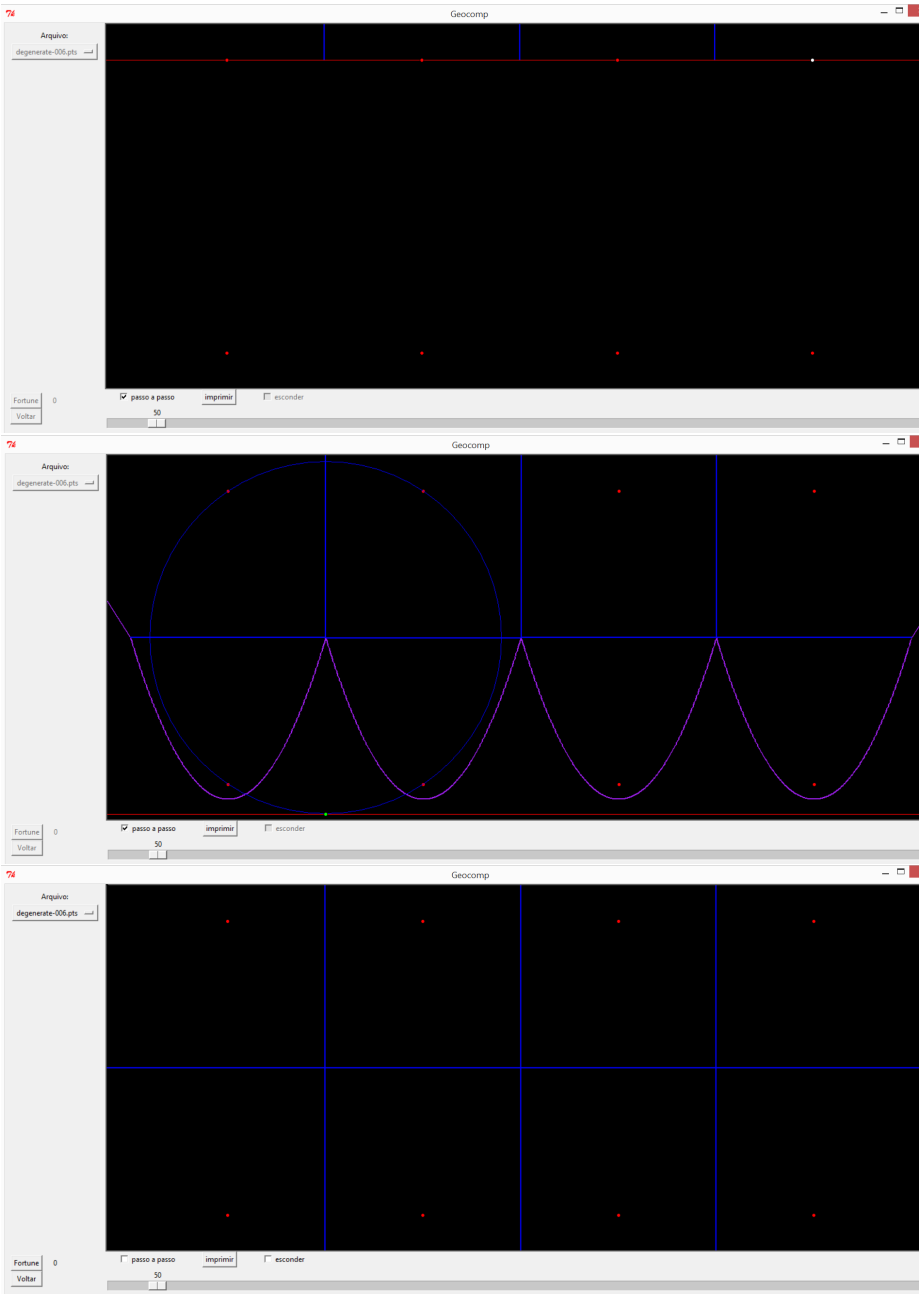
Mantive os controles padrões do projeto do alexis. Portanto, para executar passo-a-passo o algoritmo:

1. Selecionar o problema: Voronoi.
2. Selecionar o algoritmo: Fortune.
3. Selecionar a entrada: degenerate-006.pts
4. Clicar no botão fortune.
5. Pressionar a barra de espaço para ir para a próxima iteração.

2.1.1 Cores

- Roxo: Linha de praia.
- Amarelo: Circunferência do evento círculo.
- Azul: arestas do diagrama de voronoi.
- Verde: evento círculo.
- Vermelho: evento ponto.
- Branco: evento ponto selecionado.
- Vermelho: linha de varredura.

2.1.2 Exemplo de Uso

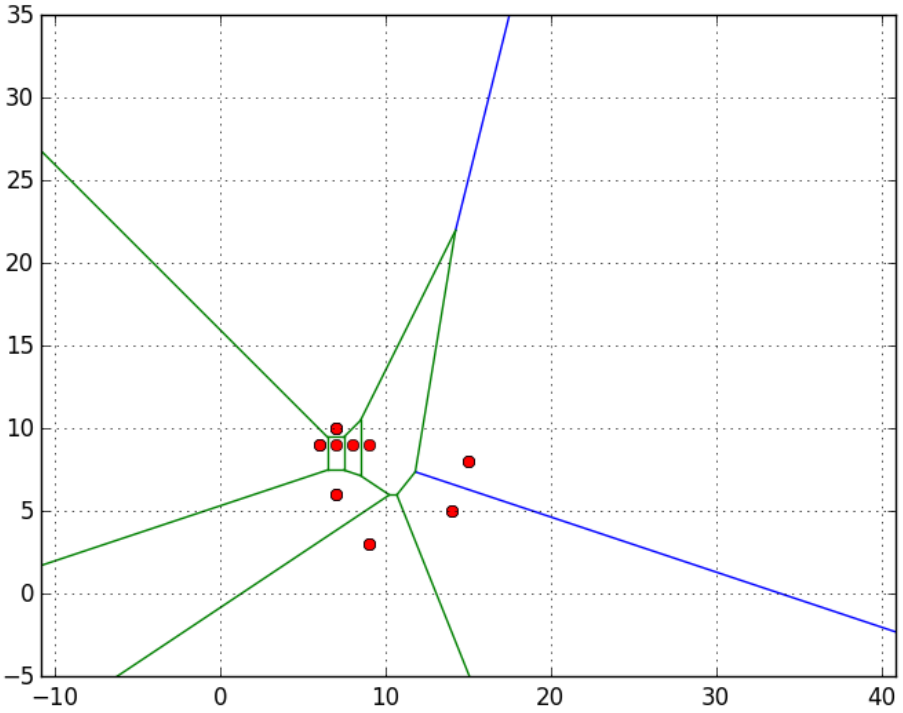


2.2 Gerando PNGs usando matplotlib

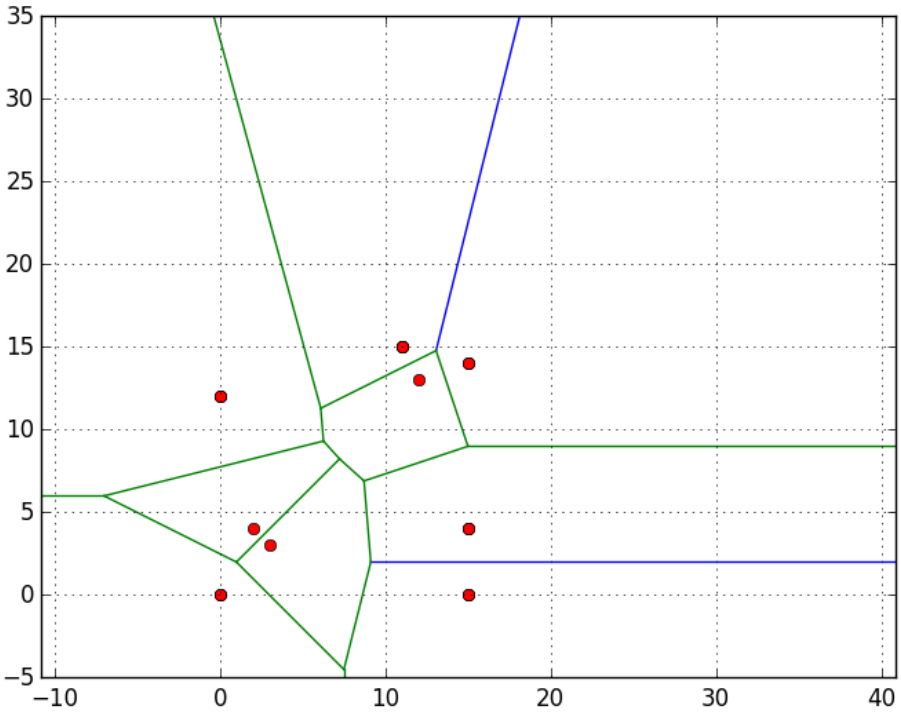
Caso o matplotlib esteja disponível é possível gerar os PNGs de todos os arquivos de entrada na pasta *dados/*.

```
$ python teste_all_data.py
```

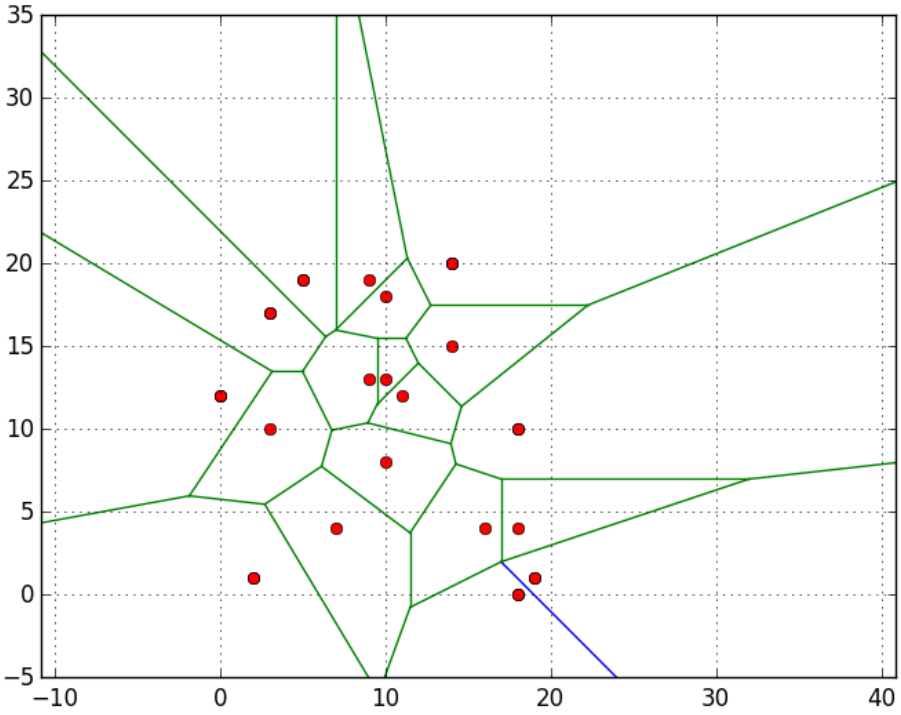
2.2.1 Exemplos de saída



degenerate-013.png



degenerate-015.png

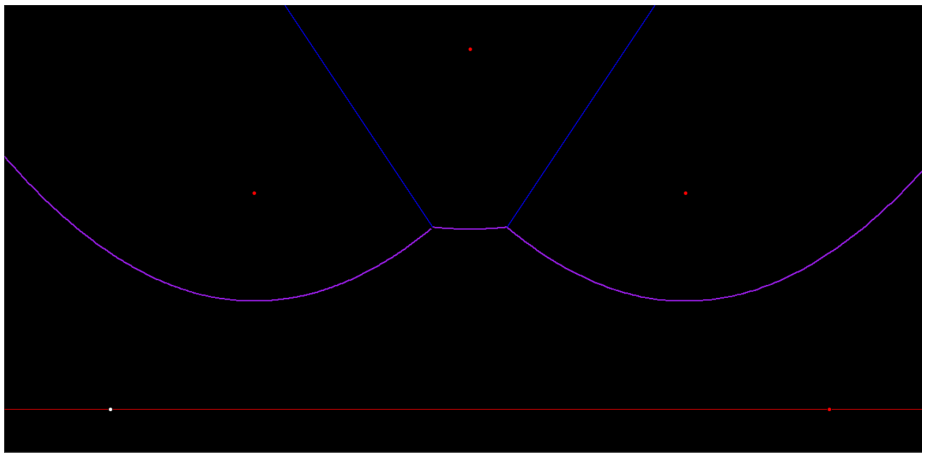


degenerate-022.png

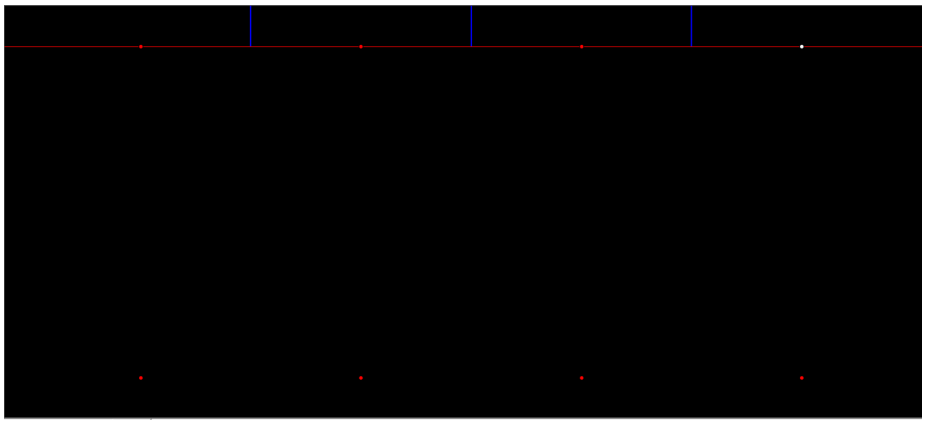
3 Casos degenerados

Acredito ter tratado todos os casos degenerados. Foi especialmente difícil tratar o caso em que o primeiro e o segundo ponto são colineares, já que, nesse caso não há intersecção. Encontrei MUITOS erros de implementação do algoritmo ao lidar com os pontos cocirculares.

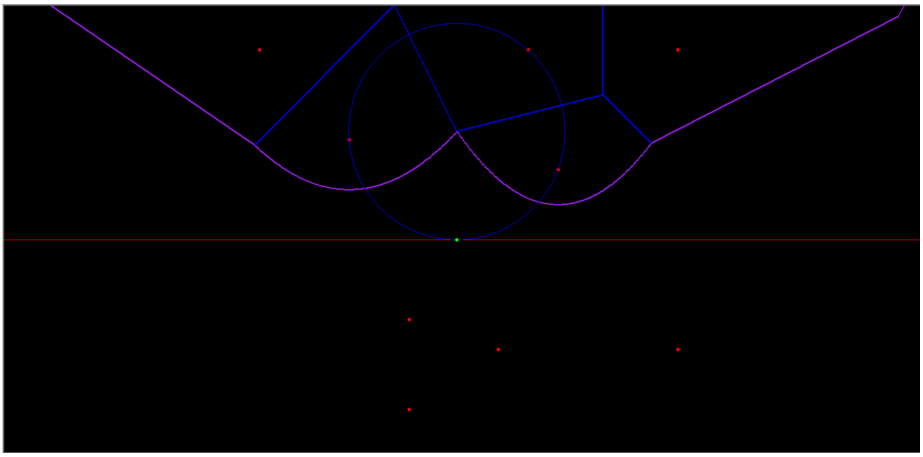
1. **Pontos colineares**



2. **2 ou mais pontos colinares (incluindo o ponto mais alto)**



3. **4 pontos cocirculares**



4 Implementação

1. **voronoi.py:** arquivo principal. Contém o algoritmo como descrito no livro do berg.
2. **beachline.py:** implementa a linha de praia utilizando uma árvore binária.
3. **event.py:** implementa a fila de eventos e os eventos (evento-ponto e evento-círculo). Utiliza uma fila de prioridades.
4. **dcel.py:** implementação da doubly connected edge list
5. **geometry.py:** contém cálculos geométricos, como encontrar a circunferência que passa por 3 pontos.
6. **utils.py:** Implementação de árvore binária genérica (mas uma implementação incompleta, só com o necessário) que serve de base à linha de praia.
7. **anim.py:** Responsável por animar voronoi.py e fazer a integração com o projeto do alexis.

8. **anim2.py:** Gera PNGs para cada iteração do algoritmo utilizando a biblioteca matplotlib.

5 Estrutura de Dados

5.1 Linha de Praia

Deveria ser um ABBB, mas é apenas uma árvore binária, não consegui ainda balancear (AVL) sem quebrar o algoritmo. Está implementada no arquivo beachline.py.

5.2 Fila de Eventos

Trata-se de uma fila de prioridades onde a prioridade é a coordenada Y dos pontos. Há dois tipos de eventos: SiteEvent e CircleEvent. O primeiro tipo consiste na coleção de pontos cujo diagrama de voronoi queremos encontrar, o segundo tipo consiste nos vértices do diagrama de voronoi, os quais vamos encontrando à medida que a linha de varredura progride.

5.3 Doubly connected edge list

O diagrama de voronoi é armazenada numa DCEL que está implementada emdcel.py.

Sua principal estrutura são os Half-Edges, os quais possuem vértices gêmeos e foram implementados segundo o livro do Berg. As outras partes do DCEL são a lista de faces, que está em VoronoiDiagram._faces e o conjunto de vértices que está em VoronoiDiagram._vertices.

6 Animação

A cada iteração do loop principal do algoritmo que encontra o diagrama de voronoi é chamada a função *animate* passando o ponto evento atual e dados de estado como a linha de praia. Assim, para integrar o algoritmo no projeto do Alexis bastou escrever um animate adequado, que fizesse uso de *geocomp.common.control*, interface que disponibiliza funções de desenho do canvas.

Isso foi feito em *geocomp.voronoi.anim*:

```
def _draw_step(self, e, beachline, event_queue, hedges, **kwargs):
        # Draw directrix
        self.plot_horizontal_line(e.y, color='red')

        self.plot_points(self.input)
        self._draw_beachline(e,beachline)
        self._draw_hedges(e, hedges)
        self._draw_circle_events(e, event_queue)

        if e.is_site:
                self.plot_points([e.site], color='white')
```

As funções de desenho são bastante diretas:

1. **_draw_beachline:** Desenha cada parabola da linha de praia utilizando plot_parabola
2. **_draw_hedges:** Desenha cada linha descoberta do diagrama de voronoi utilizando plot_line
3. **_draw_circle_events:** Caso o ponto evento atual seja um evento círculo, ele é desenhado como um ponto verde e sua circunferência é desenhada utilizando plot_circle.

7 Modificações no projeto do Alexis

7.1 geocomp.common.control

Foram acrescentadas as seguintes funções:

1. **clear_canvas**

   Limpa o canvas, isto é remove todos os artefatos gráficos.

   Optei por fazer a animação frame a frame, portanto a cada iteração era necessário limpar o canvas.

2. **plot_circle**

   Desenha uma circunferencia dado seu centro e seu raio.

   Já existia uma função *plot_disc*, no entando eu queria uma circunferência e plot_disc desenhava um círculo com preenchimento.

   As circunferências são necessárias para desenhar os eventos círculos.

3. **plot_parabola**

   Desenha uma parabola dado seu foco, diretrix e extremos.

7.2 geocomp.gui.tk

A principal modificação no arquivo foi a implementação da função *plot_parabola* que, dada o foco, a diretrix e dois pontos extremos desenha uma parábola que consiste em 240 linhas.

Segue um trecho da implementação de *plot_parabola*:

```
plot_parabola(focus, directrix, endpoints=None, color='purple')
...
f = lambda x: _parabola(x,focus,directrix)
prev = start
for x in linspace(start,end,240):
        p = (prev,f(prev))
        q = (x,f(x))
        prev = x
        line = plot_line(p[0],p[1],q[0],q[1],color=color,linewidth=2)
        lines.append(line)
```

A função *linspace(start,end,N)* retorna N pontos igualmente espaçados na reta que começa em start e termina em end. Segue sua implementação:

```
def linspace(start,end, total):
        dx = float((end - start)) / total
        x = start
        while x <= end:
                yield x
                x += dx
```