Instituto de Física Gleb Wataghin

F 590A - Iniciação Científica I - Relatório Pré-Final

Reconhecimento de Padrões em Imagens Digitais Usando Geometria Fractal

Aluno: Luan Nico (156303)

Orientador: João Florindo (jbflorindo)

6 de Junho de 2017

Il semble que la perfection soit atteinte
non quand il n'y a plus rien à ajouter,
mais quand il n'y a plus rien à retrancher.

Antoine de Saint Exupéry

0.1 Resumo

Este projeto propõe o estudo e desenvolvimento de ferramentas computacionais para análise de imagens baseadas em técnicas da geometria fractal. A modelagem por fractais parte do mapeamento de imagens de objetos da natureza para um conjunto de pontos em $\mathbb{R}^3$ que pode apresentar características (multi)fractais em um intervalo de escalas. Em seguida, são extraídas medidas fractais destes pontos. Entre essas medidas, as mais conhecidas são a dimensão fractal e o espectro multifractal. No domínio discreto das imagens, a aplicação de uma expressão analítica para este cálculo (dimensão de Hausdorff-Besicovitch) não é viável e parte-se então para métodos numéricos. O estudo destes métodos e sua relação com resultados teóricos da Geometria Fractal e Teoria da Medida é o foco deste projeto. O modelo será também aplicado a problemas de reconhecimento de padrões em imagens médicas e de plantas da flora brasileira.

0.2 Relevância

Esse trabalho tem bastante relevância no contexto da Física.

Primeiramente, os fractais podem ser aplicados em diversos ramos da física envolvendo a Teoria do Caos. Conforme visto Métodos Matemáticos da Física, muitos fenômenos físicos são puramente determinísticos, mas a solução das equações diferenciais que os regem é fortemente dependente das condições iniciais, de forma que uma mudança ínfima gera resultados mesmo conceitualmente incompatíveis. Para estudar tais processos, o máximo que se pode ser feito são análises estatísticas; e dado que seu comportamento é muito similar ao de um fractal, todo arcabouço de métodos e tecnologias dos mesmos é muito útil para compreensão desse tipo de fenômeno.

Além dessa principal aplicação, os fractais podem ser usados (como é o foco deste trabalho) para identificar imagens dos mais diversos tipos nos experimentos. Por exemplo, no experimento de Young, pode-se utilizar um programa para identificar os tipos de fendas, mas é necessário inserir heurísticas específicas que fossem condizentes com uma intuição manual, para que o computador pudesse fazer esse trabalho. Usando esse tipo de método, seria possível alcançar um método mais assertivo, ainda que estatístico em sua natureza, de avaliar resultados parciais de experimentos que sejam imagens.

Finalmente, é relevante ver como métodos computacionais podem ajudar com experimentos. Isso tem aplicação em praticamente todas as áreas da física; é inimaginável a quantidade de experimentos que poderiam se beneficiar de variações feitas no computador, nas mais diversas etapas (planejamento, execução, processamento de dados0.

Exemplos de artigos e trabalhos que relacionam fractais e física podem ser vistos em [3], [2], [7] e [1].

0.3 Palavras-chave

Geometria fractal, análise de imagens, reconhecimento de padrões.

0.4 O que foi densenvolvido?

Ainda que muitas ainda estejam por vir, uma série de etapas em direção aos objetivos mencionados já foi concluída. Nas seções a seguir, explica-se um pouco todo o trabalho realizado.

0.4.1 Estudo Inicial

Num primeiro momento, foi realizado um breve estudo teórico a respeito de fractais, incluindo sua definição, propriedades, diversas utilizações, casos na natureza e principalmente sobre dimensões; os diversos tipos, as definições, utilizações e métodos para cálculo. Foi dado um enfoque especial para o método de Box Counting, já que esse seria o que viria a ser utilizado.

Para esse estudo, foi utilizada principalmente a tese do Professor ([6]), e também o livro do Falconer ([4]). Como resultado, foi sumarizado o entendimento e dúvidas em uma apresentação de slides realizada ao Professor. Os slides da mesma podem ser vistos nesse link.

0.4.2 Implementação Inicial do Mandelbrot

Decidiu-se que inicialmente seria feita uma implementação própria do fractal conhecido como Mandelbrot, para entender melhor os conceitos na prática. Foi possível ver as belas imagens dos fractais se formando, e como mudar os parâmetros afetava o resultado final. O código em Java escrito (agora em sua última versão) pode ser visto a seguir (a parte relevante, que é a função de Mandelbrot; todo o código, com as outras funções, e coisas auxiliares, bem como tudo o mais que for mencionado neste, pode ser encontrado, de forma livre e aberta, no github):

```
1    public static boolean mand(Complex c) {
2        Complex z = c;
3        double max = 0;
4        int lastMax = 0;
```

```
5           for (int t = 0; t < 100 * ITERATIONS; t++) {
6               double abs = z.abs();
7               if (Double.isInfinite(abs)) {
8                   return false;
9               }
10
11              if (abs > max) {
12                  max = abs;
13                  lastMax = t;
14              } else if (t - lastMax > ITERATIONS) {
15                  return true;
16              }
17
18              z = z.timesPlus(z, c);
19          }
20          return false;
21      }
```

Um exemplo de fractal gerado pode ser visto na Figura 1 (não é apresentado um de resolução tão alta para não estourar o tamanho do PDF).

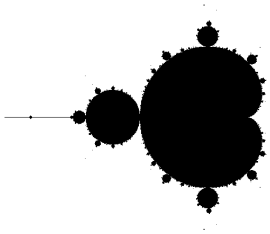

Figura 1: Exemplo de Mandelbrot gerado pelo código em Java.

0.4.3 Implementação Inicial de Box Counting

Logo em seguida, a fim de obter uma melhor compreensão sobre como aumentar ou diminuir a resolução da imagem impacta na sua boa precisão, foi implementado o algoritmo

supradito, Box Counting, para Java também, a fim de medir uma aproximação da dimensão de todos os fractais gerados. O código desse pedaço, que está, também, em sua completude no github, pode ser visto na sequência:

```
    private static double boxCount(Picture picture) {
        int n = (int) (Math.log(picture.size()) / Math.log(2));
        double[][] xs = new double[n][]; // because the library wants it
    this way
        double[] ys = new double[n];
        ProgressBar pb = new ProgressBar("Box Count", n);
        pb.start();
        for (int i = 0; i < n; i++) {
            int r = (int) Math.pow(2, i);
            int squares = countSquares(picture, r);
            xs[i] = new double[]{Math.log(r)};
            ys[i] = Math.log(squares);
            pb.step();
        }
        pb.stop();
        System.out.println("Running MMMQ...");
        double[] params = MMQ.run(xs, ys); // ax+b, returns [a, b]
        return params[0];
    }

    private static int countSquares(Picture picture, int r) {
        int total = 0;
        int size = picture.size() / r;
        for (int i = 0; i < picture.size(); i += size) {
            for (int j = 0; j < picture.size(); j += size) {
                if (countSubSquare(picture, i, j, size)) {
                    total++;
                }
            }
        }
        return total;
    }

    private static boolean countSubSquare(Picture picture, int i, int j,
    int s) {
        for (int dx = 0; dx < s; dx++) {
            for (int dy = 0; dy < s; dy++) {
                if (picture.get(i + dx, j + dy)) {
                    return true;
```

```
38                      }
39                  }
40              }
41              return false;
42      }
```

Observe que utilizou-se uma biblioteca pronta para fazer o MMQ, ou a regressão linear requerida na implementação computacional do algoritmo (nos slides pode-se ver que aquele limite vira uma função y(x) e queremos o coeficiente angular da mesma).

Objetivou-se deixar o código mais elegante possível, e em seguida, foram feitas algumas modificações para o deixar mais performático, pois inicialmente foram encontrados alguns problemas com performance.

Um bom tempo foi gasto analisando e reavaliando o algoritmo, e várias modificações foram feitas a fim de deixar mais eficiente. Finalmente, foi possível produzir uma imagem com resolução de 8192 pixels (bem grande!).

A dimensão real do fractal de Mandelbrot é exatamente 2. Ao rodar o algoritmo para diversas resoluções, é possível observar como aumentar a mesma impacta na precisão da estimativa; a Tabela 1 revela esses resultados.

Tabela 1: Valores de dimensão por resolução

Resolução (pixels)	Dimensão
128	1.2750591691431414
256	1.3705954162797864
512	1.4473073991800987
1024	1.5144765682324572
2048	1.569869732483732
4096	1.6178066227678722
8192	1.6565008486614825

Como pode-se ver, a precisão não foi tão boa, mesmo para a maior resolução obtida, mas há um crescimento inegável em direção ao valor correto conforme o tempo de execução cresce. Com um poder computacional maior, talvez fosse possível obter um resultado mais satisfatório.

0.4.4 Estudo Matlab/Octave

Tendo finalizado essa ambientação inicial com fractais e box counting em Java, que era uma linguagem com a que havia familiaridade, iniciou-se o estudo de MatLab, ou, mais

especificamente, a sua versão open source, gratuita e GNU-friendly Octave (mais detalhes).

Já havia existido um contato inicial com a mesma, mas não um uso mais aprofundado. Essa parte se mostrou mais desafiadora do que previsto, uma vez que o Octave tem um paradigma totalmente diferente das outras linguagens de programação.

Normalmente as linguagens seguem algo em comum, então conhecer uma nova linguagem Orientada a Objetos é relativamente simples, dado que já é sabido uma delas, pois só a sintaxe muda. O mesmo para uma segunda linguagem funcional. Mas Octave se diferenciava bastante de tudo, por isso foi necessário um curso online que (surpreendentemente) os criadores do Matlab disponibilizam gratuitamente. O certificado de conclusão desse curso pode ser visto aqui. Note que foram colocados um nome e email irreais a fim de evitar spam.

0.4.5 Implementação do Mandelbrot em Octave

Tendo realizado um estudo mais aprofundado da linguagem, foi possível re-implementar o Mandelbrot em Octave. Como mencionado, as linguagens são bem diferentes, e foi necessário fazer várias versões para chegar num resultado aceitável, que ficou muito diferente do que em Java.

Graças ao Professor, que deu muitas dicas de como melhor aproveitar o feature set diferenciado que a linguagem oferecia, obteve-se o seguinte código, já na versão final, como implementação do Mandelbrot em Octave:

```
function M = mand(c) # returns whether the complex c belongs to the mand
    set
  ITER = 300; # Number of consecutive iterations that are necessary to
    determine that a constant growth means explosion

  z = c; # current point
  for t = 0:ITER
    if abs(z) > 2
      M = 0;
      return
    endif
    z = z*z + c; # next in the sequence
  endfor
  M = 1; # didn't explode in ITER iterations
endfunction
```

Além disso, implementou-se o código que, utilizando essa função, era capaz de scannear o plano complexo a fim de salvar a imagem final, em branco e preto:

```
function main()

  details = 64 # Size of the result image, in pixels (always square,
    details x details) ; beware : should be a power of 2

  xc = double(0) # x,y coord of the center of the mand set (the center of
    the picture will be this point of the mand set
  yc = double(0)
  frac_size = double(5) # size of the mand we will build; the border of the
    image will have coord 5 units in the mand set
  # e.g., if (x,y) = (0,0) and frac_size = 1, we are going to print the
    mand set, from x = -.5 to +.5, y = -.5 to +.5

  imgu = i; # because I used i in the for and got all mixed up -.-

  picture = zeros(details, details); # fill with ones! (we well zero if the
    dot does not belong)
  for i = 0:(details - 1)
    for j = 0:(details - 1)
      x0 = xc - frac_size / 2 + frac_size * i / details;
      y0 = yc - frac_size / 2 + frac_size * j / details;
      picture(details - j, i + 1) = !mand(x0 + imgu*y0);
    endfor
  endfor

  d = dim(picture)

  imwrite(picture, 'results/fractal.png');
endfunction
```

Adicionou-se também comentários explicando o código de forma geral, para facilitar o entendimento. Finalmente, o código produziu resultados idênticos à versão anterior.

0.4.6 Implementação do BoxCount em Octave

Em seguida, re-implementou-se o BoxCounting, dessa vez em Octave. Esse processo se mostrou bastante trabalhoso, também devido à introdução de alguns bugs que só foram resolvidos um tempo depois.

O código a seguir é fruto dele:

```
function D = dim(frac) # calculates the dimension of the frac with the box
    count method
```

```
  n = log2(columns(frac));
  for i=1:n
    r = pow2(i - 1);
    squares = countSquares(frac, r);
    xs(i) = log(r);
    ys(i) = log(squares);
  endfor
  mmq = (ys'\xs');
  D = mmq(1);
endfunction

function total = countSquares(fractal, r)
  total = 0;
  square_size = columns(fractal) / r;
  ci = 0;
  cj = 0;
  while ci < columns(fractal)
   while cj < columns(fractal)
      if countSubSquare(fractal, ci, cj, square_size) == 1
        total += 1;
      endif
      ci += square_size;
      cj += square_size;
    endwhile
  endwhile
endfunction

function B = countSubSquare(fractal, i, j, s)
  for dx = 1:(s - 1)
    for dy = 1:(s - 1)
      if fractal(i + dx, j + dy) == 0
        B = 1;
        return
      endif
    endfor
  endfor
  B = 0;
endfunction
```

0.4.7 Implementação de Pascal

Dando continuidade aos estudos, em conjunto com o Professor, decidiu-se que seria mais interessante agora trabalhar com algum fractal do qual eu pudesse facilmente mudar a dimensão, a fim de que eu não tivesse que reescrever um novo algoritmo do zero para observar a variância das respostas de acordo com a dimensão esperada.

Para tal, decidiu-se implementar o fractal do Triângulo de Pascal, também conhecido como Sierpinski Triangle. Esse fractal é definido como a imagem formada pelo triângulo de Pascal se tomarmos os resultados módulo k.

Dependendo da escolha desse valor k, se ele for primo, pode-se calcular a dimensão esperada como sendo:

$$dim = 1 + log_k \frac{k+1}{2} \tag{1}$$

Assim, é possível analisar diversos comportamentos, com uma gama maior de parâmetros.

Sendo assim pode-se por fim rodar o novo fractal, para $k = 2$.

O valor esperado seria $d = 1.5850$, e os valores obtidos, conforme era aumentada a precisão, foram

$$d = [1.5000, 1.4815, 1.4287, 1.4175, 1.4096, 1.4069, 1.4057, 1.4057, 1.4062]; \tag{2}$$

0.4.8 Uso do BoxCounting

Para melhor ver os resultados do Box Counting, foi implementada a geração do fractal Conjunto de Cantor. Esse fractal também é muito interessante para os estudos por dois motivos; diferentemente dos anteriores, ele se apresenta num espaço de uma dimensão (uma reta). Além disso, ele também tem dimensão variável, pois nada mais é do que a coleção restante dos pontos da reta quando se retira o pedaço do meio de cada pedaço restante, infinitamente. O pedaço do meio aqui normalmente é o terço central, para $p = 3$, mas podem ser utilizados outros valores.

Um exemplo de imagem gerada pode ser visto na Figura 2. Aqui, vale ressaltar uma consideração importante. Devido a natureza 1D do fractal, sua visualização por nossos olhos é muito difícil. Dessa forma, para representar o fractal, usou-se uma imagem 2D em que todas as linhas são iguais. Isso não significa que a imagem 2D gerada é um fractal, mas sim o conceito 1D associado.

Figura 2: Exemplo de Cantor gerado pelo código em Octave.

Essa imagem foi gerada usando um código bem simples, que pode ser visto a seguir. Não foram feitas muitas optimizações ainda nesse código, uma vez que gerar Cantor é bem rápido, dado sua natureza unidimensional.

```
function main_cantor()
  H = 10 # vertical length for visualization

  function nb = next_block_list(block_list)
    nb = [];
    for block = (block_list*3)
      nb = [nb (block - 4) (block + 2)];
    endfor
  endfunction

  function picture = generate(size, it)
    picture = zeros(H, size);
    block_list = [2];
    blocks = 1;
    for i = 1:it
      blocks = blocks * 3;
      blockSize = size/blocks;
      for block = block_list
        picture(1:H, (block - 1)*blockSize:block*blockSize) = 1;
      endfor
      block_list = next_block_list(block_list);
    endfor
  endfunction

  # size, iterations (size = 3^(it - 1)
  picture = generate(3^9, 8);
  imwrite(picture, 'results/cantor_2d.png');
endfunction
```

Em seguida, foi re-implementado o BoxCounting para funcionar em fractais unidimensionais. Essa alteração foi bem simples, como pode ser observado no código a seguir. Outra alteração necessária mais trabalhosa foi fazer ele funcionar com fractais de tamanhos que não fossem potências exatas de dois, algo que não ocorria antes. Agora, todo fractal gerado precisa ter tamanho de potência de 3.

```
# USAGE
# output_precision(15)
# dim1d(picture)

```

```
function D = dim1d(frac) # calculates the dimension of the frac with the
    box count method
  n = floor(log2(columns(frac)))
  for i=1:n
    r = pow2(i - 1) # amount of squares
    # draw(frac, r);
    squares = countSquares(frac, r)
    xs(i) = log(r);
    ys(i) = log(squares);
  endfor
  mmq = (xs'\ys');
  D = mmq(1);
endfunction

function total = countSquares(fractal, r)
  total = 0;
  N = size(fractal, 2);
  square_size = ceil(N / r);
  for ci=0:square_size:(N-1)
    if countSubSquare(fractal, ci, square_size) == 1
      total += 1;
    endif
  endfor
endfunction

function B = countSubSquare(fractal, i, s)
  t = s - 1;
  if i + t > size(fractal, 2)
    t = size(fractal, 2) - i;
  endif
  for dx = 1:t
    if fractal(1, i + dx) == 0
      B = 1;
      return
    endif
  endfor
  B = 0;
endfunction
```

Uma abordagem mais naïve para esse problema foi mudar o BoxCounting para dividir tudo em 3, ao invés de 2. Para isso, bastou mudar o log e a potência:

```
function R = log3(n)
```

```
  R = log(n) / log (3);
endfunction

function R = pow3(n)
  R = 3**n;
endfunction
```

Essa abordagem, não surpreendentemente, dá o valor exato da dimensão esperada com uma precisão espantosa. De fato, o valor esperado de $D = \frac{\ln(2)}{\ln(3)} = 0.6309297535714575$, e o valor encontrado, para qualquer precisão colocada, foi de

$$D = 6.30929753571458e-01 \tag{3}$$

Ou seja, muito próximo. Mas ao fazer dessa forma, está-se utilizando de uma propriedade do fractal que não poderia ser conhecida à priori. Dessa forma, o programa foi executado com o BoxCounting convencional, na base 2, que sempre terá um pequeno erro devido a precisão escolhida. Os valores obtidos podem ser vistos na Tabela 2.

Tabela 2: Valores de dimensão por resolução

r	Dimensão
5	$7.47469193143058e-01$
6	$7.28197916108039e-01$
8	$7.02180068614000e-01$
11	$6.98544593485581e-01$

Como pode-se ver, o valor rapidamente decai para o esperado, ainda que com um erro, conforme a resolução aumenta.

As Figuras 3 e 4 mostram em vermelho as divisões utilizadas para o algoritmo, tanto para a base 2 como para a base 3. Nelas, fica evidenciado a diferença fundamental entre os dois. Elas foram salvas pelo código visto na sequência, que foi acoplado a cada etapa da função dim1d. Para facilitar a visualização, foi acoplada imagens com resolução mais baixa que as utilizadas para as contas e com tamanho vertical estendido de $H = 10$.

```
function draw(frac, r)
    C=size(frac, 1);
    L=columns(frac);
    copy = frac();
    square_size = ceil(L/r)
    for j=1:square_size:L
```

```
      copy(1:C,j) = 2;
    endfor

    red = zeros(C, L);
    green = zeros(C, L);
    blue = zeros(C, L);
    for ai=1:C
      for aj=1:L
        if copy(ai, aj) == 1
          red(ai, aj) = 255; green(ai, aj) = 255; blue(ai, aj) = 255;
        else if copy(ai, aj) == 2
          red(ai, aj) = 255;
        endif endif
      endfor
    endfor
    truecolor_image = cat(3, red, green, blue);
    imwrite(truecolor_image, strcat('results/cantor_p_', num2str(r), '.png')
   ));
endfunction
```


Figura 3: Exemplo divisão do BoxCounting para $t = 3^3$.

Figura 4: Exemplo divisão do BoxCounting para $t = 2^4$.

0.4.9 Estudo Teórico

Depois disso, foi feito um segundo estudo teórico, dessa vez utilizando dois livros do Falconer, [4] e [5].

Nesse estudo, o background matemático foi grandemente ampliado, e estudou-se tópicos como: σ-field, measure, upper and lower limit, outer measure, Borel Set, diameter, δ-cover, e por fim dimensão de Hausdorff.

Com esses conhecimentos aprofundados, está sendo possível estudar o erro no algoritmo de BoxCounting e tentar descobrir um método de medí-lo, conforme estudado em Cálculo Numérico.

0.5 Opinião do Orientador

Meu orientador concorda com o expressado neste Relatório Pré-Final e deu a seguinte opinião:

No decorrer dos últimos meses, o aluno mostrou-se dedicado e interessado no estudo dos novos temas que foram propostos, relacionados a aspectos teóricos e práticos da geometria fractal na análise de imagens digitais.

No geral, estou bastante satisfeito com o compromisso do aluno e com os resultados alcançados neste tempo relativamente curto de trabalho em conjunto. O aluno me parece maduro o suficiente para dar continuidade a um projeto acadêmico e futuramente, se for de seu interesse, ingressar em um programa de pós-graduação, com potencial para o desenvolvimento de uma pesquisa frutífera.

Bibliografia

[1] AMICUCCI, F. Fractais na sala de aula. http://www.ifi.unicamp.br/~lunazzi/F530_F590_F690_F809_F895/F809/F609_2013_sem1/FabioA-Kleinke_RF2_F609.pdf.

[2] CALZETTI, D.; GIAVALISCO, M. Fractal structures and the angular correlation function of galaxies. *Vistas in Astronomy, v. 33, n. 3/4* (1990), 295–304.

[3] CHAPPARD, D.; DEGASNE, I. H. G. L. E. A. M. B. M. Image analysis measurements of roughness by texture and fractal analysis correlate with contact profilometry. *Biomaterials, v. 24, n. 8* (2003), 1399–1407.

[4] Falconer, K. *Fractal Geometry: Mathematical Foundations and Applications.* Wiley, 1997.

[5] Falconer, K. J. *The geometry of fractal sets*, vol. 85. Cambridge university press, 1986.

[6] Florindo, J. B. *Descritores fractais aplicados à análise de texturas.* PhD thesis, Universidade de São Paulo.

[7] FLORINDO, J. B.; SIKORA, M. P. E. B. O. Multiscale fractal descriptors applied to nanoscale images. *Journal of Superconductivity and Novel Magnetism, 6p.* (2012).

[8] McAndrew, A. An introduction to digital image processing with matlab notes for scm2511 image processing. *School of Computer Science and Mathematics, Victoria University of Technology* (2004), 264.