UNIP

UNIVERSIDADE PAULISTA

Computação Gráfica

Autores: Prof. Hugo Gava Insua
Prof. Eduardo Seige Ianaguivara
Colaboradoras: Profa. Vanessa Lessa
Profa. Larissa Rodrigues Damiani

Professores conteudistas: Hugo Gava Insua / Eduardo Seige Ianaguivara

Hugo Gava Insua

Mestre pela UNIP em Engenharia de Produção, especialista em Ensino de Matemática pela Universidade Cruzeiro do Sul (2017) e graduado em Matemática pela Universidade Metodista de São Paulo (2000). É professor do curso de Ciência da Computação e Engenharia da Computação da UNIP, ministrando as seguintes disciplinas: *Matemática Discreta*, *Cálculo para Computação*, Álgebra Linear, *Estatística*, *Computação Gráfica*, *Processamento Digital de Imagem*, *Linguagens Formais e Autômatos* e *Aspectos Teóricos da Computação*. Atuou como professor na rede oficial e privada de Ensino Fundamental e Médio do Estado de São Paulo.

Eduardo Seige Ianaguivara

Doutor e mestre em Engenharia Biomédica pela Universidade de Mogi das Cruzes (2016), com Ênfase em Processamento de Sinais e Imagens Médicas. Especialista em Gerenciamento de Projetos pela Universidade de Mogi das Cruzes (2011) e em Maçonologia: História e Filosofia pela Uninter (2018) e graduado em Tecnologia em Redes de Computadores pela Universidade de Mogi das Cruzes em 2009. Atua há mais de 10 anos como docente dos cursos de Desenvolvimento de Sistemas e Produção de Jogos Computacionais, com foco na aplicação de estímulos coloridos na retenção e avaliação da atenção/destreza.

Sumário

Computação Gráfica

APRESENTAÇÃO

Em ciência da computação, há vários desafios no processamento de imagens, análise e representação das informações. As atividades realizadas pelos profissionais que trabalham com computação gráfica ajudarão outros profissionais a abstrair melhor as ideias, apresentar as informações provenientes de uma massa de dados e a simular graficamente e em tempo real a interação entre materiais, objetos e diferentes fluidos com base em modelos matemáticos.

Este livro-texto tem foco em computação gráfica, e o objetivo da disciplina é aproximar/orientar o aluno sobre conceitos, técnicas e métodos de processamento, análise e visualização de imagens em aplicações gráficas ou que demandam tais recursos.

Você terá a oportunidade de compreender os conceitos que envolvem a computação gráfica e seus componentes. Serão acentuadas a composição de imagens, as representações bidimensionais e tridimensionais, operações booleanas aplicadas aos objetos gráficos e suas diferentes técnicas de representação e separação/classificação.

Usa-se linguagem simples e direta neste livro-texto, como se houvesse uma conversa entre o autor e o leitor. Adicionalmente, as figuras auxiliam a entender os tópicos desenvolvidos. Os itens Observação e Lembrete vão ajudá-lo a solucionar eventuais dúvidas. Por sua vez, os Saiba mais ampliarão seus conhecimentos. Também há muitos Exemplos de aplicação, resolvidos em detalhes, fazendo que você absorva os assuntos abordados.

INTRODUÇÃO

A computação gráfica envolve uma série de conceitos importantes para o mercado de trabalho e, consequentemente, para a empresa que trabalhar com ela. São subáreas vitais da referida área de conhecimento a síntese, o processamento e a análise de imagens.

Na síntese de imagens, os objetos criados em computador são representados a partir de modelos matemáticos, em especial, dos conhecimentos advindos da geometria. Portanto, as representações geométricas dos objetos visam facilitar a compreensão de uma massa de dados (representação) ou mesmo através de simulação destacar dados dinâmicos. São exemplos de síntese de imagem: a elaboração de um gráfico representativo da distribuição de frequências de um determinado evento registrado em uma planilha, o registro digital de uma imagem por meio de um scanner ou câmera e a criação de imagem de uma cena para um filme de animação.

O processamento de imagens envolve o tratamento de imagens digitais, melhorando certas características que são de interesse para o usuário. Podemos citar como exemplos: a restauração de uma fotografia antiga utilizando meios digitais, o realce de detalhes de uma imagem de uma câmera de segurança e a vetorização de uma imagem cartográfica.

Já a análise de imagens considera a especificação dos componentes da imagem a partir de sua representação visual. Ou seja, após seu processamento digital, é possível extrair informações importantes, como: o reconhecimento de caracteres em um texto digitalizado (OCR – Optical Character Recognition), o estudo de manchas urbanas, áreas de desmatamento e levantamento topográfico a partir de imagens de satélite ou aerofotogrametria.

O conteúdo deste livro-texto foi dividido em duas unidades. Na primeira, serão abordados os conceitos básicos de terminologia, as primitivas gráficas em duas dimensões e as rasterizações de linhas e curvas. Na segunda, serão estudados conceitos como a síntese de cores, as transformações geométricas em duas e três dimensões, a concatenação de transformações geométricas, a representação e a modelagem de primitivas gráficas 3D (tridimensionais).

Esperamos que você tenha uma boa leitura e se sinta motivado a ler e conhecer mais sobre computação gráfica.

Bons estudos.

Unidade I

Inicialmente, definiremos os princípios da computação gráfica, seus elementos básicos de hardware ou conceituais para sua prática, por exemplo, para entender as primitivas gráficas, as bibliotecas necessárias, as aplicações e os componentes em duas dimensões. Também aplicaremos algoritmos de rasterização de linhas e curvas.

1 CONCEITOS BÁSICOS E TERMINOLOGIA

Devemos entender computação gráfica como uma subárea da ciência da computação que está em constante desenvolvimento. Dessa forma, não há uma definição pronta do que seja a computação gráfica, entretanto, podemos tentar conceituá-la de diferentes maneiras.

A partir dos conceitos de suas subáreas (síntese, processamento e análise de imagens), ela é parte da ciência da computação que estuda a geração, manipulação e interpretação de modelos matemáticos, na forma de imagens, utilizando o computador. Outra tentativa de conceituar computação gráfica é entendê-la como em um conjunto de métodos e técnicas de converter dados para um dispositivo gráfico via computador e ainda como "a área da ciência da computação que estuda a transformação dos dados em imagem. Essa aplicação estende-se à recriação visual do mundo real por intermédio de fórmulas matemáticas e algoritmos complexos" (SZESZ JÚNIOR *et al.*, s.d., p. 3).

A partir do entendimento geral da computação gráfica, podemos progredir para sua composição/propriedades. Na figura a seguir, podemos visualizar uma ilustração sobre os elementos que a compõem. Destacam-se as aplicações que envolvem imagens e dados, e nesse contexto, são aplicados os componentes de processamento de imagens, visualização, processamento de dados e visão.

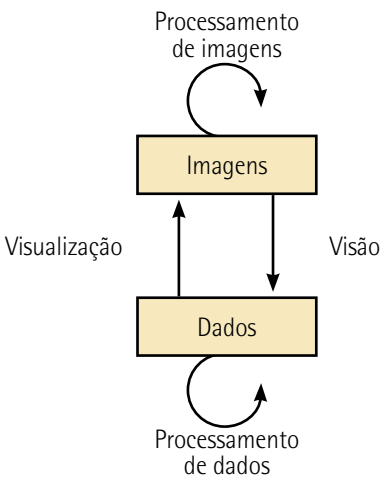


Figura 1 – Estrutura principal da computação gráfica

Com base na figura anterior, devemos definir seus componentes com relação à computação gráfica. Um dado dentro da estrutura computacional representa a menor unidade mensurável; de forma descontextualizada, não representaria nada, por exemplo, ao observar uma planilha do Excel sem os rótulos descritivos, não entenderíamos sua aplicação ou objetivo. A informação é o dado contextualizado, isto é, em uma massa de dados teríamos os rótulos de cada coluna (campo). Já o conhecimento é a aplicação da informação com componentes condicionais e filtros de informação (objetivo).

Uma imagem computacional pode ser descrita como um conjunto finito de números inteiros (pontos) de forma vetorial ou matricial.

O processamento de imagens envolve técnicas de manipulação para a melhoria de suas características, como bordas/serrilhamento, realce, aplicação de filtros e a detecção de formas, cores e bordas. Outras características também são contempladas nessa área, como melhoria do contraste, foco, diminuição de ruídos e distorções. No mercado de trabalho, encontraremos várias aplicações de processamento para as áreas biológica, pesquisa e médica.

A visualização envolve os recursos necessários para a interpretação e construção de imagens através de elementos como linhas, áreas, textos etc. Visão é a obtenção da descrição da imagem digital obtida que poderá ser fornecida por um conversor analógico/digital.

Com relação aos profissionais, a área de computação gráfica envolve engenheiros, arquitetos, matemáticos, artistas, médicos, classificando-se em:

- **Desenvolvedores de ferramentas/bibliotecas**: desenvolvimento de sistemas gráficos como OpenGL e OpenCL.
- **Programadores de APIs (aplicações)**: linguagens de programação para criação de aplicações gráficas, como VRML (Virtual Reality Modelling Language), C, C++ e Python.
- **Customizadores**: adaptação de softwares existentes para outras aplicações.
- **Usuários**: as pessoas que utilizarão a aplicação, os usuários finais.

Quanto à utilização da computação gráfica no mercado de trabalho, podemos visualizar no quadro a seguir várias aplicações e suas áreas de conhecimento/acadêmicas. São associadas as aplicações da computação gráfica com diversas áreas, o que evidencia seu caráter multidisciplinar.

Quadro 1

Área	Aplicações
Arte	Desenho digital, animação 3D, modelagem 3D etc.
Medicina	Processamento de imagens médicas, diagnóstico, estudos etc.
Arquitetura	Maquetes virtuais, projetos, simulações de materiais etc.
Engenharia	Projeto de peças, simulações de fluídos, prototipação etc.
Geografia	Processamento de imagens, georreferenciamento etc.
Meteorologia	Modelagem e visualização de dados, reconhecimento de padrões etc.
Astronomia	Síntese de imagens, tratamento de imagens, reconhecimento de padrões etc.
Marketing	Tratamento de imagens, reconhecimento de imagens, efeitos especiais etc.
Segurança Pública	Reconhecimento de padrões, treinamento etc.
Indústria	Treinamento, controle de qualidade, projetos, modelagem de dados, animação etc.
Turismo	Maquetes virtuais, mapas, georreferenciamento etc.
Moda	Modelagem 2D/3D, renderização, estamparia etc.
Lazer	Jogos computadorizados, efeitos visuais, propagandas, animações etc.
Processamento de dados	Interfaces humano-computador, projetos, mineração de dados, visualização de dados etc.
Psicologia	Avaliação de transtornos comportamentais, testes de reconhecimento de formas e cores etc.
Educação	Ensino, avaliação de reconhecimento de formas, textos e cores etc.

Adaptado de: Azevedo e Conci (2003a, p. 9).

1.1 Origens da computação gráfica

A escala temporal nos ajuda a identificar oportunidades e direções de investigação e aplicação. Algumas das fundações que merecem destaque são:

- **Euclides (300-250 a.C.)**: desenvolveu toda a geometria que norteou seu desenvolvimento até o século XVIII.
- **Brunelleschi (1377-1446)**: arquiteto e escultor italiano que usou de forma criativa a noção de percepção visual e criou em 1425 a perspectiva.
- **Descartes (1596-1650)**: matemático e filósofo francês que formulou a geometria analítica e os sistemas de coordenadas 2D e 3D.

- **Euler (1707-1783)**: o mais produtivo matemático do século XVIII, que, entre outros, criou o conceito de senos, tangentes, a expressão que relaciona o número de vértices, arestas e faces de poliedros.

- **Monge (1746-1818)**: matemático francês que desenvolveu a geometria descritiva como um ramo da geometria.

- **Sylvester (1814-1897)**: matemático inglês que inventou as matrizes e a notação matricial, uma das ferramentas mais comuns da computação gráfica.

- **Hermite (1822-1901)**: matemático francês que provou a transcendência do número (usado como base para os logaritmos naturais) e desenvolveu funções elípticas e curvas.

Continuando nossa escala temporal, podemos identificar os aspectos de mudança que são considerados marcos da investigação científica e suas principais aplicações nas indústrias e na sociedade.

- Em 1885, iniciou-se o desenvolvimento da tecnologia do tubo de raios catódicos.

- Em 1927, a indústria cinematográfica definiu o padrão de 24 imagens/segundo.

- Em 1930, P. e W. Mauchly construíram o primeiro computador, chamado ENIAC.

- Em 1938, Valensi propôs o tubo de raios catódicos colorido.

- Em 1947, os Bell Labs inventam o transistor.

- Em 1950, Laposky criou as primeiras obras de arte com bases tecnológicas usando um efeito de um osciloscópio.

- Em 1955, surgiu o sistema Sage de monitoramento aéreo.

- Em 1956, o MIT construiu o primeiro computador totalmente transistorizado.

- Em 1959, foi cunhado o termo computer graphics, criado por L. Hudson, da Boeing.

- No final da década de 1950, as universidades e empresas americanas, como a Boeing, começam a usar computadores para testar ideias e novas aplicações.

- Em 1960, foi lançado o primeiro computador comercial, o DEC PDP-1.

- Em 1961, foi criado no MIT o primeiro jogo de computador (Spacewars) para o DEC PDP-1.

- Em 1963, Sutherland apresentou um sistema de desenho interativo de primitivas gráficas 2D baseado em caneta luminosa.

- Em 1963, Englebart inventou o dispositivo de interação mouse.
- Em 1963, Zajac produziu nos laboratórios da Bell o primeiro filme gerado por computador (imagens formadas de linhas e texto).
- Em 1963, surgiu o primeiro sistema comercial de CAD (DAC-1).
- Em 1963, Coons inventou a teoria de representação de superfícies curvas através de retalhos baseados em aproximações polinomiais.
- Em 1965, Roberts criou um algoritmo de remoção de partes invisíveis de segmentos de reta e introduziu a noção de coordenadas homogêneas na representação geométrica de objetos.
- Em 1966, foi lançado no mercado o primeiro console caseiro de jogos, o Odissey.
- Em 1966, surgiu a primeira empresa de produção computacional de animações e efeitos especiais, a MAGI.
- Em 1967, Rougelet criou o primeiro simulador de voo interativo da Nasa.
- Em 1968, foi fundada a Intel.
- Em 1969, a MAGI produziu para a IBM o primeiro comercial baseado em técnicas de computação gráfica.
- Em 1969, foi criado entre os grupos da ACM o Special Interest Group on Graphics (SIGGRAPH).
- Em 1969, nasceu a ARPANET, rede precursora da internet.
- Em 1969, nos laboratórios da Bell, foi construída a primeira matriz de pixels (cada pixel representado por 3 bits).
- Em 1972, A. Kay, no Xerox PARC, produziu o computador gráfico Alto.
- Em 1972, Bushnell fundou a empresa Atari.
- Em 1973, Metcalf desenvolveu a tecnologia Ethernet. No mesmo ano, foi editado o primeiro livro que aborda detalhadamente os algoritmos e métodos da computação gráfica, obra de Newman e Sproull.
- Em 1977, a Academia de Artes e Ciências Cinematográficas de Hollywood criou a categoria de Oscar de Efeitos Especiais.

- Em 1979, G. Lucas contratou Catmull, Ray Smith e outros para uma nova empresa denominada Lucas Film.

- Em 1974, Catmull desenvolveu o algoritmo Z-Buffer.

A partir do algoritmo de Z-Buffer e com o lançamento do PC no início da década de 1980, surgiram uma infinidade de aplicações e filmes baseados em computador. O mercado da computação gráfica atingiu seu estágio de maturidade, apresentando um grande crescimento com produções realistas e técnicas avançadas de iluminação e modelagem. Foram exploradas outras possibilidades de geometrias além do espaço tridimensional, que são utilizadas com uma frequência cada vez maior pelas pessoas que trabalham com arte, computação e visualização científica.

1.2 Arquitetura de sistemas gráficos (o hardware gráfico)

A arquitetura de sistemas gráficos é o conjunto de componentes e técnicas utilizadas para projetar e implementar sistemas que produzem imagens em tempo real. Esses sistemas são usados em uma ampla gama de aplicações, como jogos, animação, visualização científica, realidade virtual e aumentada, entre outros.

A arquitetura de sistemas gráficos geralmente inclui os seguintes componentes:

- **Dispositivos de entrada**: teclado, mouse, joystick, dispositivos de rastreamento de movimento e câmeras que permitem ao usuário interagir com o sistema.

- **Unidade de processamento gráfico (GPU)**: é o componente principal que realiza as operações matemáticas necessárias para gerar imagens em tempo real. A GPU é otimizada para processamento paralelo e é capaz de executar muitos cálculos simultaneamente.

- **APIs gráficas**: são interfaces de programação de aplicativos que permitem que os desenvolvedores interajam com a GPU para criar imagens em tempo real. Algumas das APIs gráficas mais populares incluem OpenGL, DirectX e Vulkan.

- **Bibliotecas de software**: como bibliotecas de física, de animação e de inteligência artificial, que ajudam a criar uma experiência mais imersiva para o usuário. Observe alguns exemplos a seguir:

  – **NumPy**: uma biblioteca para Python que suporta matrizes e operações matemáticas de alto nível.

  – **TensorFlow**: uma biblioteca de aprendizado de máquina de código aberto desenvolvida pela Google que permite que os desenvolvedores criem modelos de aprendizado de máquina.

  – **jQuery**: uma biblioteca JavaScript popular que simplifica a manipulação do DOM e a criação de animações e efeitos.

- **Ruby on Rails**: uma estrutura de aplicativo web que é baseada na linguagem de programação Ruby e inclui muitas bibliotecas úteis.
- **Bootstrap**: uma biblioteca de componentes de interface do usuário que inclui estilos CSS predefinidos e componentes interativos.
- **Pandas**: uma biblioteca para Python que suporta análise de dados e manipulação de dados em séries temporais e estruturas de dados.
- **Pygame**: uma biblioteca de jogos para Python que inclui recursos para criação de jogos em 2D.
- **RubyGems**: um gerenciador de pacotes para a linguagem de programação Ruby que permite que os desenvolvedores instalem e gerenciem bibliotecas de software.

- **Display e saída**: inclui monitores, projetores e dispositivos de realidade virtual e aumentada que exibem as imagens geradas pelo sistema.

A arquitetura de sistemas gráficos pode ser complexa e requer conhecimento especializado em matemática, programação, engenharia de software e design gráfico para ser projetada e implementada com sucesso.

1.3 Primitivas como elementos básicos do desenho

Em computação gráfica, primitivas são os elementos básicos usados para construir imagens e objetos. Essas primitivas são formas geométricas simples, como pontos, linhas, polígonos e curvas, que podem ser manipulados para criar formas mais complexas. Esses elementos básicos são usados em diversos sistemas gráficos e em muitas aplicações, incluindo jogos, animação, design gráfico e visualização científica.

As primitivas são definidas matematicamente, usando coordenadas e equações que descrevem suas formas. Por exemplo, um ponto é definido por sua posição em um sistema de coordenadas, enquanto uma linha é definida por dois pontos. Polígonos são formados por uma sequência de vértices conectados por linhas, e curvas são definidas por equações matemáticas que descrevem sua forma.

Uma vez que as primitivas são definidas, elas podem ser transformadas, rotacionadas, escaladas e combinadas para criar objetos mais complexos. Isso permite que os desenvolvedores criem modelos 3D, animações, gráficos vetoriais e outros tipos de conteúdo visual.

As primitivas são uma parte fundamental da computação gráfica e são usadas em quase todos os sistemas gráficos. Entender como elas funcionam é essencial para qualquer pessoa que queira criar conteúdo visual usando computadores.

As primitivas gráficas são os elementos básicos usados para criar imagens em sistemas gráficos. Essas primitivas são formas geométricas simples que podem ser manipuladas e combinadas para criar imagens mais complexas. As primitivas gráficas mais comuns serão destacadas a seguir.

Ponto

É uma primitiva gráfica que não tem dimensão e é definida por um par de coordenadas (x, y). A representação matemática de um ponto é (x, y).

Um ponto é a primitiva gráfica mais simples, sendo representado por uma única coordenada no espaço 2D ou 3D. A representação matemática de um ponto é dada por:

P = (x, y)

Onde x e y são as coordenadas do ponto na horizontal e vertical, respectivamente.

A figura a seguir mostra um ponto de cor vermelha em um espaço 2D representado na coordenada (3, 2).

Os pontos são primitivas gráficas muito utilizados em sistemas gráficos, como em softwares de desenho vetorial e de modelagem 3D. Eles são frequentemente usados para marcar locais importantes em uma imagem ou para representar objetos simples, como partículas em um sistema de partículas.

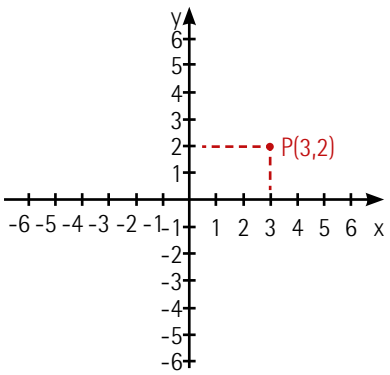


Figura 2

Linha

É um elemento gráfico definido por dois pontos, também conhecido como segmento de reta. Uma linha é uma primitiva gráfica definida por dois pontos (x_1, y_1) e (x_2, y_2). A representação matemática de uma linha é dada pela equação y = mx + b, onde m é a inclinação da linha e b é o intercepto y.

Uma linha é uma primitiva gráfica usada para conectar dois pontos em um espaço 2D ou 3D. É definida pelos pontos inicial e final, que determinam a sua posição e comprimento. A representação matemática de uma linha é dada por:

y = mx + b

Onde m é a inclinação da linha e b é o ponto onde a linha cruza o eixo y. A inclinação m é dada pela diferença entre as coordenadas y dos pontos inicial e final, dividida pela diferença entre as coordenadas x dos mesmos pontos.

A figura a seguir mostra uma linha em um espaço 2D conectando os pontos P1(3,2) e P2(5,6).

As linhas são primitivas gráficas fundamentais em sistemas gráficos, sendo usadas em praticamente todas as aplicações gráficas. Elas são comumente aplicadas para representar bordas ou contornos de objetos, ou para desenhar diagramas ou gráficos.

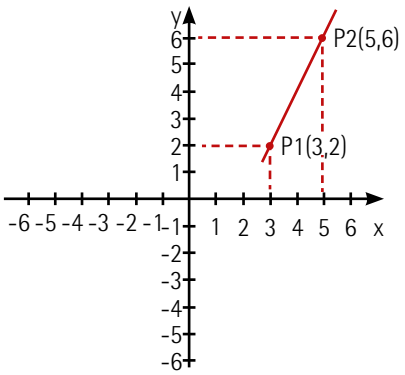


Figura 3

Polígono

É uma forma geométrica fechada e plana definida por uma sequência de pontos conectados por linhas retas. Um polígono com n vértices é chamado de n-gon. A representação matemática de um polígono é dada por uma lista de pares ordenados que indicam seus vértices, ou seja, $P = [(x_1, y_1), (x_2, y_2),..., (x_n, y_n)]$.

Um polígono é uma primitiva gráfica usada para representar formas geométricas fechadas com três ou mais lados em um espaço 2D ou 3D. É definido por uma sequência de pontos de controle que determinam as coordenadas dos vértices do polígono. A representação matemática de um polígono pode ser obtida a partir da equação da linha que conecta cada par de pontos de controle.

A figura a seguir ilustra um exemplo de polígono, um triângulo, definido pelos pontos de controle P1, P2 e P3.

Os polígonos são primitivas gráficas muito utilizadas em sistemas gráficos, especialmente em softwares de modelagem 3D e jogos. Eles são frequentemente usados para representar formas geométricas complexas, como personagens, veículos, edifícios, paisagens e outros objetos.

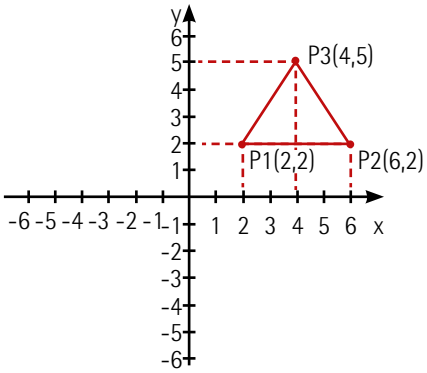


Figura 4

Retângulo

É um polígono com quatro lados, e seus lados opostos são paralelos e iguais. Um retângulo é definido por suas coordenadas (x, y), sua largura e altura. Sua representação matemática é R = (x, y, w, h), onde w é a largura e h é a altura do retângulo. Ele é uma primitiva gráfica usada para representar formas retangulares em um espaço 2D ou 3D. Os pontos de controle dos vértices opostos do retângulo determinam as suas dimensões e sua posição.

Os retângulos são muito utilizados em sistemas gráficos, especialmente em softwares de desenho vetorial, diagramação e interfaces gráficas de usuário, por exemplo, para representar botões, caixas de texto, barras de rolagem, imagens e outros elementos gráficos comuns em interfaces de usuário.

A figura a seguir destaca um retângulo com coordenadas (2, 2), largura 4 e altura 3.

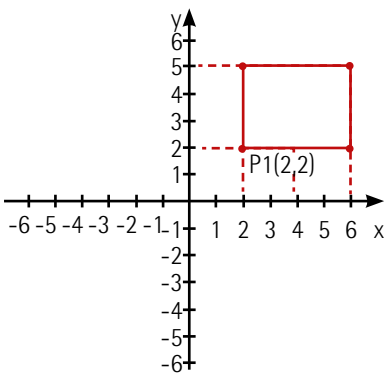


Figura 5

Círculo

É uma forma geométrica definida por um ponto central (x_c, y_c) e um raio r. A representação matemática de um círculo é $(x - x_c)^2 + (y - y_c)^2 = r^2$. O círculo também é uma primitiva gráfica usada para representar formas circulares em um espaço 2D ou 3D, e a coordenada do centro e o tamanho do raio determinarão sua posição e sua dimensão (tamanho).

O círculo é muito utilizado em sistemas gráficos, sobretudo em softwares de desenho vetorial, diagramação e interfaces gráficas de usuário, por exemplo, para representar botões, ícones, elementos de logotipos e outros elementos gráficos que requerem formas circulares.

A figura a seguir ilustra um exemplo de círculo definido pelo centro (2, 2) e raio 1.

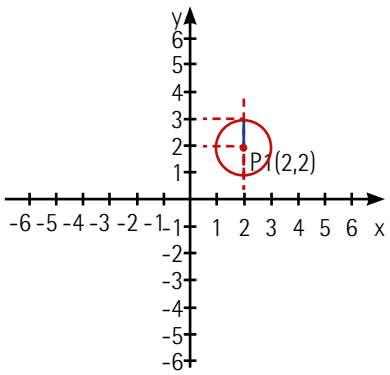


Figura 6

Elipse

É uma forma geométrica que se assemelha a um círculo achatado ou esticado. Os elementos principais são os focos F1 e F2, o eixo maior e o eixo menor. A equação matemática que representa uma elipse é dada por:

$$\frac{(x - x_c)^2}{a^2} + \frac{(y - y_c)^2}{b^2} = 1$$

Onde (x_c, y_c) é o centro da elipse, a é a distância do centro até um dos focos e b é a metade da dimensão do eixo menor.

As elipses são primitivas gráficas muito utilizadas em sistemas gráficos, como em softwares de desenho vetorial e de modelagem 3D. Por exemplo, em um software de desenho vetorial, o usuário pode criar uma elipse e ajustar seus parâmetros para criar formas curvas e arredondadas. Já em um software de modelagem 3D, a elipse pode ser usada como uma forma básica para modelar objetos mais complexos.

A figura a seguir mostra as representações gráficas de elipses, quando o eixo maior é paralelo ao eixo y e quando o eixo maior é paralelo ao eixo x.

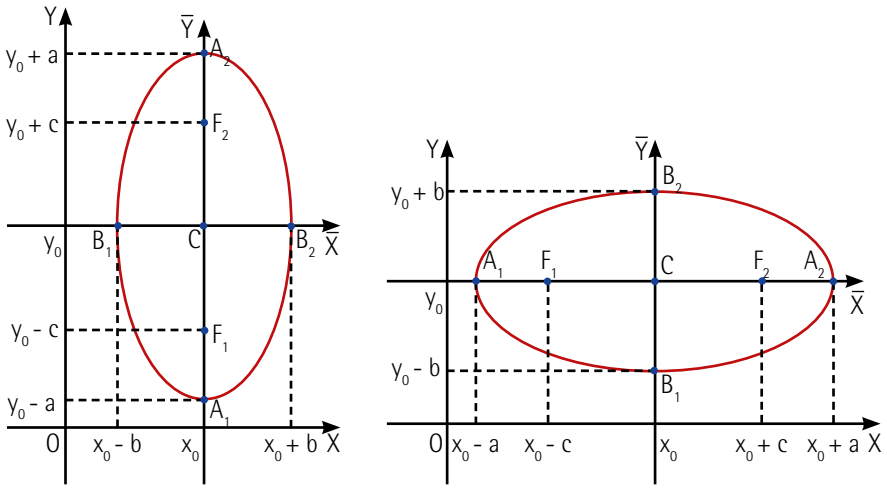


Figura 7

Adaptada de: Delgado, Frensel e Crissa (2012).

Observação

A elipse é obtida fazendo-se um corte inclinado em um cone.

Curva de Bézier

É uma curva definida por um conjunto de pontos de controle que determinam sua forma. Uma curva de Bézier é uma primitiva gráfica usada para criar curvas suaves em um plano 2D ou em um espaço 3D. Trata-se de uma função paramétrica que descreve a posição do ponto da curva para cada valor do parâmetro t. A representação matemática de uma curva de Bézier é dada por:

$$C(t) = \sum_{i=0}^{N} P_i B_i^n(t)$$

Onde são polinômios de Bernstein e são os pontos de controle previamente selecionados

As figuras a seguir ilustram duas curvas de Bézier, de grau 2 e grau 3, respectivamente, concebidas através de uma construção recursiva.

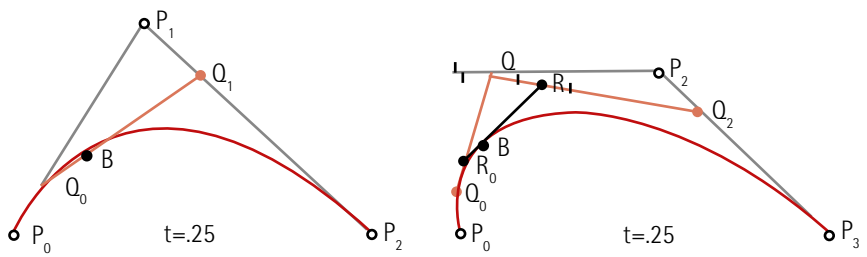


Figura 8

Adaptada de: Machado (2013).

As curvas de Bézier são primitivas gráficas muito utilizadas em sistemas gráficos, como jogos, animação, visualização científica, gráficos vetoriais e em outras aplicações que envolvam a criação e exibição de imagens. Elas permitem que o usuário crie curvas suaves e precisas com facilidade, dando mais flexibilidade e controle ao processo de criação.

Combinando todas essas primitivas, os desenvolvedores podem criar imagens mais complexas e objetos 3D, tornando possível a criação de um conteúdo visual impressionante.

Saiba mais

Para saber mais sobre a importância das curvas de Bézier, acesse:

MACHADO, F. C. *Primeiro projeto de análise numérica II*. Curvas de Bézier e desenho de fontes tipográficas. Campinas: Unicamp, 2013. Disponível em: https://bit.ly/3GhE8eA. Acesso em: 30 mar. 2023.

1.4 Primitivas com funções de linguagem

As primitivas gráficas podem ser vistas como funções de linguagem em sistemas gráficos. Em geral, elas são implementadas como sub-rotinas ou procedimentos que recebem como entrada os parâmetros necessários para desenhar a primitiva e produzem como saída a representação gráfica da primitiva na tela ou no dispositivo de saída. Por exemplo, a primitiva gráfica linha pode ser implementada como uma função que recebe como parâmetros as coordenadas dos pontos inicial e final da linha e a cor da linha a ser desenhada. Essa função calcula os pixels correspondentes à linha usando algoritmos de rasterização e desenha a linha na tela ou no dispositivo de saída usando a cor especificada.

Um pixel é a menor unidade de uma imagem digital. É uma abreviação para elemento de imagem (picture element, em inglês), e é comumente usado para medir a resolução de uma imagem digital. Um pixel é uma pequena unidade quadrada que pode conter uma cor específica. Em uma imagem digital colorida, um pixel é geralmente composto de três subpixels, cada um dos quais representa uma das cores primárias (vermelho, verde e azul) em diferentes intensidades. Quando combinados, esses subpixels criam uma ampla gama de cores visíveis. A resolução de uma imagem digital é medida pelo número de pixels em uma determinada área: quanto maior a quantidade de pixels, haverá mais qualidade e nitidez em uma imagem.

A proporção de pixels (Pixel Aspect Ratio (PAR)) pode variar de acordo com a situação. PAR é a relação de aspecto entre a largura e a altura de um pixel em um sistema de vídeo digital. Isso é importante porque, embora um pixel seja normalmente considerado um quadrado perfeito, nem sempre é assim em todas as situações.

Em alguns sistemas de vídeo digital, como o antigo padrão de televisão analógica NTSC, os pixels são retangulares em vez de quadrados, o que significa que a relação entre a largura e a altura do pixel

é diferente de 1:1. Em tais casos, o pixel aspect ratio é usado para corrigir a aparência da imagem, garantindo que ela seja exibida com as proporções corretas.

Por exemplo, se um vídeo tem um PAR de 1:2, isso significa que cada pixel é duas vezes mais alto do que é largo. Quando exibida em um dispositivo com um PAR de 1:1, a imagem pode parecer esticada ou comprimida horizontalmente, então o software de reprodução de vídeo pode ajustar o tamanho dos pixels para corrigir a aparência da imagem. A figura a seguir ilustra um exemplo de um PAR de 2:1.

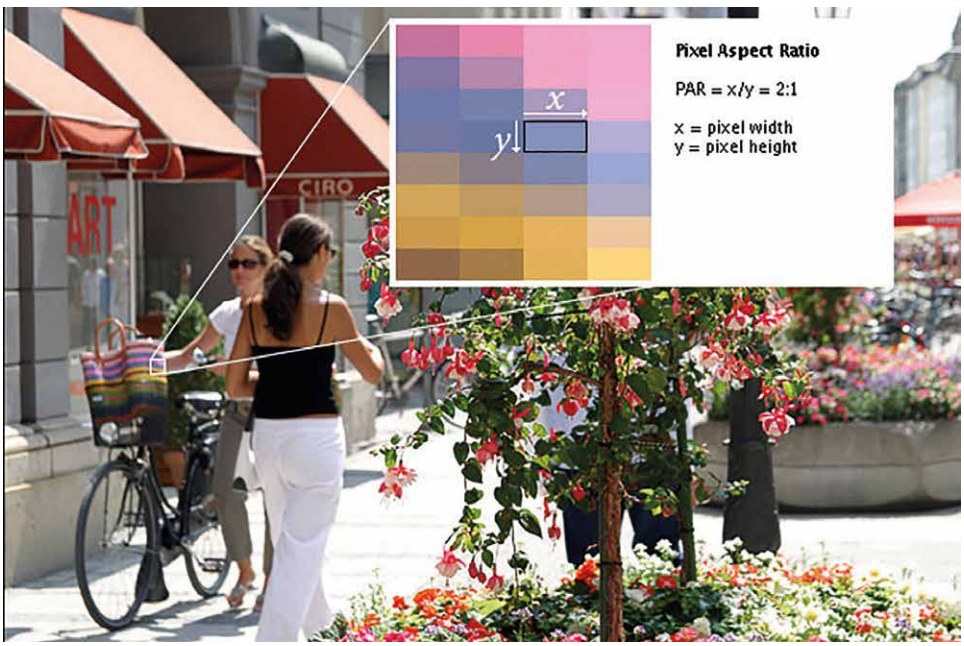


Figura 9

Disponível em: https://bit.ly/3IOZVU0. Acesso em: 30 mar. 2023.

As primitivas gráficas são algumas das principais ferramentas utilizadas na computação gráfica para desenhar objetos e imagens na tela ou no dispositivo de saída. Elas são a base para a criação de algoritmos mais complexos, que permitem criar formas mais elaboradas e efeitos visuais.

As primitivas gráficas definidas por funções de linguagem são aquelas que utilizam comandos ou instruções para criar imagens. Algumas dessas primitivas incluem:

- **Texto**: é uma primitiva gráfica que permite escrever palavras e frases na tela. A representação matemática do texto é dada pela posição do cursor e pelos caracteres que são desenhados na tela. A figura a seguir, uma tela da aplicação Paint, do sistema operacional Windows, mostra um exemplo de texto na tela com a escrita "COMPUTAÇÃO GRÁFICA". Note que no canto inferior esquerdo há a referência da posição do cursor (296,177px).

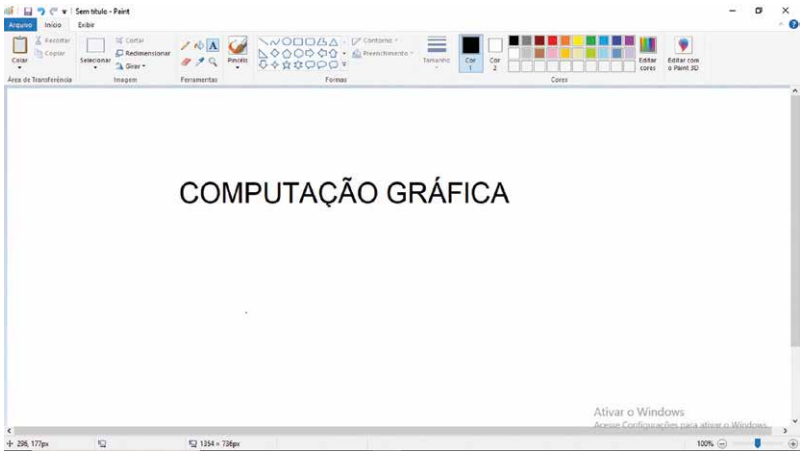


Figura 10

- **Preenchimento de cor**: é uma primitiva gráfica que permite preencher uma área com uma cor sólida. A representação matemática do preenchimento de cor é dada pela posição do ponto de partida e pela cor que é usada para preencher a área. A figura a seguir ilustra um exemplo de preenchimento de cor.

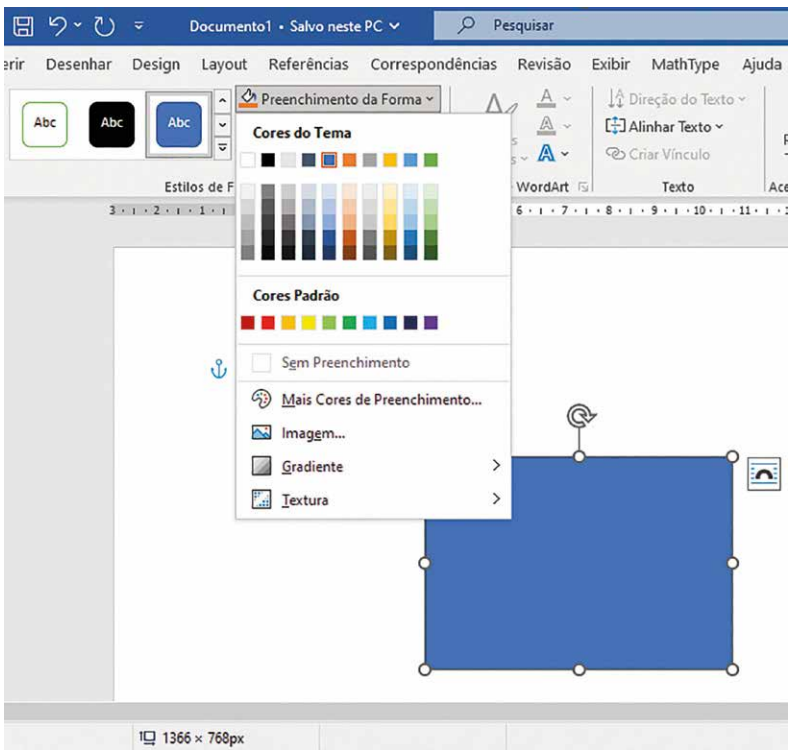
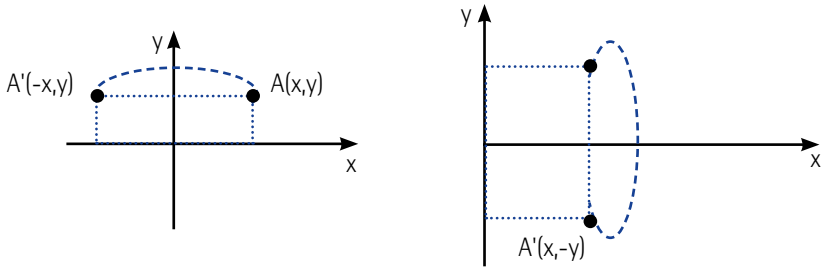


Figura 11

- **Transformações geométricas**: são primitivas gráficas que permitem mover, rotacionar, redimensionar e espelhar objetos na tela. A representação matemática dessas transformações é dada por matrizes de transformação. A figura seguinte destaca a transformação geométrica da reflexão em torno do eixo y e do eixo x de uma imagem.



Reflexão em torno do y

Reflexão em torno do x

Figura 12

- **Gradientes**: são primitivas gráficas que permitem criar transições suaves entre duas ou mais cores. A representação matemática dos gradientes é dada pela posição do ponto de partida e do ponto de término, além das cores que são usadas na transição. A figura a seguir acentua um exemplo de gradiente na tela.

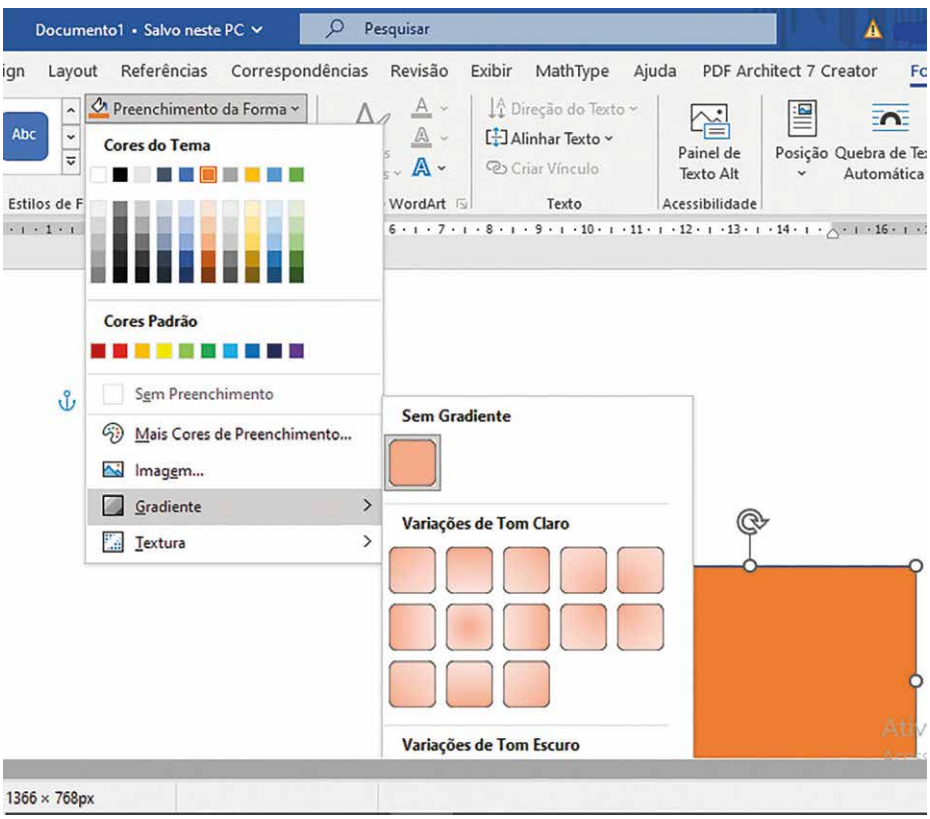


Figura 13

Esses são apenas alguns exemplos de primitivas gráficas com funções de linguagem. Cada uma dessas primitivas possui um modelo matemático específico que define como elas funcionam e como podem ser utilizadas para criar imagens na tela.

1.5 Pacotes gráficos e bibliotecas principais

Em computação gráfica, os pacotes gráficos são conjuntos de ferramentas de software para criação, edição e renderização de imagens digitais. Eles geralmente incluem uma ampla variedade de recursos, como modelagem 3D, animação, texturização, iluminação e efeitos especiais.

Algumas das principais opções de pacotes gráficos incluem:

- Autodesk Maya
- Autodesk 3ds Max
- Blender
- Cinema 4D
- Houdini
- Unity

Por outro lado, as bibliotecas principais são conjuntos de código pré-escrito que fornecem funcionalidades específicas para a criação de gráficos em um determinado ambiente de programação. Essas bibliotecas podem ser usadas para criar aplicativos de jogos, visualização de dados, interfaces de usuário e muito mais.

Algumas das principais bibliotecas gráficas incluem:

- OpenGL (para gráficos 3D).
- DirectX (para jogos em plataformas Windows).
- Vulkan (para jogos em plataformas multiplataforma).
- WebGL (para gráficos 3D em navegadores da web).
- Cairo (para gráficos 2D em várias plataformas).
- OpenCV (para processamento de imagens em tempo real).

As bibliotecas e pacotes gráficos são uma parte crucial do desenvolvimento de jogos, animações, visualizações e outras aplicações gráficas em computadores e dispositivos móveis. A escolha de qual pacote ou biblioteca usar depende das necessidades específicas do projeto e das habilidades do desenvolvedor.

Existem várias bibliotecas principais em computação gráfica, cada uma com suas próprias características e funcionalidades específicas. A seguir são acentuadas algumas das principais:

- **OpenGL**: uma biblioteca de gráficos 3D de plataforma cruzada que é amplamente usada em jogos, visualização de dados, CAD e outras aplicações.

- **DirectX**: uma coleção de APIs (interfaces de programação de aplicativos) que são usadas principalmente para desenvolver jogos e aplicativos multimídia em plataformas Windows.

- **Vulkan**: uma biblioteca de gráficos 3D de baixo nível que foi projetada para aproveitar ao máximo o desempenho dos processadores gráficos modernos. É uma alternativa de baixo nível para o OpenGL.

- **OpenCV**: uma biblioteca de código aberto que fornece ferramentas para processamento de imagens em tempo real, visão computacional e aprendizado de máquina.

- **Cairo**: uma biblioteca de gráficos 2D que é usada para renderizar imagens e textos em várias plataformas.

- **SFML**: uma biblioteca de gráficos 2D que é usada para desenvolver jogos e outras aplicações multimídia em C++.

- **SDL**: uma biblioteca de gráficos e entrada de usuário multiplataforma que é usada principalmente para desenvolver jogos em C/C++.

Essas são apenas algumas das principais bibliotecas em computação gráfica. A escolha da biblioteca certa depende das necessidades específicas do projeto e das habilidades do desenvolvedor.

As bibliotecas gráficas geralmente fornecem funções e métodos para criar e manipular objetos gráficos, como retângulos, círculos e polígonos, bem como para desenhar e renderizar imagens, texto e outros elementos gráficos. As bibliotecas também podem fornecer funcionalidades avançadas, como suporte para animação, sombras, reflexos e iluminação em tempo real.

Elas funcionam como um conjunto de ferramentas que o desenvolvedor pode usar para criar a aparência visual e interativa de um aplicativo. Por exemplo, se um desenvolvedor quiser criar um jogo 2D, ele poderá usar uma biblioteca gráfica como a SFML para criar os gráficos do jogo, lidar com entrada do usuário, gerenciar sprites e animações, entre outras funcionalidades.

A biblioteca gráfica é geralmente usada com uma linguagem de programação, como C++, Java ou Python, e é integrada no código do aplicativo por meio da inclusão de cabeçalhos de bibliotecas e a chamada de funções fornecidas pela biblioteca em diferentes partes do código.

Elas fornecem aos desenvolvedores uma maneira mais fácil e rápida de criar gráficos complexos e visualmente atraentes em seus aplicativos, sem precisar escrever todo o código do zero.

2 PRIMITIVAS GRÁFICAS EM DUAS DIMENSÕES

Em computação gráfica, elas são formas geométricas básicas que podem ser usadas para criar imagens e gráficos em uma tela ou superfície. As mais comuns em duas dimensões são:

- **Pontos**: usada para representar um único pixel na tela. Os pontos são usados para desenhar linhas, curvas e outras formas.

- **Linhas**: aplicada para conectar dois pontos. As linhas podem ser usadas para criar desenhos simples, diagramas e gráficos.

- **Retângulos**: utilizada para criar formas retangulares. Os retângulos podem ser preenchidos ou vazios e podem ser usados para criar botões, menus e outros elementos de interface do usuário.

- **Elipses**: empregada para criar formas ovais. As elipses podem ser preenchidas ou vazias e podem ser usadas para criar gráficos e diagramas.

- **Polígonos**: aplicada para criar formas com muitos lados. Os polígonos podem ser preenchidos ou vazios e podem ser usados para criar formas complexas, como diagramas de fluxo.

As primitivas gráficas em duas dimensões são usadas para criar imagens e gráficos simples, mas também são a base para a criação de gráficos mais complexos, como diagramas, gráficos e jogos em 2D. Elas podem ser manipuladas por meio de funções e métodos em bibliotecas gráficas para criar desenhos e gráficos visualmente atraentes.

2.1 Pontos, vetores e matrizes em CG

Em computação gráfica, pontos, vetores e matrizes são conceitos importantes que são usados para representar objetos e operações geométricas em um espaço tridimensional. A seguir, destacam-se as definições de cada um desses conceitos:

- **Ponto**: é uma representação abstrata de uma posição no espaço tridimensional. Ele é definido por um conjunto de coordenadas (x, y, z) que indicam sua posição relativa a um sistema de coordenadas. Na computação gráfica, os pontos são frequentemente usados para representar vértices de objetos e para descrever a posição da câmera e do observador.

- **Vetor**: um vetor é uma entidade matemática que representa uma magnitude e uma direção no espaço tridimensional. Ele é definido por um conjunto de coordenadas (x, y, z) que indicam sua direção e seu comprimento. Os vetores são comumente usados na computação gráfica para representar movimentos, rotações e escalas de objetos.

- **Matriz**: é uma estrutura de dados bidimensional que armazena valores numéricos organizados em linhas e colunas. Na computação gráfica, as matrizes são normalmente usadas para realizar operações lineares em objetos geométricos, como transformações de rotação e escala e translação.

As matrizes são especialmente úteis na computação gráfica, pois elas podem ser compostas para realizar operações complexas.

Vamos considerar um vetor V para o ponto (x,y), tendo uma direção, um sentido e um comprimento específico. Podemos utilizar a fórmula a seguir para calcular o seu comprimento:

$$|V|=\sqrt{x^2+y^2}$$

Então, se pensarmos em um vetor 3D, cuja definição de origem é o ponto (x,y,z), usa-se a fórmula a seguir para calcular seu comprimento:

$$|V|=\sqrt{x^2+y^2+z^2}$$

Quando trabalhamos com computação gráfica, utilizamos frequentemente matrizes. O primeiro passo é definir quantos elementos existem em cada direção, por exemplo: a seguir podemos observar uma matriz 5x5.

1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

Em resumo, pontos, vetores e matrizes são conceitos fundamentais na computação gráfica que permitem representar objetos e operações geométricas em um espaço tridimensional. Eles são normalmente usados com bibliotecas gráficas para criar gráficos e animações visualmente atraentes e interativas.

Para definir e modelar os objetos sintéticos que representaremos nas imagens, utilizamos a representação vetorial. Eduardo Azevedo e Aura Conci (2003b, p. 15) exemplificaram a representação vetorial:

> Na representação vetorial das imagens, são usados como elementos básicos os pontos, as linhas, as curvas, as superfícies tridimensionais ou mesmo os sólidos que descrevem os elementos, que formam as imagens sinteticamente no computador. Esses elementos são denominados primitivas vetoriais da imagem. As primitivas vetoriais são associadas a um conjunto de atributos que define sua aparência e a um conjunto de dados que define sua geometria (pontos de controle). Para esclarecer melhor, vamos considerar alguns exemplos. Dois elementos facilmente caracterizados como vetoriais, pela noção de vetores já discutida, são os pontos e as linhas retas. A cada elemento de um conjunto de pontos associa-se uma posição, que pode ser representada por suas coordenadas (geometria), e uma cor, que será como

esses pontos aparecerão na tela (atributos). No caso de um conjunto de linhas retas, cada uma pode ser definida pelas coordenadas de seus pontos extremos (geometria) e sua cor, espessura, ou ainda se aparecerá pontilhada ou tracejada (atributos)

A representação matricial é uma forma de indicar imagens digitais em uma matriz numérica de pixels. Cada pixel é designado por um conjunto de valores numéricos que determinam sua cor e intensidade.

A matriz de pixels é geralmente organizada em linhas e colunas, em que cada elemento da matriz representa um pixel específico na imagem. Cada pixel pode ser indicado por uma combinação de valores RGB (vermelho, verde e azul) ou outros sistemas de cores, como o sistema CMYK (ciano, magenta, amarelo e preto) utilizado na impressão.

A representação matricial é usada em diversos algoritmos de computação gráfica, como processamento de imagem, reconhecimento de padrões, filtragem de imagens e renderização de objetos tridimensionais. Ela permite que as imagens sejam manipuladas de forma eficiente em nível de pixel, o que é essencial para muitas aplicações de gráficos por computador. Além disso, a representação matricial é aplicada em outros contextos da computação gráfica, como a modelagem de objetos em 3D, em que as malhas tridimensionais são indicadas por matrizes numéricas que descrevem suas coordenadas e propriedades.

A figura a seguir exemplifica a forma de descrição de imagem matricial. É utilizada para formar a imagem na memória, nas telas dos computadores e nos dispositivos gráficos de saída (vídeos e impressoras).

0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	1	1	1	0	0	0	0	0
0	0	0	1	1	1	1	1	0	0	0	0
0	0	1	1	1	1	1	1	1	0	0	0
0	0	1	0	0	0	0	0	1	0	0	0
0	0	1	0	0	0	0	0	1	0	0	0
0	0	1	0	0	0	0	0	1	0	0	0
0	0	1	0	0	0	0	0	1	0	0	0
0	0	1	1	1	1	1	1	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

Memória de imagem

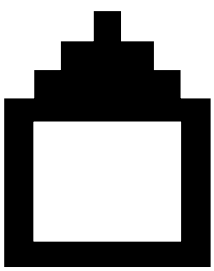


Imagem na tela

Figura 14 – Descrição de imagens matriciais por conjunto de pixels

Fonte: Azevedo e Conci (2003a, p. 15).

O modelo matemático em computação gráfica é um conjunto de equações e algoritmos que descrevem como os objetos são representados e manipulados em um espaço tridimensional. Essas equações e algoritmos são implementados em software de computador para criar imagens e animações em tempo real.

Algumas das principais equações e algoritmos utilizados no modelo matemático em computação gráfica incluem:

- **Transformações geométricas**: essas equações permitem que os objetos sejam transformados em um espaço tridimensional, incluindo rotações, translações e escalas.
- **Algoritmos de rasterização**: convertem objetos geométricos em pixels na tela. Eles são usados para criar imagens realistas em tempo real.
- **Algoritmos de iluminação**: calculam como a luz interage com os objetos na cena. Eles são usados para criar sombras, reflexões e outras propriedades visuais realistas.
- **Algoritmos de sombreamento**: calculam a cor de cada pixel na imagem com base na iluminação e nas propriedades do material do objeto.
- **Algoritmos de mapeamento de texturas**: permitem que as texturas sejam aplicadas aos objetos na cena para criar superfícies mais realistas.

Alguns exemplos de recursos em computação gráfica que são criados a partir do modelo matemático incluem:

- **Jogos**: jogos em computador usam o modelo matemático para criar ambientes 3D, personagens e animações.
- **Filmes de animação**: utilizam o modelo matemático para criar cenas e personagens em 3D.
- **Visualização científica**: aplica o modelo matemático para representar dados complexos em um espaço tridimensional.
- **Design de produto**: emprega o modelo matemático para criar modelos 3D de produtos, permitindo que eles sejam visualizados em diferentes ângulos e iluminações antes de serem produzidos fisicamente.

O modelo matemático é uma parte fundamental da computação gráfica e é usado para criar uma ampla variedade de recursos, desde jogos e filmes até visualização científica e design de produto. Ele usa uma variedade de equações e algoritmos para representar objetos e operações geométricas em um espaço tridimensional e é implementado em software de computador para criar imagens e animações em tempo real.

2.2 Sistemas de referência (universo, objeto, dispositivo)

Servem para definir um espaço tridimensional em que objetos são representados e manipulados. Geralmente, eles são compostos de um conjunto de eixos e um ponto de origem que, juntos, definem um espaço tridimensional. São eles:

- **Sistema de coordenadas cartesianas**: é um sistema de referência formado por dois ou três eixos perpendiculares entre si (2D e 3D, respectivamente) e um ponto de origem 0. Cada eixo é composto de valores numéricos que indicam a posição relativa de um objeto no espaço bi ou tridimensional.

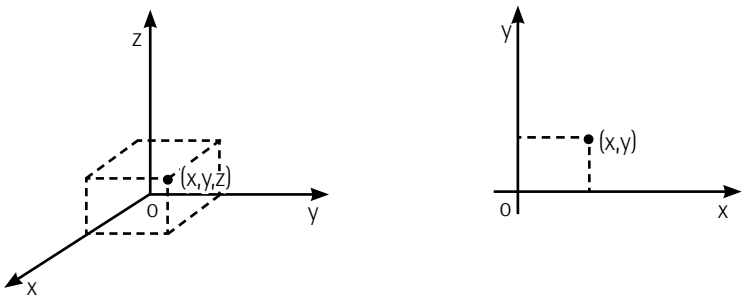


Figura 15 – Sistema de coordenadas cartesianas em 3D (x,y,z) e em 2D (x,y)

- **Sistema de coordenadas cilíndricas**: é um sistema de referência que combina um eixo vertical com um sistema de coordenadas polares. O eixo vertical representa a altura do objeto, enquanto o sistema de coordenadas polares descreve a posição do objeto em relação a um ponto central.

- **Sistema de coordenadas esféricas**: é um sistema de referência que usa um ponto central e duas coordenadas angulares para descrever a posição de um objeto. O ângulo theta indica a posição do objeto no plano horizontal, enquanto o ângulo phi expressa a posição do objeto no plano vertical.

- **Sistema de coordenadas polares**: neste sistema bidimensional, cada ponto no plano é descrito por um ângulo e uma distância radial em relação a um ponto de referência chamado de polo. O polo é geralmente representado por um ponto central na origem do sistema de coordenadas.

A figura a seguir mostra três sistemas de coordenadas.

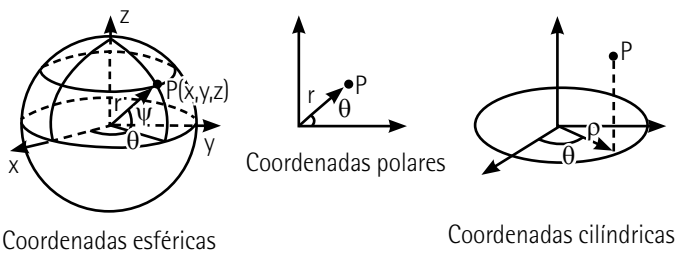


Figura 16

Fonte: Azevedo e Conci (2003a, p. 36).

Em computação gráfica, para a criação de imagens, devemos considerar quatro tipos de sistemas de referência principais, os quais são usados para definir a posição e a orientação dos objetos em relação ao espaço tridimensional. São eles:

- **Sistema de Referência Universo (SRU)**: é um sistema global que é utilizado para definir a posição dos objetos em relação a um ponto fixo no espaço, geralmente o ponto de origem do sistema de coordenadas cartesianas. Ele é importante para definir a posição inicial dos objetos em uma cena.

- **Sistema de Referência Objeto (SRO)**: é um sistema local que é usado para definir a posição e a orientação de um objeto em relação ao SRU. Cada objeto pode ter seu próprio sistema de referência objeto, que é usado para realizar transformações geométricas em relação a outros objetos na cena.

- **Sistema de Referência Dispositivo (SRD)**: é empregado para representar a saída gráfica, ou seja, a imagem final que será exibida na tela do dispositivo.

- **Sistema de Referência Normalizado (SRN)**: trabalha com as coordenadas normalizadas, isto é, com valores entre 0 e 1, onde $0 \leq x \leq 1$ e $0 \leq y \leq 1$, sendo x e y as coordenadas horizontais e verticais. O SRN serve como um sistema de referência intermediário.

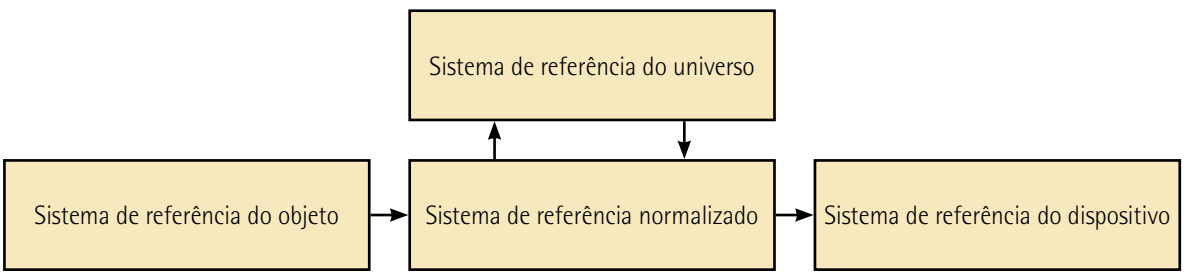


Figura 17

Fonte: Azevedo e Conci (2003a, p. 37).

Os sistemas de referência SRU, SRO e SRD são usados em computação gráfica para realizar operações geométricas, como rotações, translações e escalas, em objetos no espaço tridimensional. Essas operações são realizadas com base nas coordenadas dos objetos em relação ao sistema de referência.

Além disso, os sistemas de referência são usados para definir a posição da câmera e do observador na cena. Isso permite que o software de computador crie imagens realistas em tempo real, mostrando a cena a partir do ponto de vista do observador.

O SRU é usado para definir a posição inicial dos objetos em uma cena. Já o SRO é aplicado para realizar transformações geométricas em relação a outros objetos na cena. Por sua vez, o SRD é utilizado para mapear as coordenadas da cena em coordenadas do dispositivo para a rasterização e renderização

final da imagem. Todos eles são fundamentais para a construção e manipulação de objetos em uma cena de computação gráfica.

O modelo matemático empregado em computação gráfica é baseado em geometria analítica, álgebra linear, cálculo e outras áreas da matemática. Ele é usado para descrever a posição, a orientação, a forma, o tamanho e a cor dos objetos em uma cena de computação gráfica.

2.3 Mapeamento window to viewport

Devemos ter em mente que, em computação gráfica, a primitiva ponto, ente matemático, sem dimensão, transforma-se em pixel menor unidade gráfica manipulável (pixel: de picture element).

Ao contrário do ponto, o pixel tem forma, dimensão, mas esses atributos não são, em geral, manipuláveis diretamente através de funções, dependem do hardware.

Por outro lado, tanto o ponto quanto o pixel são objetos de um desenho que se inscreve em um espaço bidimensional. No caso do ponto matemático, tal espaço pertence ao R^2. Já o pixel é um elemento de uma matriz de tamanho W x H (Width – largura; Height – altura, em pixels, da matriz do dispositivo gráfico de saída), cujas coordenadas pertencem ao N^2. Destaca-se que R é o conjunto dos números reais e N é o conjunto dos naturais. Assim, podemos atribuir ao pixel apenas duas propriedades fundamentais: cor e posição.

Como as coordenadas do Ponto, que formam um vetor com valores no espaço real, e as coordenadas do Pixel são elementos de uma matriz, possuindo coordenadas inteiras e positivas, devemos converter os valores adequadamente.

Esse cálculo é denominado mapeamento window-to-viewport. Podemos chamá-lo de rasterização do ponto, já que o termo rasterização (rastering) é o processo de converter vetores para pixels em dispositivos raster (lembre-se de que as coordenadas de um ponto são uma grandeza vetorial).

Sejam XR e YR as coordenadas (reais) de um ponto pertencente ao R^2 e XP e YP (o P refere-se a Pixel) as coordenadas do pixel correspondente (inteiros positivos) na matriz gráfica (tela ou canvas).

As coordenadas reais (XR, YR) são, muitas vezes, referidas como coordenadas do universo (ou do mundo) e estão descritas no SRU.

Por outro lado, as coordenadas inteiras (XP, YP) são descritas no sistema de referência do dispositivo (SRD). Quando elas representam diretamente os valores das linhas e colunas no dispositivo gráfico, portanto valores inteiros que correspondem aos índices dos elementos da matriz de pixel, serão referidas como coordenadas do dispositivo. Caso essas coordenadas expressem os valores reais de seus respectivos pontos no SRU, serão referidas como coordenadas lógicas.

Uma vez que o espaço disponível para desenho no dispositivo gráfico de saída é limitado pela matriz de pixels, devemos especificar os valores mínimos e máximos de uma janela de visualização, denominada viewport, na qual visualizaremos o nosso modelo matemático (desenho, função etc.)

Portanto, a viewport corresponde a uma área no interior da matriz de pixels do dispositivo de saída gráfica. Ela pode corresponder a toda a matriz, como pode haver várias viewports no dispositivo, sobrepostas ou não.

Já o plano cartesiano onde localizamos as coordenadas reais (XR, YR) é infinito. Para representarmos qualquer modelo matemático desse espaço no interior da viewport, devemos definir um domínio que contenha os objetos a serem representados. Esse domínio será chamado de window. A window tem o mesmo papel de uma janela no sentido comum, através da qual vemos uma cena que queremos representar. Cada elemento primitivo descrito no interior da window será, então, mapeado na viewport.

Como vimos, os objetos geométricos são descritos (modelados) no SRO. Veja o esquema de mapeamento mostrado na figura a seguir.

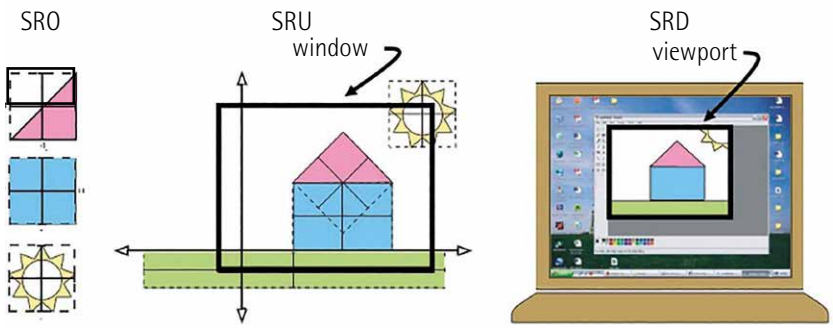


Figura 18

Na figura temos o esquema SRO-SRU-SRD. À esquerda, três primitivas (triângulo, quadrado e o sol) modelados no SRO, normalizado no intervalo [-1,1]. Ao centro, uma composição com essas primitivas, devidamente transformadas, rotação, escala e translação, inseridas no SRU com a respectiva window. À direita, temos um exemplo de dispositivo de saída gráfica com uma viewport exibindo a cena contida na window. Observe as distorções e recortes da cena. O piso é formado pela mesma primitiva, quadrado, repintada de verde.

2.4 Mapeamento de pontos (pixels) na janela de visualização

O mapeamento window-to-viewport ocorre diretamente entre os SRU e o SRD, mas podemos usar as coordenadas lógicas como coordenadas intermediárias no mapeamento.

O sistema de referência utilizado na modelagem de objetos é o SRN, que nada mais é do que o SRO limitado no intervalo [0, 1] ou [-1, 1] em cada uma de suas coordenadas.

Sejam, portanto, XRMIN e XRMAX os valores mínimos e máximos horizontais da janela no espaço cartesiano (window) e XPMIN e XPMAX os valores correspondentes em pixel (viewport). YRMIN, YRMAX, YPMIN e YPMAX serão os valores limites na vertical (conforme a figura a seguir). A transformação (XR,YR) para (YP,YP) é obtida por meio de um simples cálculo de proporcionalidade (regra de três simples!). Para a coordenada horizontal (coordenada x), a transformação é:

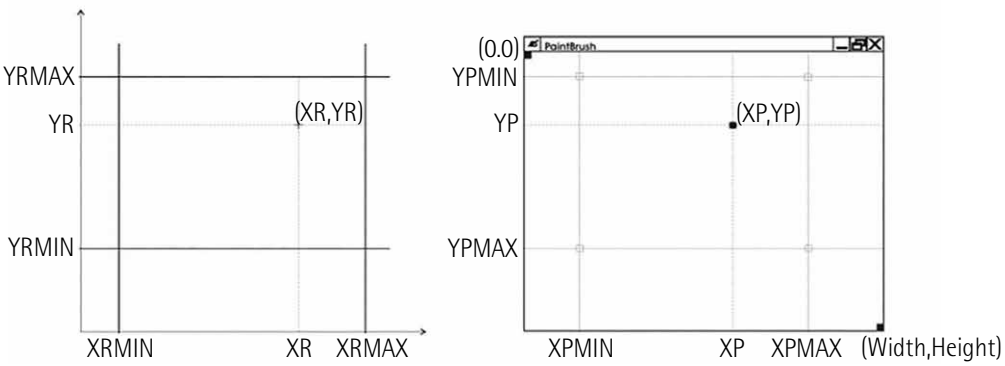


Figura 19

Resolvendo XP em função de XR, temos:

$$\frac{XP - XPMIN}{XR - XRMIN} = \frac{XPMAX - XPMIN}{XRMAX - XRMIN}$$

$$XP - XPMIN = \frac{(XR - XRMIN)(XPMAX - XPMIN)}{XRMAX - XRMIN}$$

$$XP = \frac{(XR - XRMIN)(XPMAX - XPMIN)}{XRMAX - XRMIN} + XPMIN$$

Denominando-se $SX = \frac{XPMAX-XPMIN}{XRMAX-XRMIN}$ como fator de escala horizontal, temos:

Com XR em função de XP, temos:

$$\frac{XR - XRMIN}{XP - XPMIN} = \frac{XRMAX - XRMIN}{XPMAX - XPMIN}$$

$$XR - XRMIN = \frac{(XP - XPMIN)(XRMAX - XRMIN)}{XPMAX - XPMIN}$$

$$XR = \frac{(XP - XPMIN)(XRMAX - XRMIN)}{XPMAX - XPMIN} + XRMIN$$

Denominando-se $SX=\frac{XPMAX-XPMIN}{XRMAX-XRMIN}$ como fator de escala horizontal, temos:

$$XR = \frac{(XP - XPMIN)}{SX} + XRMIN$$

Expressões análogas são obtidas da mesma forma para a transformação da coordenada vertical (y).

Ao implementarmos essas expressões em alguma linguagem de programação, devemos observar os tipos primitivos das variáveis (XR, YR) e (XP, YP). As primeiras são de tipo real, ponto flutuante, e as outras são de tipo inteiro, o que exige converter adequadamente os tipos.

Observação

A origem do sistema de coordenada em uma viewport é no canto superior esquerdo da janela de visualização. Por isso, atente-se ao montar a regra de três.

Exemplo de aplicação

Seguindo o raciocínio desenvolvido nos últimos parágrafos, obtenha as relações de YP em função de YR e de YR em função de YP. Considere que os limites da window e da viewport na vertical são, respectivamente, (YRMIN, YRMAX) e (YPMIN, YPMAX).

Onde

$$SY=\frac{YPMAX - YPMIN}{YRMAX - YRMIN}$$

$$\frac{YP - YPMAX}{YR - YRMIN} = \frac{YPMIN - YPMAX}{YRMAX - YRMIN}$$

$$YP = \frac{(YR - YRMIN)(YPMIN - YPMAX)}{YRMAX - YRMIN} + YPMAX$$

$$YP = -SY\ (YR - YRMIN\ ) + YPMAX$$

$$SY = \frac{YPMAX - YPMIN}{YRMAX - YRMIN}$$

$$\frac{YR - YRMIN}{YP - YPMAX} = \frac{YRMAX - YRMIN}{YPMIN - YPMAX}$$

$$YR = \frac{-(YP - YPMAX)}{SY} + YRMIN$$

$$YR = YRMIN - \left(\frac{YP - YPMAX}{SY}\right)$$

Considere o ponto P, pertencente ao SRU, cujas coordenadas (XR,YR) são P = (-1.45, 0.32). Sabendo que as coordenadas que definem a window são XRMIN = YRMIN = -2.0 e XRMAX = YRMAX = +2.0, qual o valor correspondente das coordenadas (XP,YP) do pixel em uma viewport de coordenadas XPMIN = YPMIN = 0, XPMAX = 1023 e YPMAX = 767?

Lembre-se de que:

XP = SX (XR - XRMIN) + XPMIN

YP = -SY YR - YRMIN) + YPMAX

XP = SX (XR - XRMIN)+XPMIN

$$SX = \frac{XPMAX - XPMIN}{XRMAX - XRMIN}$$

e

YP = -SY (YR - YRMIN)+YPMAX

$$SY = \frac{YPMAX - YPMIN}{YRMAX - YRMIN}$$

$$SX = \frac{1023 - (0)}{2 - (-2)} = 255,75$$

$$SY = \frac{767 - 0}{2-(-2)} = 191,75$$

Logo:

$$XP = 255{,}75 \cdot [-1{,}45 - (-2)] + 0 = 140.663$$

$$YP = -191{,}75 \cdot [0.32 - (-2)] + 767 = 322{,}14$$

Como XP e YP são os valores inteiros (coluna e linha no SRD, respectivamente), devem ser convenientemente arredondados. Assim: XP = 141; YP = 322.

Para que se entenda o mapeamento window-to-viewport perfeitamente, o resultado anterior significa que, uma vez definidas a window e a viewport, o ponto (-1.45, 0.32) no SRU corresponderá ao pixel (141, 322) no SRD.

O mapeamento de pontos na janela de visualização é uma operação importante em computação gráfica que permite que os objetos sejam exibidos corretamente na tela. As equações matemáticas fornecem um meio de realizar essa operação, transformando as coordenadas normalizadas em coordenadas de pixel na janela de visualização.

3 RASTERIZAÇÃO DE LINHAS

Já aprendemos a transformar um ponto do espaço vetorial R^2 em um ponto que representa um elemento de uma matriz de pixels. Essa transformação ocorre em ambos os sentidos, ou seja, de coordenadas cartesianas do SRU para pixels no SRD ou pixels do SRD para coordenadas cartesianas no SRU.

Agora veremos como traçar uma linha ligando dois pixels em uma janela de visualização gráfica.

Obviamente que ligar dois pixels mapeando (rasterizando) cada coordenada do SRU para pixel no SRD não é a tarefa mais adequada, considerando o tempo de processamento desse algoritmo. Bem mais simples é rasterizar apenas os vértices no interior da viewport e uni-los usando apenas os pixels existentes na matriz do SRD.

Os algoritmos que estudaremos adiante são o DDA (Digital Differential Analyser) e o de Bresenham.

Contudo, antes de conhecermos esses algoritmos, é fundamental relembrarmos alguns conceitos da geometria analítica.

3.1 Equação da reta

É uma função matemática que descreve uma linha em um sistema de coordenadas cartesianas.

É vital entender e saber manipular algebricamente as diferentes formas em que a equação da reta se apresenta, pois em computação gráfica a utilizamos em diversos contextos, por exemplo:

- **Rasterização de linhas**: usada em algoritmos de rasterização para desenhar linhas em uma tela ou imagem. O algoritmo usa a equação da reta para determinar os pixels que devem ser pintados para desenhar a linha.

- **Transformações geométricas**: aplicada em transformações geométricas para mover, rotacionar e escalar objetos em uma cena. A equação é usada para mapear pontos de um sistema de coordenadas para outro.

- **Detecção de colisões**: empregada para detectar colisões entre objetos em uma cena. Se as equações de duas retas se interceptam em algum ponto, isso significa que os objetos correspondentes estão colidindo.

- **Renderização de gráficos 3D**: utilizada em renderização de gráficos 3D para determinar quais objetos estão na linha de visão do observador. Isso é feito traçando uma linha do observador até cada ponto da cena e verificando se a linha intercepta algum objeto.

Essas são apenas algumas das aplicações da equação da reta em computação gráfica. A equação também é usada em outras áreas, como processamento de imagens, visão computacional e simulação de física.

Recordemos que a função f: R→R definida por f(x) = mx + b ou y = mx + b, com m e b ∈ e m ≠ 0, tem como gráfico uma reta.

Pela equação da reta, é possível estudar propriedades dessa reta, assim como, a partir de uma propriedade da reta, pode-se identificar a equação.

Conforme já acentuado, essa reta está inserida no sistema de referência cartesiano, composto de dois eixos, x e y, ortogonais entre si, e um ponto de origem O (intersecção dos eixos x e y). Os eixos x e y formam um plano, e cada eixo é constituído por valores numéricos que, unidos, formam os pares ordenados (x,y). Cada par ordenado (x,y) está associado biunivocamente a um único ponto do plano, conforme a figura a seguir:

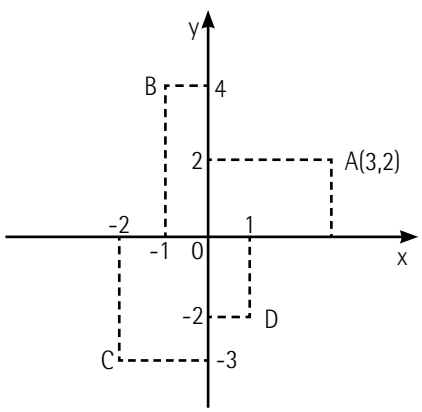


Figura 20

Quanto ao par ordenado de números reais, temos:

- (0,0) está associado ao ponto O (origem);
- (3,2) está associado ao ponto A;
- (-1,4) está associado ao ponto B;
- (-2,-3) está associado ao ponto C;
- (1,-2) está associado ao ponto D.

Considerando o ponto B(-1,4), dizemos que o número -1 é a coordenada x ou abscissa do ponto B, e o número 4 é a coordenada y ou a ordenada do ponto B.

Equação reduzida da reta

Na forma de equação reduzida, ela é denotada por y = mx + b, onde m é a inclinação da linha, também chamado coeficiente angular, e b é o intercepto no eixo y. A inclinação ou coeficiente angular representa a taxa de variação da linha e pode ser calculada como a diferença entre as coordenadas y (Δy) dividida pela diferença entre as coordenadas x (Δx) entre dois pontos na linha. O intercepto indica onde a linha cruza o eixo y e, portanto, tem coordenadas (0, b).

A figura a seguir representa o esquema para obtenção do coeficiente angular m.

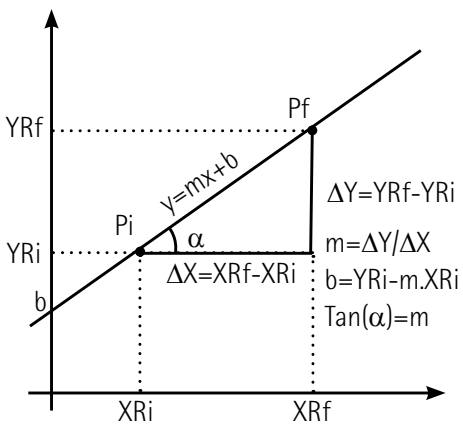


Figura 21 – Elementos da reta que passa por dois pontos

Assim, para determinar a equação de uma reta que passa por dois pontos, podemos fazer como no exemplo a seguir.

Exemplo de aplicação

Determine a equação da reta que passa pelos pontos $P_i = (2, 3)$ e $P_f = (4, 7)$.

Resolução

Sabendo que a equação da reta é definida pela expressão $y = mx + b$, devemos determinar os valores de m e de b.

Para isso, denominaremos $x_i = 2$, $x_f = 4$, $y_i = 3$ e $y_f = 7$

$$m = \frac{\Delta y}{\Delta x} = \frac{y_f - y_i}{x_f - x_i} = \frac{7 - 3}{4 - 2} \Rightarrow m = 2$$

Então, são considerados os valores de m de P_i ou o valor P_f e substituímos na expressão $y = mx + b$. Assim, determina-se o valor de b.

$y = mx + b \Rightarrow 7 = 2 \cdot 4 + b \Rightarrow b = 7 - 8 = -1$

Logo, $m = 2$ e $b = -1$

Conclui-se que a equação da reta que passa pelos pontos é P_i e $P_f = (4, 7)$ e $y = 2x - 1$.

Observação

A medida do ângulo deve sempre ser considerada do eixo x para a reta, no sentido anti-horário.

Equação geral da reta

Já na forma de equação geral, ela é denotada por $Ax + By + C = 0$. Os coeficientes A, B e C são denominados constantes e dependem da inclinação e do intercepto. Outra informação importante é que A e B jamais poderão ser nulos (iguais a zero) concomitantemente.

Observe os exemplos a seguir e aprenda como determinar a equação geral da reta a partir de outra equação da reta. É possível determinar a equação geral da reta a partir de qualquer outro tipo de equação que também a defina.

Exemplos

Conforme as equações da reta a seguir, determine as respectivas equações gerais.

$$y = -\frac{3}{4}x + 1$$

Resolução

Tira-se o m.mc:

$$\frac{4y = -3x + 4}{4}$$

Como se trata de uma igualdade, podemos desconsiderar o denominador da fração:

$4y = -3x + 4$

Iguala-se a equação a 0 passando todos os números para o primeiro membro e assim determina-se a equação geral da reta:

$3x + 4y - 4 = 0$, onde $A = 3$, $B = 4$ e $C = -4$

$$\frac{x}{2}+\frac{y}{5}=1$$

Resolução

Tira-se o m.mc:

$$\frac{5x + 2y = 10}{10}$$

Como se trata de uma igualdade, podemos desconsiderar o denominador da fração:

$5x + 2y = 10$

Iguala-se a equação a 0 passando todos os números para o primeiro membro e assim determina-se a equação geral da reta:

$5x + 2y - 10 = 0$, onde $A = 5$, $B = 2$ e $C = -10$

y = -5

Resolução

Neste caso, temos uma reta paralela ao eixo x.

Iguala-se a equação a 0 passando todos os números para o primeiro membro e assim determina-se a equação geral da reta:

$0x + 1y + 5 = 0$, $A = 0$, $B = 1$ e $C = 5$

Pode-se também determinar a equação geral da reta a partir de dois pontos conhecidos. Todavia, precisamos recordar a condição de alinhamento de três pontos. Vejamos.

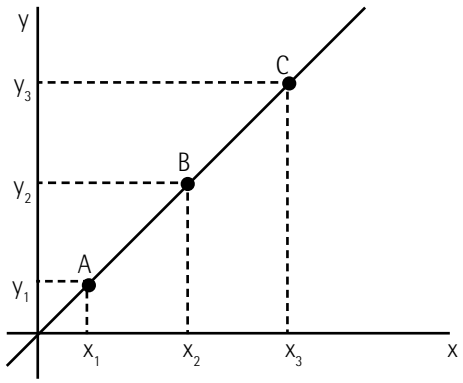


Figura 22

Ao analisar o gráfico, podemos aplicar as regras de proporcionalidade do teorema de Tales, assim:

$$\frac{x_2 - x_1}{x_3 - x_1} = \frac{y_2 - y_1}{y_3 - y_1}$$

Multiplicam-se em cruz as expressões:

$$(x_3 - x_1)(y_2 - y_1) = (x_2 - x_1)(y_3 - y_1)$$

Aplica-se a propriedade distributiva da multiplicação:

$$x_3y_2 - x_3y_1 - x_1y_2 + x_1y_1 = x_2y_3 - x_2y_1 - x_1y_3 + x_1y_1$$

Iguala-se a 0 passando todos os números para o primeiro membro e são feitas as simplificações possíveis:

$$x_3y_2 - x_3y_1 - x_1y_2 + \cancel{x_1y_1} - x_2y_3 + x_2y_1 + x_1y_3 - \cancel{x_1y_1} = 0$$

Reescreve-se ordenando os termos:

$$x_1y_2 - x_1y_3 + x_2y_3 - x_2y_1 + x_3y_1 - x_3y_2 = 0$$

O primeiro termo da igualdade corresponde ao determinante:

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}$$

Logo, pode-se afirmar que se três pontos $A(x_1,y_1)$, $B(x_2,y_2)$ e $C(x_3,y_3)$ estão alinhados:

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} = 0$$

E que:

$$\begin{vmatrix} x_A & y_A & 1 \\ x_B & y_B & 1 \\ x_3 & y_3 & 1 \end{vmatrix} = 0$$

Agora que recordamos a condição de alinhamento de três pontos, vejamos um exemplo para determinar a equação geral da reta, conhecido por dois pontos.

Observe mais um exemplo a seguir.

Encontre a equação geral da reta que passa pelos pontos A(2,1) e B(4,5).

Resolução

Monta-se o determinante:

$$\begin{vmatrix} x_A & y_A & 1 \\ x_B & y_B & 1 \\ x_3 & y_3 & 1 \end{vmatrix} = 0 \Rightarrow \begin{vmatrix} 2 & 1 & 1 \\ 4 & 5 & 1 \\ x & y & 1 \end{vmatrix} = 0$$

Calcula-se o determinante pela regra de Sarrus:

$$\begin{matrix} 2 & 1 & 1 & 2 & 1 \\ 4 & 5 & 1 & 4 & 5 = 0 \\ x & y & 1 & x & y \end{matrix}$$

$(2 \cdot 5 \cdot 1 + 1 \cdot 1 \cdot x + 1 \cdot 4 \cdot y) - (1 \cdot 4 \cdot 1 + 2 \cdot 1 \cdot y + 1 \cdot 5 \cdot x) = 0$

$10 + 1x + 4y - 4 - 2y - 5x = 0$

Operam-se os termos semelhantes e define-se a equação geral da reta:

$-4x + 2y - 6 = 0$

Lembrete

É possível determinar a equação geral da reta a partir de qualquer outro tipo de equação que também a defina.

3.2 Algoritmo DDA (Digital Differential Analyser)

Agora que recordamos os conceitos de geometria analítica sobre equações da reta, podemos estudar os algoritmos usados para rasterizar uma linha em uma janela de visualização gráfica.

Considere o SRU e seja (x_0,y_0) a coordenada do ponto inicial A (posição relativa do plano onde a reta se inicia) e (x_1, y_1) a coordenada do ponto final B (posição relativa do plano onde a reta termina), conforme a figura a seguir.

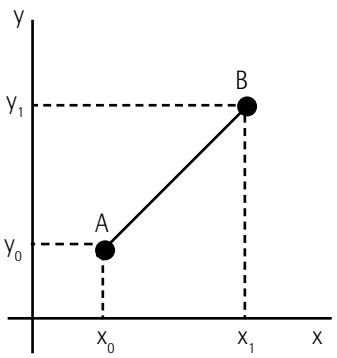


Figura 23

A equação reduzida da reta que passa por A e B, como visto anteriormente, é da forma: $y = mx + b$.

A princípio, aplicamos o mapeamento window-to-viewport para rasterizar os pontos A e B. Contudo, para fins didáticos, estabelecemos que os pontos inicial e final já estejam rasterizados (como é o caso da aplicação PaintBrush).

Observação

Nos softwares aplicativos como o Excel e o CAD/CAM, a rasterização dos pontos inicial e final é necessária.

Seja qual a for a circunstância, a rasterização da linha será mais eficiente quanto mais próximo da reta teórica estiverem os pixels que serão mapeados. Portanto, será na matriz de pixel da janela de visualização que ocorrerá a rasterização de linhas.

O algoritmo DDA pode ser implementado em várias linguagens de programação, incluindo C, C++, Java, Python, entre outras. Ele é usado em muitos aplicativos de desenho e gráficos, como editores de imagens, jogos e software de modelagem 3D.

A ideia do DDA é utilizar a equação reduzida da reta e incrementar a cada passo, uma unidade, na variável x.

Outros algoritmos incrementais, como do ponto médio, que veremos na sequência, pode-se incrementar tanto na variável x como na variável y.

Observe como funciona o DDA para uma reta de equação $y = mx + b$ que passa pelos pontos $A(x_0,y_0)$ e $B(x_1,y_1)$ para determinar uma fórmula geral de aplicação desse algoritmo.

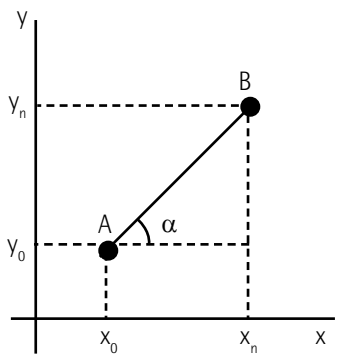


Figura 24

Determinamos o coeficiente angular m:

$n = x_n - x_0$ (define a quantidade de colunas de pixels entre A e B)

$$m = \frac{y_n - y_0}{n}$$

Conforme acentuado anteriormente, uma reta é uma função da forma $f(x) = mx + b$ e que a cada x se associa um único y. Logo, é válida a implicação de que, se $f(x) = y_0$, então $y_0 = m \cdot x_0 + b$.

Lembremos que é preciso mapear os pixels pertencentes às colunas existentes entre x_0 e x_n. Dessa forma, incrementamos uma unidade a cada x existente, a partir de x_1 até x_n e calculamos os y correspondentes:

Logo:

$x_1 = x_0 + 1$

$y_1 = m \cdot x_1 + b = m(x_0+1) + b = m \cdot x_0 + m + b = m \cdot x_0 + b + m$

$y_1 = y_0 + m$

Continuando (faça as contas):

$x_2 = x_1 + 1$, então $x_2 = x_0 + 2$

$y_2 = mx_2 + b = m(x_0 + 2) + b = mx_0 + 2m + b$

Logo:

$y_2 = y_0 + 2m$

Seguindo esse raciocínio, nota-se que:

$x_3 = x_0 + 3$

$y_3 = y_0 + 3_m$

⋮

$x_n\text{-}1 = x_0 + (n\text{-}1)$

$y_j = y_0 + (n\text{-}1) \cdot m$

O último ponto será o (x_n, y_n), onde:

$x_n = x_0 + n$

$y_n = y_0 + n \cdot m$

As expressões anteriores representam uma fórmula geral para mapear a coordenada de cada pixel dentro da viewport.

Como esse algoritmo foi pensado para inclinações entre -45° e 45°, significa que $|m| < 1$. Para outros valores, devemos usar simetrias, tema que estudaremos mais adiante.

A consequência imediata dessa restrição é que os valores calculados de y deverão ser arredondados ou truncados, visto que as coordenadas dos pixels só poderão assumir valores inteiros e positivos.

Exemplo:

Usando o DDA, compute quais pixels devem ser escolhidos para representar a reta que passa pelos pontos (6,9) e (11,12).

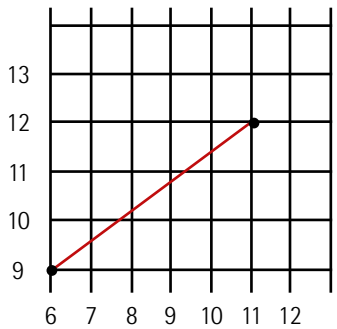


Figura 25

Com o objetivo de resolvermos esse exemplo, devemos usar as fórmulas dos termos gerais demonstradas anteriormente:

$$x_n = x_0 + n$$

$$y_n = y_0 + n \cdot m$$

Para tanto, devemos conhecer os valores de x_0, y_0 e de m. Os valores de x_0 e y_0 correspondem à coordenada onde o segmento de reta se inicia, logo:

$$x_0 = 6$$

$$y_0 = 9$$

Já o valor de m, que corresponde à inclinação da reta, ou coeficiente angular, é dado pelo quociente da diferença entre as componentes final e inicial de y, $y_f - y_i$ e a diferença entre as componentes final e inicial de x, $x_f - x_i$, assim:

$$m = \frac{\Delta y}{\Delta x} = \frac{y_f - y_i}{x_f - x_i}$$

$$m = \frac{12-9}{11-6} = 0{,}6$$

$x_1 = x_0 + 1 = 6 + 1 = 7$

$y_1 = y_0 + m = 9 + 0{,}6 = 9{,}6 \Rightarrow (x_1 ; y_1) = (7; 9{,}6)$

$x_2 = x_0 + 2 = 6 + 2 = 8$

$y_2 = y_0 + 2m = 9 + 1{,}2 = 10{,}2 \Rightarrow (x_2 ; y_2) = (8; 10{,}2)$

$x_3 = x_0 + 3 = 6 + 3 = 9$

$y_3 = y_0 + 3m = 9 + 1{,}8 = 10{,}8 \Rightarrow (x_3 ; y_3) = (9; 10{,}8)$

$x_4 = x_0 + 4 = 6 + 4 = 10$

$x_4 = y_0 + 4m = 9 + 2{,}4 = 11{,}4 \Rightarrow (x_4 ; y_4) = (10; 11{,}4)$

$x_5 = x_0 + 5 = 6 + 5 = 11$

$y_5 = y_0 + 5m = 9 + 3 = 12 \Rightarrow (x_5 ; y_5) = (11; 12)$

Como os pixels computados não podem assumir valores reais, e sim valores inteiros positivos, devemos arredondá-los. Assim:

$(7; 9{,}6) \sim (7; 10)$

$(8; 10{,}2) \sim (8; 10)$

$(9; 10{,}8) \sim (9; 11)$

$(10; 11{,}4) \sim (10; 11)$

Agora podemos observar como ficam os pixels acesos e computados em uma janela de visualização de algum dispositivo gráfico e compará-los à reta teórica dada:

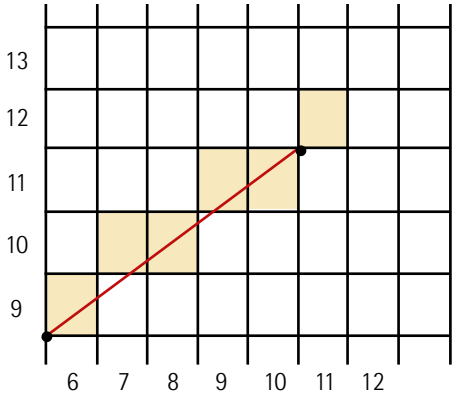


Figura 26

A seguir, um pseudocódigo em linguagem C para o cômputo dos pixels.

step=max of fabs(Y2 – Y1), fabs(X2 – X1);

Xinc = (X2 – X1)/step;

Yinc = (Y2 – Y1)/step;

step = Max of (fabs(11 – 6), fabs(12 – 9)) = 5

Xinc = (11 – 6)/5 = 1

Yinc = (12 – 9)/5 = 0,6

X = X + Xinc

Y = Y + Yinc

3.3 Algoritmo de Bresenham

Outra família de algoritmos é formada pelo algoritmo de Bresenham ou algoritmo do ponto médio.

Como dito anteriormente, o algoritmo de Bresenham é incremental, a diferença é que nele é possível incrementar uma unidade tanto na coordenada x quanto na coordenada y. Assim, teremos para cada passo x = x + 1, e o valor de y será incrementado ou não, dependo das diferenças calculadas a cada ponto.

Outra diferença é que o desenvolvimento do algoritmo se dá estabelecendo as diferenças das distâncias entre os pixels mais próximos à reta teórica que será rasterizada, em vez de usar a equação da reta, como no DDA. Assim, o algoritmo do ponto médio acenderá somente pixels adjacentes, fazendo com que a reta, no dispositivo gráfico, seja a mais contínua possível.

Diferentemente de outros algoritmos de rasterização de linhas retas, o algoritmo de Bresenham, durante sua aplicação, faz uso apenas de uma aritmética inteira dentro dos loops (passos), portanto não é necessário arredondar os valores obtidos. Tal fato acarreta maior eficiência no desempenho de processamento.

Uma semelhança com o DDA é que o algoritmo de Bresenham também foi desenvolvido para linhas com inclinação entre -45º e 45º.

Para entender melhor, observe a figura a seguir, que representa um dispositivo gráfico.

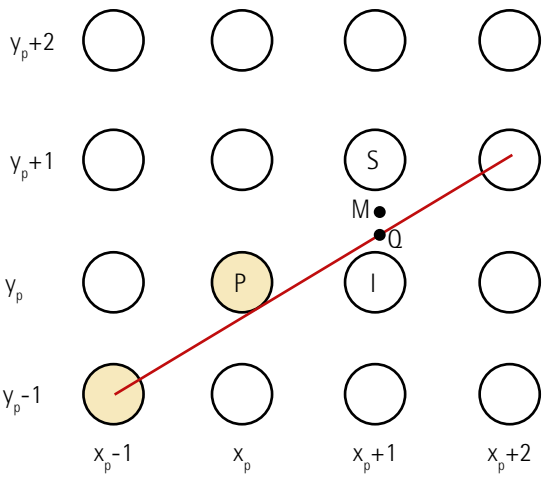


Figura 27

É possível observar que a reta teórica passa sobre o pixel de coordenadas (x_{p-1}, y_{p-1}) e tangenciando o pixel de coordenadas (x_p, y_p). Todavia, a reta teórica não encosta nos pixels I e S, que estão na coluna $x_p + 1$.

A pergunta que devemos fazer é: qual é o próximo pixel que deve ser aceso?

Devemos então decidir se acendemos o pixel I, incrementando 1 à coordenada x, ou o pixel S, incrementando 1 a ambas as coordenadas.

Voltando à figura, podemos observar o ponto M, denotado como o ponto médio da distância entre I e S e o ponto Q, que representa a intersecção entre a coluna $x_p + 1$ e a reta teórica.

Como o ponto Q está abaixo do ponto médio, acenderemos o pixel I por estar mais próximo da reta teórica. Caso contrário, se Q estivesse acima de M, o pixel S estaria mais próximo à reta teórica e o acenderíamos.

Concluindo, é a posição da reta teórica em relação ao ponto médio M que permitirá a escolha do próximo pixel a ser aceso.

Entendido como funciona o algoritmo do ponto médio, agora vamos fazer a demonstração algébrica de seu desenvolvimento:

Deve-se determinar o coeficiente angular da reta:

$\Delta x = x_2 - x_1$ e $\Delta y = y_2 - y_1$

$$m = \frac{\Delta y}{\Delta x}$$

A equação da reta é da forma y = mx + b, logo:

$$y = \left(\frac{\Delta y}{\Delta x}\right)x + b$$

Rearranjando os termos:

$$\left(\frac{\Delta y}{\Delta x}\right)x - y + b = 0$$

Multiplicando a expressão por Δx:

$x\Delta y - y\Delta x + b\Delta x = 0$

Agora podemos definir implicitamente uma função:

$f(x,y) = x\Delta y - y\Delta x + b\Delta x = 0$

Chamando $y\Delta = A$, $-\Delta x = B$ e $b\Delta x = C$, temos:

$f(x,y) = Ax + By + C = 0$

Não é difícil notar que:

- f(x,y) = 0 para todos os pontos sobre a reta (a reta intercepta M);
- f(x,y) > 0 para pontos abaixo da reta (M está abaixo da reta);
- f(x,y) < 0 para pontos acima da reta (M está acima da reta).

Portanto, calculando o valor da função para a coordenada do ponto médio $M(x_p+1, y_p+1/2)$, podemos verificar seu sinal e escolher o pixel que será aceso. Assim, criamos um parâmetro de decisão pd.

$$pd = f(M) = A\left(x_p + 1\right) + B\left(y_p + \frac{1}{2}\right) + C$$

Assim:

- se pd ≥ 0, incrementa-se x e y acendendo o pixel S.
- se pd < 0, incrementa-se x acendendo o pixel I.

Suponhamos que o pixel I foi aceso, logo incrementamos 1 na direção x e atualizamos pd.

$$pd_0 = f\left(x_p+1, y_p+\frac{1}{2}\right) = A\left(x_p+1\right) + B\left(y_p+\frac{1}{2}\right) + C$$

$$pd_1 = f\left(x_p+2, y_p+\frac{1}{2}\right) = A\left(x_p+2\right) + B\left(y_p+\frac{1}{2}\right) + C$$

Com essas informações, já é possível acender o pixel na tela de visualização, entretanto devemos determinar o valor inicial do pd, ou seja, pd_i.

$$pd_i = f\left(x_1+1, y_1+\frac{1}{2}\right) = A\left(x_1+2\right) + B\left(y_1+\frac{1}{2}\right) + C$$

$$dp_i = Ax_1 + By_1 + C + \frac{B}{2} + A = f\left(x_1,y_1\right) + \frac{B}{2} + A$$

$$dp_1 = \frac{B}{2} + A$$

$$dp_1 = \Delta y - \frac{\Delta x}{2}$$

$$dp_i = 2\Delta - \Delta x$$

- Se $pd_i \geq 0$, o próximo pd, ou seja, $pd_{i+1} = pd_i + 2_{\Delta y} - 2_{\Delta x}$
- Se $pd_i < 0$, o próximo pd, ou seja, $pd_{i+1} = pd_i + 2_{\Delta y}$

Veja agora um passo a passo de como implementar o algoritmo de Bresenham para a reta que passa pelos pontos (x_i, y_i) e (x_f, y_f) (pontos inicial e final do segmento de reta).

1) Calcula-se $\Delta x = x_f - x_i$ e $\Delta y = y_f - y_i$

2) Considere que o ponto inicial seja: $x = x_i$ e $y = y_i$

3) Calcula-se o parâmetro de decisão inicial: $pd_i = 2\Delta y - \Delta x$

4) Plota-se (acende-se) o ponto (x, y).

5) Se pd_i for negativo (isto é, se $pd_i < 0$), então incrementa-se na direção $x = x + 1$ e calcula-se o próximo pd, assim: $pd_{i+1} = pd_i + 2Dy$ e passa-se para o passo 7.

6) Se pd_i for positivo ou zero (isto é, $pd_i \geq 0$), incrementa-se na direção $x = x + 1$ e na direção $y = y + 1$ e calcula-se o próximo pd, ou seja, $pd_{i+1} = pd_i + 2_{\Delta y} - 2_{\Delta x}$

7) Repetem-se os passos 4 a 6 até que o ponto final, (xf, yf), seja alcançado.

Exemplo

Calcule os pixels que devem ser plotados (acesos) no segmento de reta compreendido entre os pontos (20,10) e (30,18) utilizando o algoritmo de Bresenham.

Resolução

Vamos seguir o passo a passo.

$\Delta x = x_2 - x_1 = 30 - 20 \Rightarrow \Delta x = 10$

$\Delta y = y_2 - y_1 = 18 - 10 \Rightarrow \Delta y = 8$

Coloca-se nas variáveis de trabalho o ponto inicial: $x = 20$ e $y = 10$

Calcula-se o parâmetro de decisão inicial: $pd_i = 2\Delta y - \Delta x$

$pd_0 = 2\Delta y - \Delta x = 2 \cdot 8 - 10 \Rightarrow pd_0 = 6$, logo $pd_0 \geq 0$, então (21,11)

$pd_1 = pd_0 + 2\Delta y - 2\Delta x = 6 + 16 - 20 \Rightarrow pd_1 = 2$, logo $pd_1 \geq 0$, então (22,12)

$pd_2 = pd_1 + 2\Delta y - 2\Delta x = 2 + 16 - 20 \Rightarrow pd_2 = -2$, logo $pd_2 < 0$, então (23,12)

$pd_3 = pd_2 + 2\Delta y = -2 + 16 \Rightarrow pd_3 = 14$, logo $p_3 \geq 0$, então (24,13)

$pd_4 = pd_3 + 2\Delta y - 2\Delta x = 14 + 16 - 20 \Rightarrow pd_4 = 10$, logo $p_4 \geq 0$, então (25,14)

$pd_5 = pd_4 + 2\Delta y - 2\Delta x = 10 + 16 - 20 \Rightarrow pd_5 = 6$, logo pd_5 0, $\geq$ então (26,15)

A partir daqui os parâmetros se repetem, assim, os próximos pontos serão (27,16); (28,16); (29,17) e (30,18)

Observe agora a reta teórica e os pixels plotados na janela de visualização.

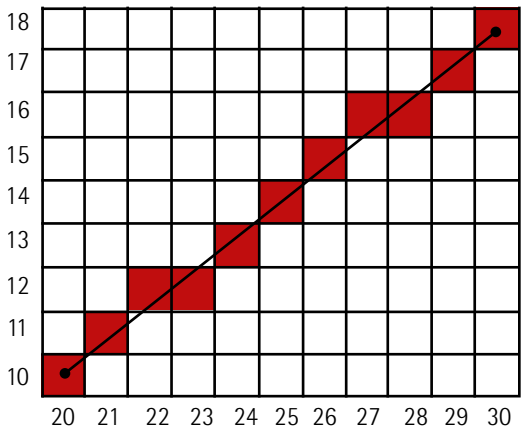


Figura 28

A figura a seguir ilustra um pseudocódigo de implementação do algoritmo de Bresenham.

```
x = x1;
y = y1;
dx = x2-x1;
dy = y2-y1;

d = (2*dy-dx);
incI = 2*dy; /* incE */
incS = 2(dy - dx);

PutPixel(x,y);

while(x!=x2){ // desenha até chegar ao ponto final
   if(d<=0){ // verifica a posição do ponto(abaixo||acima)
      d = d+incI;
      x++;
   }
   else{
      d+=incS;
      x++;
      y++;
}
PutPixel(x,y);
}
```

Figura 29

Fonte: Diniz (2014, p. 8).

3.4 Extensão para traçado de linhas em qualquer direção

A extensão para traçado de linhas em qualquer direção em computação gráfica refere-se a um conjunto de algoritmos que permitem traçar uma linha em qualquer direção, não apenas nas direções vertical, horizontal e diagonal. Esses algoritmos são baseados em técnicas de interpolação e aproximação para determinar os pixels mais próximos da linha desejada.

Essa extensão é importante porque muitas aplicações de computação gráfica exigem que as linhas possam ser desenhadas em qualquer direção. Por exemplo, em programas de desenho vetorial ou CAD, é importante que as linhas possam ser traçadas em qualquer ângulo para permitir a criação de formas complexas e curvas suaves.

Existem vários algoritmos que podem ser usados para estender o traçado de linhas em qualquer direção, como o algoritmo de Wu, o de Xiaolin Wu e o algoritmo de anti-aliasing de Xiaolin Wu. Eles são baseados em diferentes técnicas matemáticas, mas todos têm em comum a capacidade de traçar linhas suaves e precisas em qualquer direção.

Em geral, eles funcionam ao determinar os pixels mais próximos da linha desejada usando técnicas de interpolação e aproximação. Eles também podem incorporar técnicas de anti-aliasing para produzir linhas mais suaves e com menos aliasing.

No geral, a extensão para traçado de linhas em qualquer direção é um recurso importante na computação gráfica que permite a criação de formas complexas e linhas suaves em uma ampla gama de aplicações.

O algoritmo de Bresenham é uma técnica eficiente para traçar linhas em direções fixas, mas quando se trata de linhas inclinadas, é necessária uma abordagem diferente.

Um algoritmo comum para essa extensão é o algoritmo de linha digital (Digital Line Algorithm), que utiliza a equação da reta em sua forma simétrica:

$$\frac{|x_2 - x_1|}{d \text{ (diagonal)}} = \frac{|y_2 - y_1|}{s \text{ (step)}}$$

Figura 30

O algoritmo começa a partir do ponto inicial (x_1, y_1) e move-se na direção de x ou y, dependendo de qual diferença absoluta é maior. Em cada etapa, o ponto mais próximo da linha ideal é escolhido para ser desenhado. O erro de cada etapa é calculado e acumulado, e a decisão de mover para o próximo ponto é tomada com base nesse erro.

Por exemplo, considere os pontos (2,3) e (9,8). O primeiro passo é calcular dx e dy:

```
dx = 9 - 2 = 7
dy = 8 - 3 = 5
```

Figura 31

Aqui, a diferença absoluta de dy é maior que dx, então a linha será desenhada na direção y. Começando em (2, 3), a linha deve passar pelos pontos (3, 4), (4, 5), (5, 6), (6, 7) e (7, 8). Para cada passo, o valor de y é incrementado em 1 e o valor de x é incrementado ou decrementado, dependendo do valor acumulado de erro. O valor inicial de erro é 0.5, já que estamos começando no ponto (2, 3) e a linha ideal passa pelo ponto (2.5, 3.5):

y	x	err
3	2	0.5
4	3	1.5
5	4	2.5
6	5	3.5
7	6	4.5
8	7	5.5

Figura 32

Aqui, o valor de x é incrementado em cada etapa, e o valor de erro é atualizado com a diferença absoluta entre dy e dx. Quando o valor de erro atinge 1, o valor de x é incrementado ou decrementado, dependendo da direção da linha, e o valor de erro é subtraído de 1.

Esse algoritmo pode ser usado para traçar linhas em qualquer direção e com uma precisão aceitável para a maioria das aplicações

Lembrete

A extensão para traçado de linhas em qualquer direção é uma técnica aplicada em computação gráfica para traçar linhas em qualquer direção, não apenas na horizontal e vertical.

3.5 Técnicas de antisserrilhamento (anti-aliasing)

Também conhecidas como anti-aliasing, são utilizadas em computação gráfica para reduzir ou eliminar o efeito de serrilhamento (jaggies) nas bordas de objetos ou linhas diagonais em imagens digitais. O serrilhamento ocorre porque a imagem é formada por pixels, que são elementos discretos, e os objetos ou linhas que não estão alinhados com a grade de pixels podem ser representados de forma escalonada.

Existem várias técnicas de antisserrilhamento que podem ser usadas, incluindo:

- **Supersampling**: envolve a renderização da imagem em uma resolução mais alta do que a resolução final desejada e, em seguida, reduzir a imagem para a resolução desejada. Isso pode suavizar as bordas das linhas e dos objetos.

- **Multisampling**: engloba a amostragem de vários pontos em cada pixel e a combinação dessas amostras para obter uma cor final para o pixel. Isso pode ajudar a reduzir o efeito de escalonamento.

- **Filtro de suavização**: abrange a aplicação de um filtro que suaviza as bordas das linhas e objetos para reduzir o efeito de escalonamento.

- **Pós-processamento**: compreende a aplicação de filtros ou efeitos especiais após a renderização da imagem para suavizar as bordas das linhas e dos objetos.

Em geral, as técnicas de antisserrilhamento são importantes para produzir imagens de alta qualidade em computação gráfica e torná-las mais realistas.

A figura a seguir apresenta em A a conversão da representação de uma reta na forma vetorial para a matricial. Em B, é incluído um tratamento de anti-aliasing.

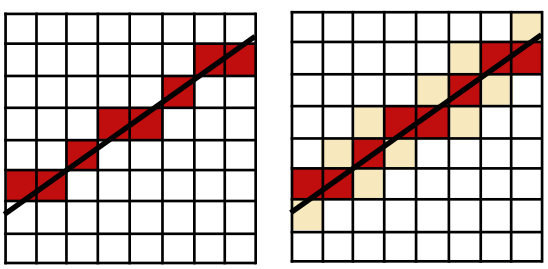


Figura 33

Fonte: Azevedo e Conci (2003a, p. 255).

O modelo matemático das técnicas de antisserrilhamento pode variar dependendo do método aplicado. No entanto, em geral, essas técnicas envolvem o uso de filtros para suavizar as bordas das

linhas e dos objetos na imagem. Esses filtros normalmente são baseados em funções matemáticas, como filtros de média, gaussiano, triangular ou cúbico.

Outro exemplo simples de técnica de antisserrilhamento é a supersampling. Nesse método, a imagem é renderizada em uma resolução mais alta do que a resolução final desejada. Para cada pixel da imagem final, vários pixels são renderizados na resolução mais alta e combinados para produzir uma cor final para o pixel. Isso ajuda a suavizar as bordas das linhas e dos objetos na imagem.

Destaca-se também o filtro gaussiano, que é comumente usado em técnicas de antisserrilhamento. Ele é baseado na função gaussiana, que é uma curva de sino simétrica. O filtro gaussiano é aplicado na imagem para suavizar as transições de cor nas bordas dos objetos e linhas diagonais, reduzindo o efeito de escalonamento. O efeito do filtro pode ser controlado pela variação da amplitude da curva de sino e da largura da função gaussiana.

Em resumo, as técnicas de antisserrilhamento são baseadas em modelos matemáticos que envolvem o uso de filtros para suavizar as bordas das linhas e dos objetos na imagem. Esses filtros podem variar em complexidade e eficácia, e a escolha do método depende das necessidades específicas de cada aplicação.

4 RASTERIZAÇÃO DE CURVAS

A rasterização de curvas em computação gráfica é o processo de converter uma representação vetorial de uma curva em uma representação baseada em pixels. Esse processo é necessário porque as telas dos monitores e os dispositivos de saída gráfica são baseados em pixels, o que significa que não é possível exibir diretamente curvas ou outros objetos vetoriais.

Existem várias técnicas para a rasterização de curvas em computação gráfica, e as mais comuns incluem:

- **Rasterização de linhas**: emprega a geração de uma sequência de pixels ao longo de uma linha reta entre dois pontos. Isso pode ser usado para rasterizar linhas retas ou segmentos de curvas.

- **Algoritmos de Bresenham**: permitem a rasterização de linhas retas ou curvas suaves a partir de sua representação vetorial. Eles funcionam determinando qual dos dois pixels vizinhos deve ser preenchido em cada etapa ao longo da linha ou curva.

- **Curvas de Bézier**: são um tipo comum de curvas usadas em gráficos vetoriais. Para rasterizar uma curva de Bézier, ela pode ser dividida em segmentos retos e cada segmento pode ser rasterizado usando técnicas de rasterização de linha.

- **Técnicas de antisserrilhamento**: aplicadas para suavizar as bordas das curvas rasterizadas.

Destacaremos a seguir métodos para traçar circunferências de círculos e circunferências de elipses usando a aritmética inteira do algoritmo de Bresenham. Pode parecer estranho o uso do termo

"circunferência de círculo", mas há uma pequena diferença: circunferência refere-se ao perímetro, ou ao menos ao perímetro de uma envoltória; e círculo, à área.

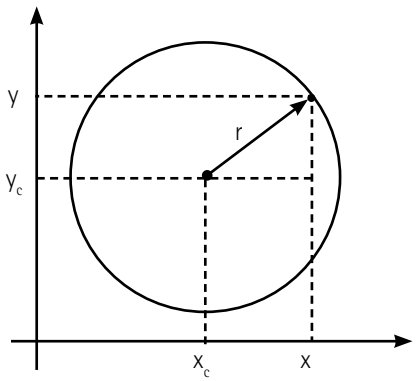


Figura 34 – Geometria da circunferência

Além desses algoritmos, bastante eficientes, mas limitados às formas integrais que expressam, veremos como podemos traçar a maioria das curvas e arcos usando coordenadas polares. Infelizmente, esses algoritmos padecerão da aritmética de ponto flutuante, por causa das funções seno e cosseno que entram em suas definições.

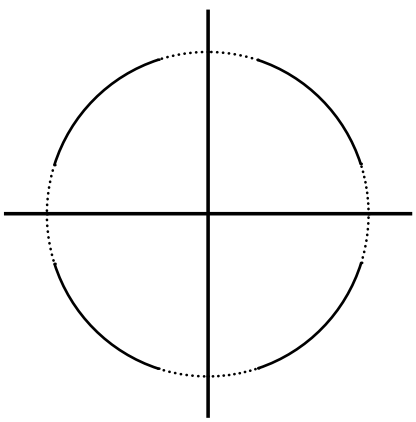


Figura 35 – Imprecisões no traçado da circunferência

4.1 Equação da circunferência

Uma circunferência de círculo, ou simplesmente circunferência, é definida como o conjunto dos pontos que estão a uma mesma distância de um ponto central. A distância é o raio (r) da circunferência, e o ponto equidistante a todos os outros é o centro da circunferência (x_c, y_c). Matematicamente, a equação geral da circunferência é dada por:

$$(x - x_c)^2 + (y - y_c)^2 = r^2$$

Onde (x_c, y_c) é o centro da circunferência e r é o raio da circunferência (conforme a figura da geometria de circunferência).

Essa equação pode ser usada para determinar se um ponto (x, y) está dentro, fora ou na borda da circunferência. Se a distância entre o ponto e o centro da circunferência for menor ou igual ao raio, o ponto está dentro da circunferência. Se a distância for maior que o raio, o ponto está fora da circunferência. Se a distância for igual ao raio, o ponto está na borda da circunferência.

Essa equação não é conveniente para ser usada em computação gráfica, pois nem sequer é uma função (lembra-se da definição de função?), porque se deseja uma definição do tipo y=f(x) ou x=g(y).

Essas definições podem ser facilmente obtidas isolando-se as variáveis:

$$\begin{cases} x = x_c \pm \sqrt{r^2 - (y - y_c)^2} \\ y = y_c \pm \sqrt{r^2 - (x - x_c)^2} \end{cases}$$

A operação ± é necessária, pois a raiz quadrada pode ser positiva ou negativa. Isso significa que para cada valor de x (na primeira expressão) são obtidos dois valores de y: um para a metade superior da circunferência e o outro para a metade inferior.

Essas expressões, apesar de rigorosamente corretas, apresentam dois problemas para serem usadas em computação gráfica:

- Exigem muitos cálculos (quadrados e raiz quadrada).
- Geram imprecisão no traçado. Quando o arco de circunferência fica quase horizontal ou vertical, um pequeno incremento em x = x+1 (ou y = y+1) leva a um salto, e o arco apresentar-se-á descontínuo (conforme a figura de imprecisões no traçado de circunferência).

Agora vamos estudar outros métodos para o traçado de circunferências.

4.2 Traçado de curvas usando coordenadas polares

O traçado de curvas usando coordenadas polares em computação gráfica é uma técnica para desenhar curvas bidimensionais usando uma parametrização polar. Nesta técnica, a curva é descrita como uma função do um raio e o ângulo polar θ, em vez das coordenadas cartesianas x e y.

A função que descreve a circunferência em coordenadas polares é dada por:

$$\begin{cases} x = r \cdot \cos(\theta) \\ y = r \cdot \text{sen}(\theta) \end{cases}$$

Onde r é a distância do ponto à origem e θ é o ângulo polar.

O ângulo θ (téta) é um ângulo que varia entre 0 e 2π (os ângulos devem ser tratados com unidades em radianos, 2π radianos é igual a 360°). O número de passos deverá aumentar com o raio. Um algoritmo trivial de traçado de circunferências pode ser idealizado a partir da escolha de um passo para θ, que chamaremos de Dθ, e um loop que calcula os valores de (x, y) para vários valores de θ. Cada ponto calculado é inserido numa tabela que mais tarde é fornecida como parâmetro para um traçado de uma polilinha fechada. A circunferência desenhada será na realidade um polígono, cuja quantidade de lados está relacionada à precisão desejada:

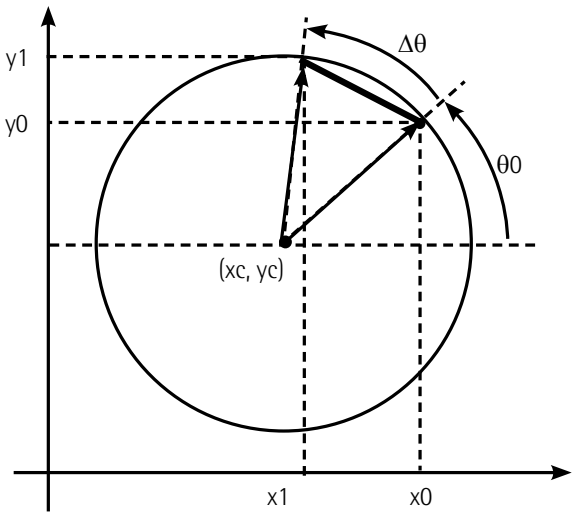


Figura 36 – Circunferência de coordenadas polares

1) Leia as coordenadas do centro (x_c, y_c) e o raio r.

2) Leia o número (n) de vezes em que será dividida a circunferência.

3) Calcule $D\theta = 2\pi/n$

4) Para i de 0 até n, faça:

$\theta = i * D\theta$

$$\begin{cases} x = r \cdot \cos(\theta) \\ y = r \cdot \text{sen}(\theta) \end{cases}$$

5) Converta (x,y) para pixels da viewport.

6) Una os pixels na sequência com linhas retas usando-se, por exemplo, o algoritmo de Bresenham.

Note que o pseudocódigo anterior serve não apenas para o traçado de circunferências, mas para qualquer curva representável em coordenadas polares. Para tanto, basta substituir, convenientemente, a fórmula do passo 4.

A seguir, destacam-se exemplos de curvas que podem ser construídas com coordenadas polares:

- **Espiral de Arquimedes**: $r = a + b\theta$, onde a e b são constantes que determinam o tamanho e a forma da espiral. Essa curva tem a forma de uma espiral que se expande continuamente.

- **Hipérbole equilátera**: $r = a / \cos(\theta)$, onde a é uma constante que define o tamanho da hipérbole. Essa curva tem duas assíntotas que se encontram nos ângulos $\theta = \pm\pi/2$.

- **Cardioide**: $r = a(1 + \cos(\theta))$, onde a é uma constante que estipula o tamanho da cardioide. Essa curva tem a forma de um coração.

- **Lemniscata de Bernoulli**: $r^2 = a^2 \cos(2\theta)$, onde a é uma constante que determina o tamanho da lemniscata. Essa curva tem a forma de uma figura em forma de 8.

Essas curvas são usadas em várias aplicações de computação gráfica, como na criação de formas complexas em gráficos 2D, animações e efeitos visuais.

Apesar de simples e eficaz, esse algoritmo apresenta dois defeitos: a precisão depende do raio da circunferência; e o cálculo de funções trigonométricas (senos e cossenos), apesar dos poucos pontos, implica perda de tempo e agilidade.

Saiba mais

Conheça outras curvas e suas respectivas equações em coordenadas polares acessando os seguintes materiais:

FRIENDLY, M.; DENIS, D. J. *Marcos na história da visualização de dados*. Tradução: Mário Kanno. Toronto: Universidade de York, [s.d.].

Disponível em: https://www.mathwords.com/. Acesso em: 30 mar. 2023.

WJEC FURTHER MATHEMATICS. *Polar coordinates*. [s.d.]. Disponível em: https://bit.ly/3G4iaLU. Acesso em: 30 mar. 2023.

Disponível em: http://xahlee.org/. Acesso em: 30 mar. 2023.

4.3 Algoritmo de Bresenham ou ponto médio para circunferências e elipses

Iniciemos este tópico com as circunferências.

O algoritmo do ponto médio para circunferências foi desenvolvido com o objetivo de agilizar o traçado da circunferência. É otimizado levando-se em conta as simetrias presentes nas circunferências, a simetria por octantes. O ponto calculado em um octante (um oitavo de circunferência, com θ indo de 0 até $\pi/4$ apenas, ou 45°) pode ser copiado para os outros octantes, conforme a figura a seguir:

Octante	X_n	Y_n
I	Y	X
II	X	Y
III	-X	Y
IV	-Y	X
V	-Y	-X
VI	-X	-Y
VII	X	-Y
VIII	Y	-X

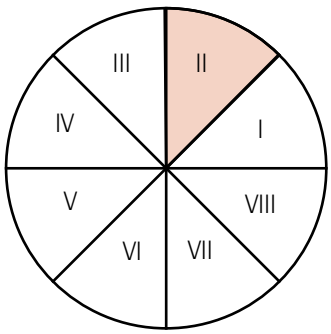


Figura 37 – Simetria de octantes para o algoritmo de circunferência

Observe que os pontos calculados como (x, y), diferentemente dos algoritmos de rasterização de retas, são obtidos para o octante II, como veremos em seguida. Outra coisa muito importante referente às simetrias é que cada algoritmo incremental que se crie pode usar um critério próprio que não obedeça, necessariamente, ao da figura anterior. Portanto, a tabela não é para ser decorada, mas sim entendida. Dessa forma, o conceito de simetria como empregado em computação gráfica será muito mais útil e geral.

O algoritmo segue os mesmos critérios do algoritmo de Bresenham para retas:

1) Elege-se como ponto inicial (0, r), que é plotado nos oito octantes, e ainda se calcula uma variável auxiliar p = 1 – r.

2) Incrementa-se x.

3) Se p for negativo, então calcula-se um novo p = p + 2x + 1; caso contrário, decrementa-se y e calcula-se p= p + 2x +1 – 2y.

4) Plota-se o novo ponto (x,y) nos oito octantes.

5) Os passos 2, 3 e 4 são repetidos enquanto x < y.

Note que ele não possui cálculos complexos (raiz quadrada, senos ou cossenos), sendo, portanto, eficiente e rápido. Sua simplicidade também permite que seja implementado em linguagem de máquina ou ainda em hardware de aceleradores gráficos.

Uma última observação: note que, no algoritmo, os valores calculados são relativos ao centro da circunferência, por isso podem assumir valores positivos (à direita ou acima do centro) e negativos (à esquerda ou abaixo do centro). Assim, além de considerar a simetria da figura, devemos adicionar os valores de (x_c, y_c) em cada ponto.

A seguir dois exemplos de aplicação.

Exemplos de aplicação

Exemplo 1

Considere uma função plota(x, y) que acende um pixel na posição (x, y) da tela. Considere, ainda, o pixel de coordenadas (0, 0), que está situado no canto inferior esquerdo da tela (coordenadas em um sistema lógico). Com o algoritmo do ponto médio (Bresenham), obtivemos, para o segundo octante, o ponto (300,400). Como a circunferência que queremos rasterizar está centrada no pixel (700,100), a chamada para a função que irá acender, por simetria, o pixel no quarto octante é:

A) Plota(400,300)

B) Plota(700,100)

C) Plota(300,400)

D) Plota(400,100)

E) Plota(700,300)

Resolução

Trata-se de um problema no domínio da viewport, não sendo necessária qualquer conversão entre o SRU e o SRD. Como vimos, o algoritmo de Bresenham para circunferências calcula os valores de pixel apenas para o segundo octante e considerando a circunferência centrada na origem (0,0). Uma vez calculado o ponto, ao plotarmos o resultado na viewport devemos deslocá-lo relativamente ao centro da circunferência estabelecido pelo usuário. Isso é feito somando-se aos valores de (x, y) os valores de (xc, yc) do centro da circunferência. Assim, em pseudocódigo:

PLOTA (XC + X, YC + Y)

Um ponto no quarto octante (conforme a figura) tem as seguintes propriedades relativas: o valor de x é igual ao valor calculado para y, só que de sinal negativo, e o valor de y é igual ao calculado para x, com sinal positivo. Assim, teremos:

PLOTA (XC – Y, YC + X) = PLOTA (700 – 400, 100 + 300) = PLOTA (300, 400)

Desse modo, a alternativa correta é C.

Nota: suponha que se queira implementar o algoritmo de Bresenham para circunferências apresentado. Nesse caso, o passo 4 deve ser considerado como:

PLOTA($x_c + y$, $y_c + x$)

PLOTA($x_c + x$, $y_c + y$)

PLOTA($x_c - x$, $y_c + y$)

PLOTA($x_c - y$, $y_c + x$)

PLOTA($x_c - y$, $y_c - x$)

PLOTA($x_c - x$, $y_c - y$)

PLOTA($x_c + x$, $y_c - y$)

PLOTA($x_c + y$, $y_c - x$)

Onde x e y são os únicos valores calculados pelo algoritmo.

Exemplo 2

Seja uma circunferência de raio 10 pixel e centro em (0,0), considere uma simulação da aplicação do algoritmo do ponto médio para curvas:

Resolução

Ponto inicial (0,r) = (0,10)

p = 1- r

p<0 – p = p+2x+1

P>0 – p = p + 2x+1 – 2y

Tabela 1

Loop	Fórmula p	Valor de p	(x, y)	2x	2y
0	1 – 10	-9	(0,10)	0	20
1		-9	(1,10)	2	20
2	-9 + 2 + 1	-6	(2,10)	4	20
3	-6 + 4 + 1	-1	(3,10)	6	20
4	-1 + 6 + 1	6	(4,9)	8	18
5	6 + 8 + 1 – 18	-3	(5,9)	10	18
6	-3 + 10 + 1	8	(6,8)	12	16
7	8 + 12 + 1 – 16	5	(7,7)	14	14

A seguir, os pixels acesos na janela de visualização para o octante II.

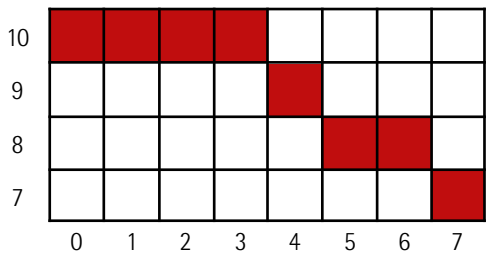


Figura 38

Para traçar uma elipse, o algoritmo de Bresenham é modificado para levar em conta a relação entre o eixo maior e o eixo menor da elipse. Nesse caso, os valores iniciais de x e y são escolhidos de acordo com a relação entre os eixos, e os cálculos de pixel são feitos levando ponderando a inclinação da elipse.

A modelagem matemática do algoritmo de Bresenham para elipses é baseada na escolha de um ponto central (xc, yc), nos eixos maior a e menor b e na definição das variáveis x e y iniciais.

As variáveis x e y iniciais são (0, b), onde b é o eixo menor da elipse, e a equação da elipse é dada por:

$$\frac{(x-xc)^2}{a^2}+\frac{(y-yc)^2}{b^2}=1$$

O algoritmo de Bresenham modificado para elipses considera a relação entre os eixos maior e menor da elipse. Nesse caso, a cada iteração o algoritmo verifica se o pixel mais próximo da elipse está em uma das quatro posições possíveis: (x+1, y), (x+1, y-1), (x, y-1) ou (x-1, y-1). Para determinar qual pixel é o mais próximo, o algoritmo calcula a distância entre o pixel e a elipse e escolhe o pixel que tem a menor distância.

Resumo

Nesta unidade, vimos que a computação gráfica pode ser entendida como uma área da ciência da computação que estuda a geração, manipulação e interpretação de modelos matemáticos, na forma de imagens, utilizando o computador. Para isso, ela se subdivide em três campos de estudo: síntese de imagens; processamento (manipulação); análise de imagens (visão computacional).

Em seguida, estudamos as primitivas gráficas, elementos básicos de gráficos/desenhos a partir dos quais são construídos outros objetos, mais complexos, mas também entram na definição de primitivas os comandos e funções que manipulam e alteram os elementos gráficos de uma imagem.

- **em 2D**: pontos, linhas, polilinhas, retângulos, circunferência, elipse etc.
- **em 3D**: planos, caixas, esferoides, cilindros, cones etc.

Também servem como exemplos de primitivas gráficas funções como:

- **em 2D**: WritePixel(x, y,cor), cor = GetColor(x,y), Form1.Canvas.Pixel[x,y]:=RGB(r,g,b)
- **em 3D**: plane{ y,0 texture{pigment{color rgb <r,g,b>}}}}

Dois termos muito empregados em computação gráfica e processamento de imagens são rasterização (rastering) e renderização (rendering). Rasterização é a tarefa de tomar uma imagem descrita vetorialmente e convertê-la em uma imagem raster (matriz de pixel) para a saída em vídeo ou impressora. Já renderização é o processo pelo qual se pode obter o produto final de um processamento digital qualquer.

Prosseguindo nosso estudo, acentuamos o pixel, a menor unidade gráfica manipulável. Vimos que só podemos lhe atribuir duas propriedades: cor e posição. Ele está inserido em uma matriz de tamanho W x H (Width – largura; Height – altura) do dispositivo gráfico de saída, cujas coordenadas pertencem ao N2.

Ao analisar o mapeamento window to viewport, destacamos que é possível transformar um ponto (XR, YR) do SRU em pixel (XP, YP) aceso no SRD através do uso de proporções, e o contrário também é valido:

$$\begin{cases} XP = SX(XR - XRMIN) + XPMIN \\ XR = \dfrac{XP - XPMIM}{SX} + XRMIN \end{cases}$$

e

$$\begin{cases} YP = -SX(YR - YRMIN) + YPMAX \\ YR = YRMIN - \dfrac{(YP - YPMAX)}{SY} \end{cases}$$

Onde o fator de escala horizontal SX e o fator de escala vertical SY são dados, respectivamente, por:

$$SX = \frac{XPMAX - XPMIN}{XRMAX - XRMIN}$$

$$SY = \frac{YPMAX - YPMIN}{YRMAX - YRMIN}$$

Também estudamos a rasterização de linhas, em especial, o algoritmo DDA e o algoritmo de Bresenham. Na rasterização de curvas, destacamos a equação da circunferência, o traçado de curvas usando coordenadas polares e o algoritmo de Bresenham ou ponto médio para circunferências e elipses.

Exercícios

Questão 1. (Avança SP 2022, adaptada) A computação gráfica está em todo lugar e é o grande elo na comunicação entre a humanidade e os computadores. Podemos encontrar a computação gráfica nos mais variados produtos e serviços disponíveis no mercado global. Essa área de conhecimento está presente na indústria do entretenimento, na engenharia, na arquitetura, no design e na comunicação. Leia as afirmativas a seguir sobre a computação gráfica.

I – A computação gráfica é o estudo e a manipulação de informações visuais e geométricas por meio da utilização de técnicas computacionais.

II – A computação gráfica é muito utilizada em projetos de arquitetura, mas pouco tem a contribuir para o design gráfico.

III – A computação gráfica está associada a outras áreas do conhecimento humano, como a matemática aplicada, a geometria e a topologia computacional e o processamento de imagens, além da visualização científica de informações.

É correto o que se afirma em

A) I, apenas.

B) III, apenas.

C) I e III, apenas.

D) II e III, apenas.

E) I, II e III.

Resposta correta: alternativa C.

Análise das afirmativas

I – Afirmativa correta.

Justificativa: a computação gráfica é uma subárea da ciência da computação e é destinada à geração, manipulação e interpretação de modelos matemáticos na forma de imagens utilizando um computador para processar esses dados. De forma mais simplificada, ela trata da área da transformação dos dados em imagem por meio do processamento computacional.

II – Afirmativa incorreta.

Justificativa: a computação gráfica é muito utilizada em projetos de arquitetura e, também, em projetos de design gráfico, conforme sugerido no próprio texto do enunciado da questão. Além disso, diversas outras áreas beneficiam-se dessa ferramenta.

III – Afirmativa correta.

Justificativa: a computação gráfica está associada a diversas áreas do conhecimento humano, o que evidencia seu caráter multidisciplinar. Áreas como engenharia, medicina, artes, meteorologia e astronomia encontram inúmeras aplicações para a computação gráfica.

Questão 2. (FCC 2018, adaptada) Considere que, em uma aplicação de localização geoespacial, um cientista da computação necessitou desenvolver uma função para executar rasterização gráfica. Com base nesse contexto, o termo rasterização significa

A) Sobrescrever um bitmap usando coordenadas geográficas.

B) Converter um vídeo de filme aéreo em imagem vetorial.

C) Transformar uma imagem jpeg em um arquivo vetorizado.

D) Converter uma imagem vetorial em uma imagem de pixels.

E) Sequenciar várias imagens de pixels para uso em vídeo, simulando movimento.

Resposta correta: alternativa D.

Análise da questão

No contexto da computação gráfica, a rasterização é um processo de amostragem que consiste na conversão de uma representação vetorial de imagem em uma matricial. Cada elemento da representação matricial é um pixel, que é a menor unidade de uma imagem digital. Logo, a expressão matricial é uma forma de ilustrar imagens digitais em uma matriz numérica de pixels, em que cada pixel é indicado por um conjunto de valores numéricos que determinam sua cor e sua intensidade. Em resumo, o processo de rasterização consiste na conversão de uma imagem vetorial em uma imagem de pixels. Diversos algoritmos, como o de Bresenham, podem ser aplicados no processo de rasterização de uma imagem.