Fernanda Tostes Marana - no.usp: 4471070
Matheus Aparecido do Carmo Alves - no.usp: 9791114
Sylviane da Silva Vitor - Aluna Especial

Projeto da Disciplina Computação Gráfica: Desempenho da Conversão Matricial de Circunferências

Professora Doutora Agma J. M. Traima
Universidade de São Paulo - USP
Instituto de Ciências Matemáticas e de Computação - ICMC
Bacharelado em Ciências de Computação

São Carlos, SP
07 de Abril de 2018

Sumário

1 Introdução

Arcos de circunferência são considerados primitivas gráficas elementares e seu traçado requer a construção de algoritmos que determinem, na matriz de *pixels* da superfície de exibição, quais *pixels* devem ser alterados - calculando seu valor para simular a aparência dessa primitiva.

Serão apresentadas 3 abordagens para se traçar uma circunferência: a técnica do Meio-Ponto, a Conversão pela Equação Implícita da Circunferência e a Conversão por Coordenadas Esféricas. O presente relatório objetiva realizar uma análise quantitativa e qualitativa da implementação desses três algoritmos de conversão matricial de circunferências. O objetivo é determinar a eficiência dessas técnicas, bem como seu desempenho.

O relatório está estruturado como segue: na Seção 2, encontram-se instruções de compilação e observações acerca do ambiente de compilação utilizado nos testes. Na Seção 3, são descritas metodologia e implementação utilizando a técnica do Meio-Ponto. Na Seção 4, o mesmo é descrito, utilizando a equação implícita da circunferência. Na Seção 5, utilizando a equação implícita da circunferência. Na Seção 6, são apresentados os resultados e na Seção 7 é apresentada uma breve conclusão.

2 Instruções e Ambiente de Compilação

A implementação dos algoritmos foi feita na linguagem *Python* (versão 3.6) de programação, utilizando como principais bibliotecas suporte a *Pygame* (biblioteca para modelagem gráfica de jogos multiplataforma) e *Math* (biblioteca de funções matemáticas). A primeira foi usada para a configuração da interface gráfica do trabalho, provendo como método principal o *pygame.gfxdraw.pixel()* que desenha um pixel na posição especificada do *display* (já configurado com resolução específica). Já a segunda foi utilizada para realizar operações que envolvessem raízes, potências e parâmetros trigonométricos.

Para compilação, é necessário somente a correta instalação das bibliotecas mencionadas e a versão correta da linguagem utilizada. Sua execução pode ser feita via terminal Linux pelo comando (considerando que o usuário se encontra na pasta local):

```
1        $ python main.py
```

O ambiente usado para testes possui as seguintes características de arquitetura:

(1) Sistema Operacional: Linux - Ubuntu 16.04 LTS

(2) Memória 3,8GiB

(3) Processador: Itel Core i3-3110M CPU @ 2.40GHz x 4

(4) Gráficos: Intel Ivybridge Mobile

(5) Tipo de sistema: 64-bits

(6) Disco: 106,2 GB

Os testes foram rodados com o mínimo de processos paralelos à aplicação possível.

3 Considerações de Implementação

Para implementar as técnicas, foi considerado apenas o primeiro quadrante da circunferência e os demais foram obtidos pela propriedade de reflexão, utilizando o método SimetricCirclePoints(x, y, cx, cy, color) que espelha, as coordenadas x e y. Além disso, foi definido um display de resolução 400x400 para visualização, limitando o raio máximo a 200. O tamanho do raio pode ser redefinido, bem como as coordenadas do centro.

4 Meio-Ponto

4.1 Metodologia

A técnica do Meio-Ponto, utilizada na implementação do método MidPointCirc(r, cx, cy, color) é baseada na abordagem incremental, sendo r o raio da circunferência, cx e cy as coordenadas do centro do circulo (abscissa e ordenadas respectivamente) e *color* a cor do circulo (sendo por padrão branco *(255,255,255)*).

Resumidamente, o algoritmo realiza o traçando escolhendo a cada passo 2 *pixels* para marcar com base no sinal da variável de decisão d, calculada na iteração anterior; a

seguir a variável de decisão é atualizada, adicionando-se o incremento correspondente ao pixel escolhido; em nossa abordagem um dos incrementos representa um "passo"à Sudeste SE e o outro um passo a Leste E.

Se o valor da variável de decisão for negativo, isto é, escolhemos o pixel à leste, o incremento de d será de E = d + 2 * x + 3 e o deslocamento ocorre apenas em x. Caso contrário, escolhendo o pixel à sudeste, temos o incremento SE =d + 2 * (x - y) + 5.

Desta forma, a cada iteração um pixel é escolhido, pintado e o valor da variável de decisão é recalculado incrementalmente, sem a necessidade de recalcular o valor da função no ponto.

4.2 Implementação

Conforme o algoritmo proposto por *Foley*, a implementação do algoritmo do Meio-Ponto para conversão matricial de circunferências segue abaixo.

```
1         def MidPointCirc(r,cx,cy,color):
            # Initializing globals
3           ResetDisp()
            global n_pixels
5           n_pixels = 0

7           # Initializing the local variables
            x = 0
9           y = r
            d = 5 / 4 - r
11
            # Starting the drawn
13          SimetricCirclePoints(x, y, cx, cy, color)

15          # Fitting the drawn with Middle Point Transform
            while (y > x):
17              if (d < 0):
                    d = d + 2 * x + 3
19                  x = x + 1
                else:
21                  d = d + 2 * (x - y) + 5
                    x = x + 1
23                  y = y - 1
                SimetricCirclePoints(x, y, cx, cy, color)
25
                pygame.display.flip()                   #update display
27
            return(n_pixels)
```

5 Equação Implícita da Circunferência

5.1 Metodologia

O algoritmo utilizado segue a implementação mais intuitiva para a conversão matricial, isto é, partindo de um x e y iniciais, incrementa-se o valor de x e calcula-se o valor de y correspondente, seguindo a equação da circunferência: y = r2-x2. Desta forma, a cada iteração os valores de x e y são alterados e o pixel correspondente é pintado.

5.2 Implementação

A implementação segue abaixo:

```
        def ImplEqCirc(r,cx,cy,color):
2         # Initializing globals
          ResetDisp()
4         global n_pixels
          n_pixels = 0
6
          # Initializing the local variables
8         x = 0
          y = r
10
          # Starting the drawn
12        SimetricCirclePoints(x, y, cx, cy, color)

14        # Fitting the drawn with Implicit Equation
          while (y > x):
16            x = x + 1
              y = int(math.sqrt(math.pow(r, 2) - math.pow(x, 2)))
18            SimetricCirclePoints(x, y, cx, cy, color)
              pygame.display.flip()
20
          return(n_pixels)
```

6 Coordenadas Esféricas

6.1 Metodologia

Analogamente à implementação que utiliza a equação da circunferência, a conversão matricial por coordenadas esféricas busca traçar a circunferência incrementando o ângulo entre um ponto na extremidade do raio do circulo e,a cada iteração, alterando a cor dos *pixels* neste ponto. Os valores de x e y são calculados através do cálculo de transformação de coordenada: $x = r.\ cos(\theta)$ e $y = r.\ sen(\theta)$.

6.2 Implementação

A implementação segue abaixo:

```
    def SphereCoordCirc(r,cx,cy,color):
        # Initializing globals
        ResetDisp()
        global n_pixels
        n_pixels = 0

        # Initializing the local variables
        theta = 0.0
        x = r
        y = 0

        # Starting the drawn
        SimetricCirclePoints(x, y, cx, cy, color)

        # Fitting the drawn with Sphere Coordinates
        while (x > y):
            theta = theta + 0.1

            if(x != int(r*math.cos(math.radians(theta))) or y != int(r*math
                .sin(math.radians(theta)))):
                x = int(r*math.cos(math.radians(theta)))
                y = int(r*math.sin(math.radians(theta)))
                SimetricCirclePoints(x, y, cx, cy, color)

                pygame.display.flip()                    #update display

        return(n_pixels)
```

7 Resultados

Os testes realizados dizem respeito à execução de cada algoritmo 100 vezes, para diferentes valores de raio. Para cada caso, foram calculados o total de pixels desenhados e o tempo médio de execução. A circunferência foi centrada em (200, 200) para ser visualizada no centro do display. O resultado dos testes está descrito nas tabelas abaixo:

Tabela 1 – Algoritmo Middle-Point.

Raio da Circunferência	**Tempo médio de execução** (s)	**Pixels usados**
5	0.0070177293	40
10	0.0083651639	64
20	0.0139344859	120
50	0.0382401252	296
70	0.0425977850	408
100	0.0699393678	576
120	0.068290834	688
150	0.0870565724	856
170	0.0980760121	968
200	0.0956490493	1144

Tabela 2 – Algoritmo Implicit Equation.

Raio da Circunferência	**Tempo médio de execução** (s)	**Pixels usados**
5	0.0067022157	40
10	0.0090110707	64
20	0.0143307400	120
50	0.0368202567	288
70	0.0414624906	400
100	0.0744559693	576
120	0.0701114583	688
150	0.0865880704	856
170	0.0976769042	968
200	0.1101042414	1136

Tabela 3 – Algoritmo Sphere Coordinates.

Raio da Circunferência	**Tempo médio de execução** (s)	**Pixels usados**
5	0.1598183346	40
10	0.1877284837	80
20	0.187704043462	160
50	0.2421295166	392
70	0.2025003362	544
100	0.2554195023	776
120	0.2284564447	928
150	0.2360477622	1136
170	0.2501374102	1256
200	0.2808123732	1480

No que diz respeito ao traçado, os algoritmos apresentaram resultados como mostrado nas figuras que seguem, para os valores 10, 50 e 150 de raio:

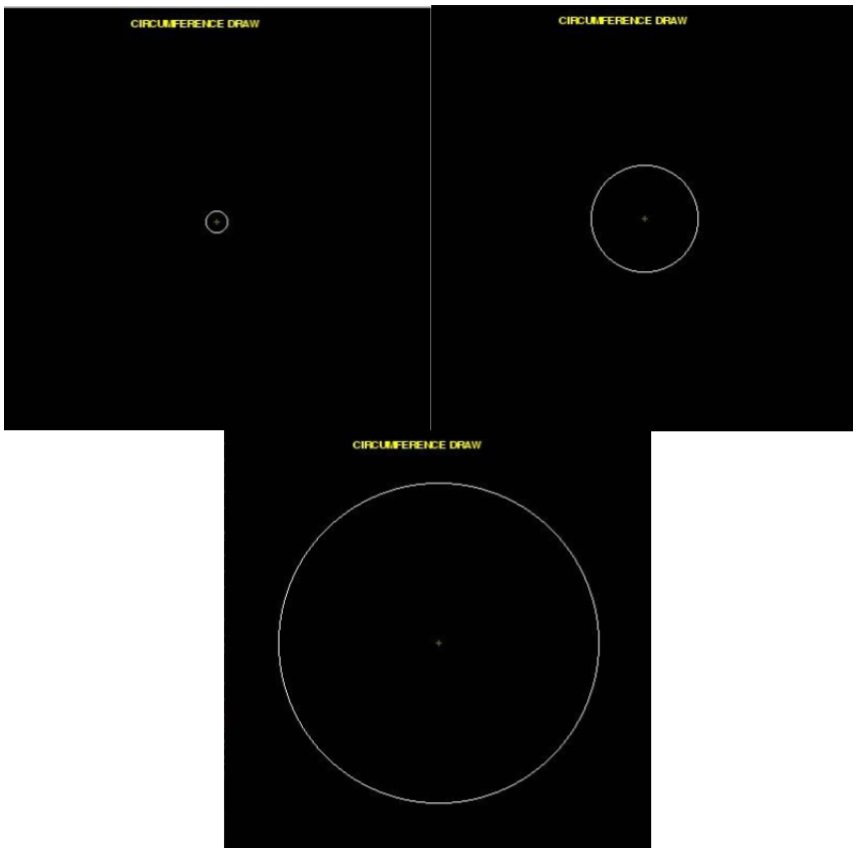

Figura 1 – Cincunferências Middle-Point.

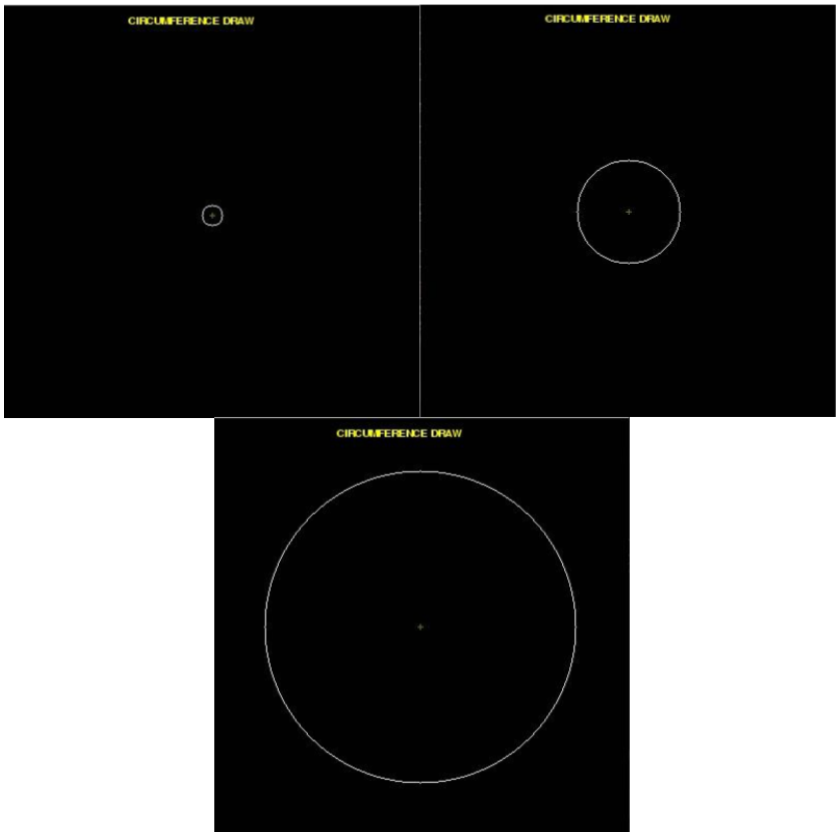

Figura 2 – Cincunferências Implicit Equation.

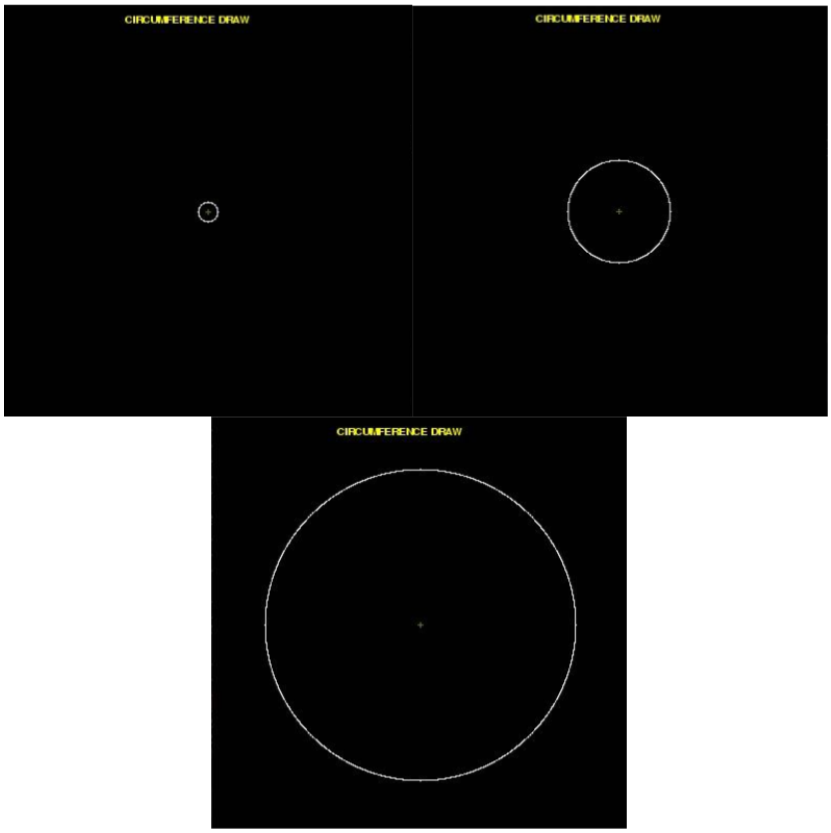

Figura 3 – Cincunferências Sphere Coordenates.

Graficamente, podemos observar o comportamento descrito acima como mostram as figuras 4 e 5:

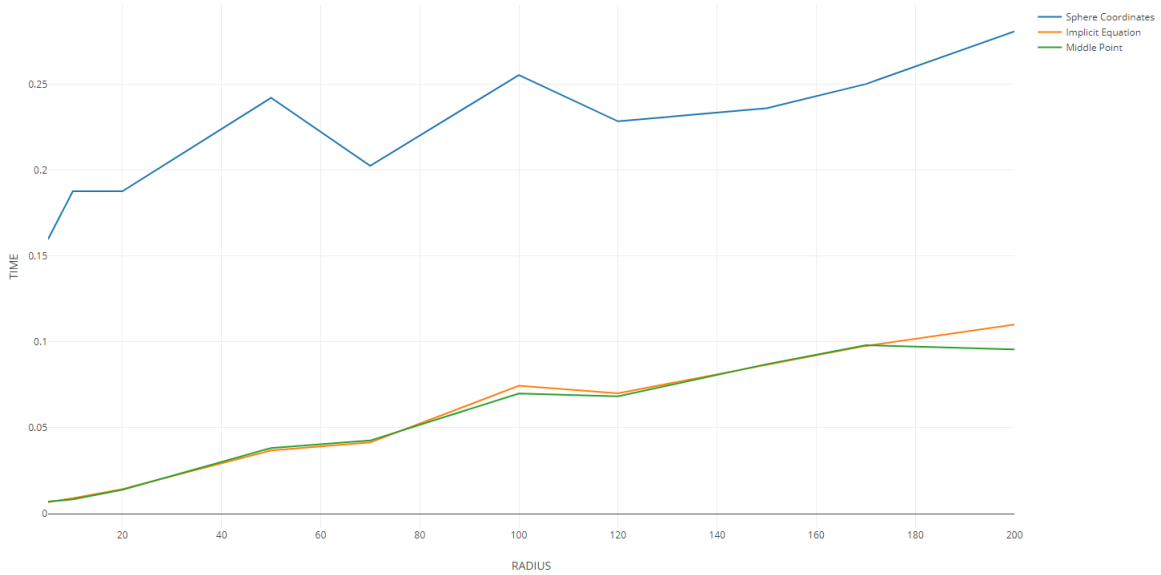

Figura 4 – Gráfico apresentando a relação entre tamanho do raio e seu tempo de execução.

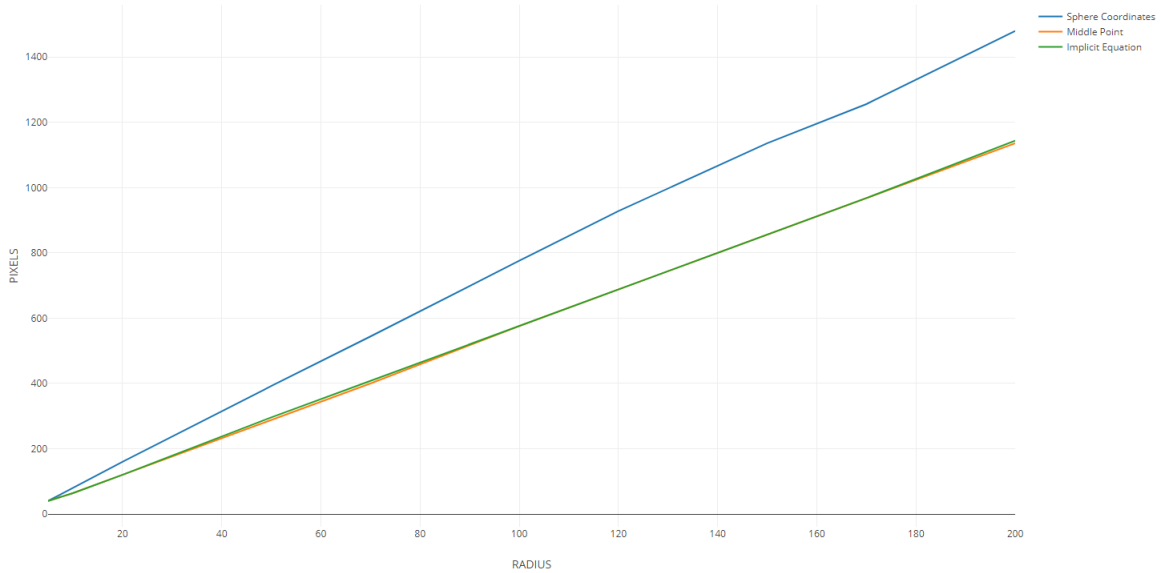

Figura 5 – Gráfico apresentando a relação entre tamanho do raio e número de pixels usados.

8 Discussão

Em relação ao tempo médio para o desenho do traçado, conforme as Tabelas (1, 2 e 3) é possível verificar que o algoritmo *Middle-Point* obtém melhores resultados, para quaisquer tamanhos de raio. Isto ocorre devido ao fato da grande diferença de tempo para executar operações como cálculo de ponto flutuante e senos e cossenos, em detrimento a aritmética inteira. Como mostrado nas Seções 3 a 3, a lógica de implementação das três abordagens é similar: um *loop* principal onde são calculadas as coordenadas. Entretanto, no caso do Meio-Ponto, esses valores são determinados de forma incremental, garantindo assim a eficiência.

Já no que diz respeito à qualidade do traçado, como mostram as Figuras 1 a 3, para um raio suficientemente pequeno, ainda que seja usada a mesma quantidade de *pixels* - 40 (conforme Tabelas), o traçado difere. No caso do Meio-Ponto, a circunferência tem um aspecto mais contínuo e em alguns pontos é possível observar serrilhamento leve, inerente à algoritmos de conversão. Por outro lado, na implementação que utiliza a Equação Implícita, ainda que seja usada a propriedade de reflexão, a circunferência fica bastante distorcida, se aproximando de uma elipse. Isso ocorre porque, conforme x se aproxima de r, a tangente tende a infinito, causando esse efeito. Como o quadrante é refletido, o mesmo é observado no seu simétrico, quando x se aproxima de -r. O algoritmo que utiliza coordenadas esféricas, por sua vez, traz também o aspecto serrilhado nos pontos dos eixos de simetria, devido ao uso de senos e cossenos.

Para um raio médio, como mostrado na Figura 1, é possível perceber que esses efeitos se atenuam, mas ainda são aparentes. Por fim, para um raio suficientemente grande, quase não notamos diferenças na qualidade do traçado das três técnicas, embora seja necessário ressaltar a diferença na quantidade de *pixels* usados. Nos algoritmos do Meio-Ponto e usando a Equação da Circunferência, são usados 856 *pixels* (conforme Tabelas 1 e 2) enquanto o algoritmo de Coordenadas Esféricas exige 1136 *pixels* para atingir resultado semelhante, mostrando, mais uma vez, sua ineficiência em comparação aos demais.