KLS

Computação Gráfica e Processamento de Imagens

Computação Gráfica e Processamento de Imagens

André Vital Saúde

Dados Internacionais de Catalogação na Publicação (CIP)

Saúde, André Vital
S255c Computação gráfica e processamento de imagens / André Vital Saúde. – Londrina : Editora e Distribuidora Educacional S.A., 2019.
200 p.

ISBN 978-85-522-1367-3

1. Modelagem. 2. Texturização. 3. Renderização.
I. Saúde, André Vital. II. Título.

CDD 006

Thamiris Mantovani CRB-8/9491

2019
Editora e Distribuidora Educacional S.A.
Avenida Paris, 675 – Parque Residencial João Piza
CEP: 86041-100 — Londrina — PR
e-mail: editora.educacional@kroton.com.br
Homepage: http://www.kroton.com.br/

Sumário

Palavras do autor

Caro aluno, seja bem-vindo à disciplina *Computação Gráfica e Processamento de Imagens*. A cada dia observamos um incremento na capacidade dos processadores e na qualidade das telas de exibição e, consequentemente, experimentamos cada vez mais os produtos desta área da computação, que é aplicada, bastante vasta, que interage com diversas outras e que possui aplicações para as mais diversas necessidades humanas.

A computação gráfica lida com todos os problemas que envolvem processamento computacional de elementos gráficos, em diversos formatos e para diversas finalidades, incluindo captura, modelagem, síntese e processamento de imagens. Interage com disciplinas como inteligência artificial, matemática discreta e geometria para produzir tanto aplicativos para o entretenimento quanto aplicativos funcionais de auxílio à medicina e à reabilitação de pessoas com deficiência. Uma disciplina apaixonante, com desafios constantes e sempre abrindo mais seu leque de oportunidades.

Com o conjunto deste livro será possível conhecer os fundamentos das imagens, saber analisar um problema de computação gráfica e classificá-lo em suas subáreas. Você vai assimilar os conceitos das transformações geométricas e saberá aplicá-las em diferentes contextos da computação gráfica. Conhecerá a modelagem tridimensional e aprenderá a realizar a síntese de imagens. Explorará a aplicação de filtros em imagens digitais para extrair delas informações.

O livro está organizado em quatro unidades: a Unidade 1 apresenta um panorama inicial, uma introdução ao grande universo da computação gráfica, descreve as suas subáreas e apresenta os fundamentos de imagens matriciais e vetoriais. A Unidade 2 aprofunda os conhecimentos da geometria na computação gráfica, mostrando as diferenças das transformações geométricas no espaço contínuo, sobre modelos tridimensionais, e no espaço discreto, sobre imagens digitais. A Unidade 3 explora as subáreas de modelagem tridimensional e a síntese de imagens, com visualização e animação. Por fim, a Unidade 4 traz os conceitos principais da subárea de processamento de imagens, com a aplicação de filtros e segmentação de imagens.

Esta disciplina é mais uma porta que se abre para a sua vida profissional. A computação gráfica já faz parte do dia a dia das pessoas, e elas desejam sempre uma experiência melhor. Dedique-se ao estudo e você poderá vir a ser o desenvolvedor de novas soluções, trazendo novas experiências às pessoas e contribuindo para a felicidade humana.

Unidade 1

Introdução à computação gráfica

Convite ao estudo

Caro aluno, nesta primeira unidade faremos uma introdução à computação gráfica e apresentaremos os fundamentos das imagens, para que você seja capaz de analisar um problema de computação gráfica, classificá-lo e preparar um ambiente de desenvolvimento inicial para solucioná-lo. Ao final da unidade você poderá demonstrar a compreensão das áreas da computação gráfica, dos fundamentos e dos formatos das imagens fazendo um primeiro programa.

Imagine que você tenha como cliente uma empresa de grande porte, que contrata há alguns anos uma terceirizada para a elaboração de conteúdo gráfico destinado à propaganda de seus produtos. O cliente já possui, com isso, um conjunto de imagens vetoriais e de vídeos de animação, além de aplicações das imagens já realizadas em diferentes resoluções, desde chaveiros (baixa resolução) até outdoors (alta resolução), em diferentes materiais promocionais, com ou sem fundo opaco, e com aplicações de imagens vetoriais completas ou parciais (recorte).

Imagine, então, que esse cliente não está satisfeito com o serviço terceirizado e pretende internalizar todo o trabalho, visando a um trabalho mais criativo e diferenciado, principalmente no que diz respeito à exibição do conteúdo gráfico. Por esse motivo, contratou sua empresa de consultoria para ajudar a montar a equipe técnica, padronizar os formatos de imagens e vídeos e a infraestrutura de hardware e software para a equipe, além de executar um projeto piloto para servir de modelo de boas práticas de programação a serem usadas pela nova equipe. O cliente já possui uma equipe de tecnologia da informação e uma equipe de marketing, portanto não tem dificuldades em contratar gerentes de projeto ou designers gráficos. Sua dificuldade está na contratação de especialistas em processamento gráfico.

Você saberia classificar o problema do cliente com base nas subáreas da computação gráfica? Que habilidades devem apresentar os desenvolvedores a serem contratados? Com base na característica do conteúdo gráfico a ser trabalhado, quais seriam os formatos de imagem e vídeo e a infraestrutura

de hardware e software mais adequados? Você seria capaz de desenvolver um software piloto para servir de modelo para a nova equipe?

Esta unidade vai ajudá-lo a dar respostas positivas às perguntas acima e está dividida de acordo com as seguintes seções: a Seção 1.1 traz uma introdução à computação gráfica, apresentando as subdivisões da computação gráfica e os conceitos de realidade material, virtual e aumentada, além de uma breve descrição de tipos de hardware e software de processamento gráfico. A Seção 1.2 apresenta os fundamentos de imagens vetoriais e matriciais, os formatos de arquivos de imagem, estruturas de dados que representam a imagem na memória, modelos de cor, opacidade e transparência. A Seção 1.3 traz atividades práticas de processamento básico de imagens bidimensionais, vetoriais e matriciais, com a criação e desenho de retas e círculos em imagens vetoriais e matriciais.

Seção 1.1

Introdução à computação gráfica

Diálogo aberto

A computação gráfica não está presente apenas em jogos digitais e filmes de animação, mas também em todas as aplicações do nosso dia a dia que, de alguma forma, lidam com imagens e o sentido da visão do ser humano. Os smartphones e tablets são dispositivos comuns que incorporam diversas dessas aplicações, tais como: leitor de impressão digital, câmera digital com detecção de sorrisos, filtros e reconhecimento facial, interação do usuário feita por tela sensível ao toque, exibindo-se uma interface gráfica de alta resolução, além de funcionalidades de aplicativos que possam ser instalados.

Atividades, como a criação de modelos tridimensionais, animações, edição de imagens e vídeos ou a criação de um sistema de visão de um robô, utilizam o processamento gráfico. O aumento da capacidade de processamento dos dispositivos móveis, a evolução de equipamentos médicos de imageamento e até mesmo o aumento da largura de banda da Internet têm demandado um uso cada vez mais efetivo de processamento gráfico, o que exige, portanto, um aprofundamento do conhecimento em cada tipo de problema de computação gráfica.

Para permitir tal aprofundamento, a computação gráfica é dividida em subáreas que têm ganhado destaque isoladamente e têm se tornado disciplinas isoladas da computação. Precisamos, antes de tudo, conhecer essas subáreas e o que cada uma delas realiza de processamento gráfico, para, então, conhecer os tipos de hardware e software que lidam com cada uma delas.

Seu cliente, uma empresa de grande porte, contratou sua empresa de consultoria para ajudar a montar uma equipe técnica de computação gráfica e padronizar os formatos de imagens e vídeos e a infraestrutura de hardware e software para a equipe, entre outras demandas.

A primeira necessidade do seu cliente é a definição de requisitos técnicos que possam orientá-lo na contratação de sua equipe de profissionais de computação gráfica. Você deve fazer um relatório de diagnóstico do trabalho a ser realizado pela equipe, classificando-o nas subáreas da computação gráfica. O relatório deve descrever um conjunto de habilidades e experiências prévias essenciais e desejáveis para os novos membros da equipe e incluir, ainda, uma proposta de aquisição de software e hardware para a nova equipe.

Nesta seção você terá uma introdução à computação gráfica, apresentando os processamentos gráficos que existem da captura à síntese de imagens, as subdivisões da computação gráfica, uma breve descrição do hardware e do software demandado para cada subárea, e a apresentação dos conceitos de realidade virtual e aumentada. Com esse conteúdo, você será capaz de resolver a primeira situação problema. À medida que avançarmos no assunto, você poderá fazer muito mais. Vamos em frente!

Não pode faltar

Computação gráfica lida com todos os problemas que envolvem processamento computacional de elementos gráficos, em diversos formatos e para diversas finalidades (MANSSOUR; COHEN, 2006). Inicia sua atuação na captura de imagens ou na modelagem geométrica de um objeto ou cena do mundo real, criando uma representação da cena ou do objeto na memória do computador. Tal representação em memória pode ser um modelo matemático, uma imagem digital ou outro tipo de descritor de imagem que, posteriormente, passará por diferentes tipos de processamento até que possa retornar ao mundo real na forma de imagem exibida em tela, um arquivo de imagem ou imagem impressa.

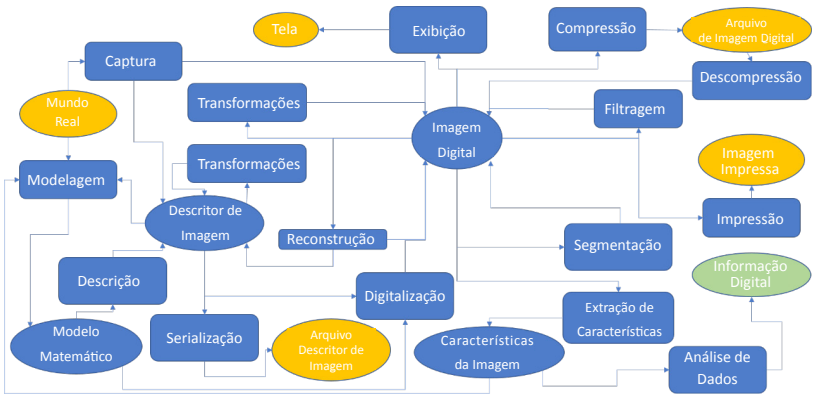
A Figura 1.1 mostra um panorama com diversos fluxos possíveis de processamento gráfico, desde a captura de imagens do mundo real. Nela estão representados, por meio de três cores diferentes, os elementos da computação gráfica, em um fluxograma construído com base na literatura (AMMERAAL; ZHANG, 2008; GONZALEZ; WOODS, 2011; HUGHES et al., 2013). Em azul são apresentados os artefatos (elipses) e processamentos (retângulos) que se encontram ou são realizados no processador ou na memória do computador. Em amarelo, artefatos que podem ser físicos ou arquivos digitais. Em verde é apresentado um artefato resultante do processamento gráfico, mas que será processado fora da computação gráfica.

Para melhor compreensão da Figura 1.1, partimos do **mundo real**. Uma cena ou objeto do mundo real é traduzido para o mundo digital por meio de um processo de **captura** ou de **modelagem**.

A modelagem é o processo de representar objetos do mundo real por meio de equações, pontos, retas, superfícies ou outras formas geométricas. O resultado da modelagem é um **modelo matemático**, também denominado modelo tridimensional. A modelagem pode ser feita manualmente, por um ser humano com habilidades artísticas (desenhista) ou com o uso de aplicativos próprios para este fim. Pode também ser realizada totalmente por processamentos gráficos, a partir de imagens capturadas do mundo real.

A **captura** é o processo de, literalmente, capturar informações do mundo real com sensores e representar a informação capturada na forma de uma imagem. Há diversos dispositivos de captura de imagem, sendo o mais conhecido a câmera fotográfica, que é um conjunto de lentes e um sensor fotossensível (CCD – *charge-coupled device*). Nem todos os dispositivos de captura, porém, são construídos com sensores de luz. Um exemplo são as máquinas de ressonância magnética, que capturam imagens aplicando um campo magnético e utilizando receptores de ondas de rádio para fazer medições no interior de um corpo vivo. Não há sensores de luz.

Figura 1.1 | Fluxos de processamento gráfico



Fonte: elaborada pelo autor.

Assimile

Nem todos os dispositivos de captura de imagem são como as câmeras fotográficas, que utilizam sensores de luz. Um exemplo são as máquinas de ressonância magnética que utilizam receptores de ondas de rádio.

Voltando à Figura 1.1, o processo de captura gera uma **imagem digital** ou um **descritor de imagem**. Um descritor de imagem é qualquer estrutura de dados que descreva objetos gráficos ou cenas, mas que não possua um conjunto de pontos em formato de matriz, como na imagem digital. A câmera fotográfica é um exemplo de dispositivo de captura que gera diretamente uma imagem digital. Já a máquina de ressonância magnética gera um conjunto de medições de rádio que não é uma imagem digital, mas sim um descritor de imagem, que depois passa por um processo de **digitalização** para gerar uma imagem digital.

Um modelo matemático pode passar por um processo de **descrição** para se gerar um descritor de imagem. Por se tratar de um descritor de objetos

gráficos, o descritor de imagens pode sofrer **transformações** geométricas como translação e rotação. Um descritor de imagem pode ainda passar ou por um processo de **serialização** para ser armazenado em um **arquivo descritor de imagem**.

A imagem digital também pode sofrer transformações e ser **exibida** em uma **tela**. Na Figura 1.1, as transformações sobre a imagem digital foram representadas separadas das transformações sobre descritores de imagem por possuírem características diferentes. Os demais processamentos da imagem digital ilustrados na Figura 1.1 serão descritos nas subáreas da computação gráfica.

As subáreas da computação gráfica

A computação gráfica pode ser dividida em subáreas com base nos objetivos das aplicações, considerando o que se tem como entrada e o que se busca obter como saída em um fluxo de processamento gráfico. Adotaremos a divisão em quatro subáreas: modelagem tridimensional, síntese de imagens, visão computacional e processamento de imagens (GONZALEZ; WOODS, 2011; HUGHES et al., 2013; MANSSOUR; COHEN, 2006).

Modelagem tridimensional

A modelagem tridimensional é a subárea da computação gráfica cujo objetivo é a criação de modelos matemáticos tridimensionais de objetos ou cenas. A modelagem tridimensional pode ser considerada parte da síntese de imagens, mas o mercado hoje trabalha com equipes dedicadas exclusivamente à modelagem tridimensional, por isso a separação.

A Figura 1.2 mostra os artefatos e processamentos relacionados à subárea de modelagem tridimensional. Todos os elementos desta figura foram extraídos do panorama da Figura 1.1.

Como já mencionado, a modelagem pode ser feita à mão por um ser humano com dons artísticos, porém, para se obterem modelos tridimensionais mais realísticos, o melhor caminho é construir o modelo computacionalmente, com base em imagens capturadas do mundo real. Existem dispositivos específicos para a captura de cenas e objetos do mundo real para se criar modelos tridimensionais. Alguns desses dispositivos são os scanner 3D a laser.

É possível também criar modelos 3D a partir de imagens 2D capturadas por câmeras digitais. Nesse caso, diversas câmeras são posicionadas ao redor do objeto, e posteriormente é feito um alinhamento das imagens obtidas, num processo denominado **reconstrução**. Com o uso de câmeras, é possível também fazer a **extração de características** adicionais do objeto, como cor e textura. Modelos tridimensionais de seres humanos são obtidos por um

processo de captura combinando scanner 3D e fotografias digitais, como é o caso dos seres humanos em jogos digitais de futebol.

Pesquise mais

Exemplos de capturas de imagens para o processo de modelagem tridimensional de jogadores de futebol.

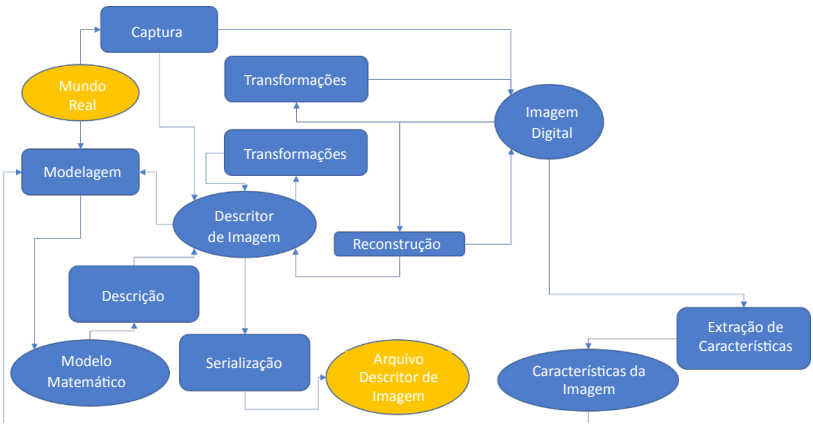
Modelagem 3D a partir de câmeras digitais:

LIVERPOOL FC. **FIFA 14 | Liverpool 3D Team Head Scan**, 2013.

Scanner 3D laser:

VIRTUMAKE. **3D Scan - Professional vs Hobby - Artec Eva, Carmine** 1.09, 2013.

Figura 1.2 | Artefatos e processamentos relacionados à modelagem tridimensional



Fonte: elaborada pelo autor.

Síntese de imagens

A síntese de imagens é a subárea da computação gráfica que visa à produção de imagens sintéticas a partir de modelos matemáticos e transformações do modelo. É também a subárea que trata da criação de interfaces gráficas de usuário e animações (AMMERAAL; ZHANG, 2008;

HUGHES et al., 2013).

A Figura 1.3 mostra os artefatos e os processamentos relacionados à subárea de síntese de imagens. Todos os elementos da Figura 1.3 foram extraídos do panorama da Figura 1.1.

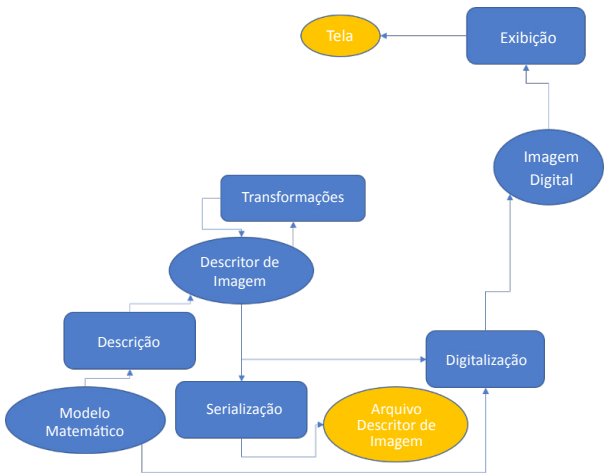
A síntese de imagens utiliza modelos matemáticos já existentes, produzidos pela subárea de modelagem tridimensional, e a imagem digital é o

produto final resultante de processamentos realizados sobre modelos e descritores de imagem. Um modelo matemático se diferencia do descritor de imagem porque pode incluir equações matemáticas enquanto o descritor de imagem é um conjunto de pontos organizados de forma a descrever uma superfície tridimensional. Diferentes descritores de imagem podem ser gerados a partir de um único modelo matemático, principalmente no que diz respeito à resolução.

Exemplificando

Nos jogos digitais, a equipe de modelagem tridimensional busca construir o modelo mais realístico possível. A equipe de síntese de imagem vai criar, a partir desse modelo, um descritor de imagem para cada tipo de equipamento onde o jogo será processado. Para um console de última geração, será gerado um descritor de alta resolução, com um grande número de pontos descrevendo a superfície do objeto. Para o console da geração anterior será gerado um descritor de mais baixa resolução, com um número menor de pontos descrevendo a superfície do objeto.

Figura 1.3 | Artefatos e processamentos relacionados à síntese de imagens



Fonte: elaborada pelo autor.

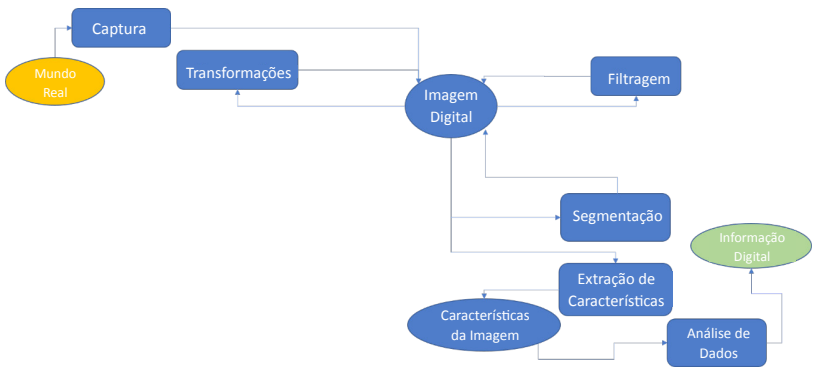
A síntese de imagens é fortemente embasada na geometria. Transformações e digitalização são processamentos que envolvem geometria. Movimentações de pontos de observação e fontes de luz são realizadas por transformações geométricas em 3D. A digitalização é realizada por transformações geométricas que levam do espaço 3D para o espaço 2D, a tela de exibição (AMMERAAL; ZHANG, 2008).

Visão computacional

A visão computacional é a subárea da computação gráfica que visa a extração de informações úteis a partir de imagens capturadas do mundo real. O termo visão computacional é algumas vezes apresentado como sinônimo de análise de imagens, no entanto, para fins desta classificação, a análise de imagens é uma parte da visão computacional, que inicia sua atuação a partir de uma imagem digital já existente, enquanto a visão computacional atua de forma mais ampla, partindo do processo de captura da imagem digital (GONZALEZ; WOODS, 2011).

A Figura 1.4, cujos elementos foram extraídos do panorama da Figura 1.1, mostra os artefatos e processamentos relacionados à subárea de visão computacional.

Figura 1.4 | Artefatos e processamentos relacionados à visão computacional



Fonte: elaborada pelo autor.

A captura da imagem é uma etapa crítica na visão computacional, assim como na modelagem tridimensional. A diferença é que enquanto a modelagem tridimensional busca capturar imagens em diversos ângulos, com alta resolução, para facilitar o processo de reconstrução 3D, a visão computacional busca capturar imagens com nitidez suficiente para se analisarem objetos de interesse. Algumas aplicações de visão computacional exigem que a captura da imagem seja feita em uma câmara fechada, iluminada artificialmente, como algumas aplicações de controle de qualidade em uma indústria. Outras aplicações exigem capturas de

baixa resolução, para o processamento em tempo real, como algumas aplicações de vídeo vigilância.

A análise de imagens é o conjunto de artefatos e processamentos para extrair **informação digital** da imagem digital. O objetivo é a **extração de características** e **análise de dados** extraídos da imagem. Para isso, a imagem digital pode passar por pré-processamento, visando facilitar a extração de características. Pode ser uma **filtragem**, com o objetivo de realçar características, ou uma **segmentação**, com o objetivo de separar objetos de interesse.

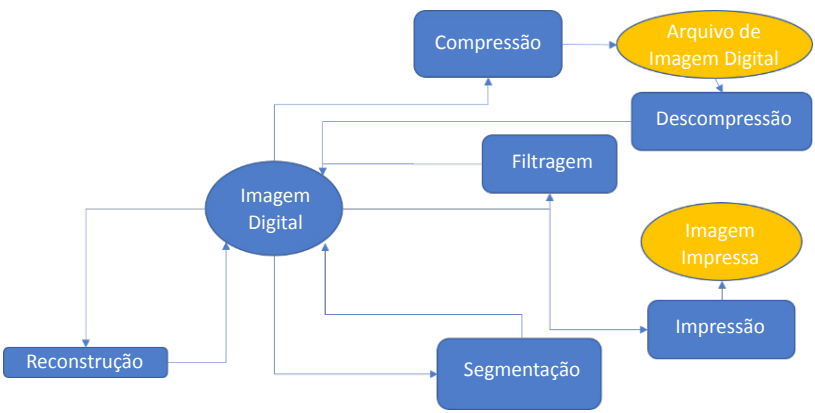
A visão computacional interage com diversas outras disciplinas da Ciência da Computação. Os processos de filtragem, segmentação, extração de características e análise de dados podem envolver disciplinas como inteligência artificial, estatística, cálculo diferencial e integral, matemática discreta, otimização, teoria de conjuntos, dentre outras.

Processamento de imagens

O processamento de imagens é a subárea da computação gráfica que visa o tratamento e alterações de imagens digitais. O objetivo do processamento de imagens é fazer modificações em imagens digitais para se obterem outras imagens digitais (GONZALEZ; WOODS, 2011).

A Figura 1.5, cujos elementos foram extraídos do panorama da Figura 1.1, mostra os artefatos e processamentos relacionados à subárea de processamento de imagens.

Figura 1.5 | Artefatos e processamentos relacionados ao processamento de imagens



Fonte: elaborada pelo autor.

O processamento de imagens é a subárea que engloba qualquer aplicação de edição de imagens. O processamento gráfico mais comum é a filtragem, mas a segmentação e a reconstrução também são usadas no processamento de imagens. Se um objeto for removido de uma imagem digital, por exemplo, uma lacuna será criada, e a imagem precisará passar por um processo de reconstrução. Se for relevante aplicar um filtro apenas sobre um objeto de interesse da imagem, é preciso antes fazer uma segmentação.

Ainda existem os processos de **compressão** e **descompressão** de **arquivos de imagem digital** ou vídeo digital. Os arquivos de imagem e vídeo são comprimidos para ocupar menos espaço de armazenamento ou demandar menos banda de transmissão via internet. A preparação de imagens para **impressão** também envolve processamento de imagens. Para imprimir uma imagem de baixa resolução a ser aplicada em um outdoor, por exemplo, é preciso fazer uma ampliação.

Reflita

Seria mais coerente se as subáreas de modelagem tridimensional e síntese de imagens fossem consideradas uma única subárea? Por outro lado, a análise de imagens deveria ser considerada uma subárea destacada da visão computacional? O que você pensa?

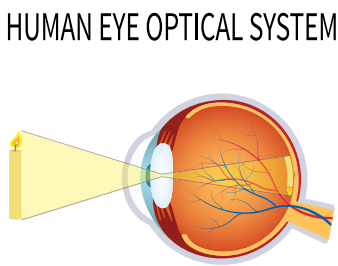
A interação e a computação gráfica

O estudo da interação humana com a Computação Gráfica começa pela compreensão do sistema visual. O olho humano é o órgão responsável pelo sentido da visão e funciona como um sistema de lentes, como mostra a Figura 1.6.

A córnea faz o papel de lente, e a retina, o de aparato onde é projetado o objeto visualizado. A visão é o sentido que permite a percepção do mundo real na forma de imagens, e é também o sentido que permite a percepção de imagens sintéticas, produzidas pelo computador (VAN DER GRAAFF, 2003).

O mundo material, sem imagens sintéticas, é a **realidade material**. Com a computação gráfica é possível criar um ambiente totalmente virtual onde um ser humano se sente imerso em outra realidade, diferente da realidade material. Ao isolar o campo visual humano da realidade material, cria-se um ambiente de **realidade virtual**, o qual permite não só que o indivíduo esteja imerso em outro espaço por meio da sua visão, mas também que ele interaja com esse espaço, permitindo que se crie um ambiente propício para diversos tipos de simuladores.

Figura 1.6 | Sistema ótico do olho humano



Fonte: iStock.

Os conceitos da realidade virtual podem ser usados para a criação de uma **realidade aumentada**. A realidade aumentada é a sobreposição de elementos gráficos de realidade virtual sobre a imagem capturada do mundo real. Ao abrir o aplicativo de câmera fotográfica de um smartphone, o usuário enxerga na tela uma imagem do mundo real que será capturada. Se, sobre essa imagem, for sobreposto um elemento gráfico virtual, que não esteja presente no mundo material, estará sendo criado um ambiente de realidade aumentada (SHERMAN; CRAIG, 2003).

Há uma vasta gama de hardware e software destinados ao universo da computação gráfica. O hardware mais utilizado para processamento é o próprio microprocessador de uso geral, acompanhado de uma unidade de processamento gráfico (GPU – *Graphics Processing Unit*). O processo de captura exige hardware específico, como as câmeras digitais e dispositivos que usam métodos de captura por ondas de rádio, laser, ultrassom, dentre outros. Para realidade virtual e aumentada, existem máscaras e óculos específicos.

Em se tratando de software, há dois conjuntos a se destacar: as bibliotecas ou APIs (*application programming interfaces*) utilizadas para o desenvolvimento de aplicações e os aplicativos interativos, que permitem a execução de processamento gráfico de forma interativa. Existem APIs para processamento de imagens, geometria, jogos, animação, visualização e diversas outras, fazendo com que o desenvolvimento de aplicativos de computação gráfica se torne um grande projeto de reuso de software.

Dentre os aplicativos interativos, encontram-se os aplicativos destinados à modelagem tridimensional e síntese de imagens, com ferramentas de modelagem, animação e visualização, aplicativos destinados à edição de imagem, com ferramentas de filtragem, segmentação, compressão, dentre outras, e os aplicativos de aplicação específica, como os aplicativos de visualização de imagens médicas, processamento de imagens de satélite, entre outros.

Chegamos ao final desta seção. Um panorama da Computação Gráfica e Processamento de Imagens foi apresentado. As próximas seções apresentarão conteúdos mais técnicos e aplicados. Continue engajado nos estudos!

Sem medo de errar

Retomando o contexto com que iniciamos esta unidade, seu cliente, uma empresa de grande porte, contratou sua empresa de consultoria para ajudar a montar uma equipe técnica de computação gráfica e padronizar os formatos de imagens e vídeos e a infraestrutura de hardware e software para a equipe, entre outras demandas.

Sua empresa precisa definir requisitos técnicos para orientar seu cliente na contratação de sua equipe de profissionais de computação gráfica. Lembrando que a atividade da equipe a ser contratada será a elaboração de conteúdo gráfico e que o cliente já possui um conjunto de imagens vetoriais e de vídeos de animação, além de aplicações das imagens já realizadas em diferentes resoluções, com imagens vetoriais completas ou parciais (recorte).

Um ponto a se destacar é que o seu cliente busca internalizar a equipe de produção de tal tipo de conteúdo porque não está satisfeito com o serviço terceirizado e, ao internalizar todo o processo, espera um trabalho mais criativo e **diferenciado**, principalmente no que diz respeito à exibição do conteúdo gráfico.

Essa expectativa de **diferenciação** na exibição do conteúdo gráfico é que exige uma equipe técnica de desenvolvedores especialistas em computação gráfica. Uma equipe de designers gráficos pode ser capaz de agregar criatividade ao trabalho, mas o que se espera de um designer é que ele saiba operar aplicativos de software de mercado, e não que saiba obter mais do que esses aplicativos permitem.

Em seu relatório, você deve fazer um diagnóstico do trabalho a ser realizado, classificando-o nas subáreas da computação gráfica. Sobre este ponto você deve observar que a diferenciação é esperada na exibição do conteúdo gráfico. O que é exibida é uma imagem digital, obtida por digitalização de um descritor de imagem vetorial, um processamento típico da subárea de síntese de imagens. Sobre a imagem digital a ser exibida podem ser aplicados filtros ou transformações (pense na exibição de uma imagem de baixa resolução em um outdoor), que são atividades de processamento de imagens. Não faz parte do trabalho uma diferenciação no processo de modelagem tridimensional, nem há demanda por extração de informações de imagens digitais. Essas informações definem o tipo de especialização demandada. Seu cliente precisa de uma equipe que domine ferramentas de síntese de imagens e processamento de imagens.

No seu relatório, é interessante apresentar de forma sucinta o conjunto de artefatos e processamentos da Figura 1.1, destacando os processamentos que serão realizados pela equipe contratada, como forma de justificar os requisitos técnicos exigidos na contratação.

Este problema permite demonstrar a compreensão das áreas da computação gráfica.

Avançando na prática

Reconhecimento facial em vídeo vigilância

Descrição da situação-problema

Sua empresa foi contratada pelo Estado para implantar um sistema de câmeras em estações de metrô e desenvolver uma aplicação capaz de reconhecer, nas imagens das câmeras instaladas, faces de criminosos procurados pela polícia. A solução deverá capturar imagens do metrô, detectar faces nas imagens e, de cada face detectada, extrair um conjunto de características. Cada conjunto de características corresponde a uma face detectada, e precisa ser comparado a outros conjuntos de características que compõem um banco de dados de criminosos. As características extraídas das faces detectadas no metrô devem ser as mesmas características previamente extraídas das faces do banco de dados. Faz parte da solução também a construção do banco de dados a partir de imagens de criminosos.

Você foi alocado como gerente deste projeto e precisa, então, alocar sua equipe. Para tanto, é preciso fazer uma descrição do problema a ser abordado, indicando que subárea da computação gráfica está envolvida e que processamentos são críticos.

Resolução da situação-problema

Aplicações de vídeo vigilância em geral são aplicações de visão computacional e, neste caso, não é diferente, entenda por quê.

Vamos destacar, nos requisitos técnicos da solução requerida, os processamentos gráficos necessários. A solução deverá **capturar** imagens, **detectar** faces e **extrair características**, tanto para a construção do banco de dados de criminosos quanto para a comparação das faces detectadas no metrô com as faces do banco de dados.

Você deve observar que há dois pontos críticos para solucionar o problema. O primeiro é a extração de características das imagens capturadas, que é o que permitirá comparar as faces detectadas com as faces do banco de dados. O segundo ponto crítico é a captura de imagens, pois são necessárias imagens que mostrem faces com nitidez para que as características possam ser extraídas.

Tanto a subárea de modelagem tridimensional (Figura 1.2) quanto a de visão computacional (Figura 1.4) fazem captura de imagens e extração de características. A diferença é que a primeira utiliza esses processamentos

para construir modelos matemáticos e a segunda os utiliza para extrair informações da imagem. Com isso, pode-se concluir que se trata de um problema de visão computacional.

Finalmente, a equipe do projeto precisa ser uma equipe com experiência em todo o processo de uma aplicação de visão computacional.

Faça valer a pena

1. Um professor desenvolveu um aplicativo para celular que permite que o usuário registre ocorrências de buracos nas ruas da cidade. O usuário tira uma foto do buraco e o aplicativo registra a geolocalização do buraco fotografado. O aplicativo envia a foto para um servidor, gerando uma ordem de serviço para a prefeitura local. Para evitar que imagens quaisquer sejam submetidas ao servidor, foi desenvolvido um módulo que classifica a imagem como uma imagem que contém ou não um buraco na rua.

Assinale a alternativa que diz o tipo de processamento usado pelo aplicativo para auxiliar a classificação da imagem como uma imagem que contém ou não um buraco na rua.
a) Compressão.
b) Captura.
c) Extração de características.
d) Reconstrução.
e) Modelagem tridimensional.

2. Um neurocirurgião utiliza imagens de ressonância magnética da cabeça de pacientes epilépticos para planejar a cirurgia. Para isso, ele usa uma ferramenta interativa de visualização tridimensional do cérebro a ser operado e marca a região de interesse. Com base na região do cérebro delineada, a própria ferramenta já desenha, sobre o crânio, na imagem do paciente, a posição menos invasiva para se realizar a incisão. A imagem de ressonância magnética é uma imagem digital, representando um volume.

Assinale a alternativa que cita as duas subáreas da computação gráfica a que esta ferramenta melhor se enquadra.
a) Processamento de imagens e visão computacional.
b) Visão computacional e modelagem tridimensional.
c) Visão computacional e síntese de imagens.
d) Processamento de imagens e síntese de imagens.
e) Modelagem tridimensional e processamento de imagens .

3. A realidade aumentada é a sobreposição de objetos virtuais sobre imagens digitais capturadas do mundo real. A realidade aumentada usa diversos tipos de processamento gráfico, de diferentes subáreas. Um exemplo de realidade aumentada seria um aplicativo de celular que mostre na imagem capturada pela câmera, personagens virtuais de um jogo digital, com base na geolocalização da câmera.

Assinale a alternativa correta com relação aos processamentos gráficos utilizados para a realidade aumentada implementada por esse aplicativo.
a) São usadas captura e transformações da imagem digital para transportar a imagem do mundo real para o ambiente de imersão virtual.
b) É usada a reconstrução 3D, para construir um modelo tridimensional do objeto virtual a partir de imagens do objeto real capturadas a partir de diferentes ângulos.
c) São usadas a descompressão e transformações da imagem digital capturada do mundo real, que será sobreposta ao modelo do objeto virtual.
d) São usadas filtragem e segmentação da imagem digital da câmera para extrair informações sobre como posicionar o objeto virtual sobre a imagem real.
e) Será usada a digitalização de modelos para gerar uma imagem bidimensional do objeto virtual que será sobreposta à imagem capturada pela câmera.

Seção 1.2

CGPI: fundamentos de imagens

Diálogo aberto

Caro aluno, nesta seção vamos tratar de assuntos técnicos. Conhecer as subáreas da computação gráfica é essencial para que se possa reconhecer o que deve ser feito para cada tipo de problema a ser resolvido. Para ser capaz de resolver o problema por completo, é preciso mais.

Diferente do designer gráfico, que utiliza aplicativos interativos para fazer desenhos e animações, o desenvolvedor especialista em computação gráfica deve ser capaz de construir soluções não disponibilizadas por tais aplicativos, seja estendendo-os, seja construindo novos. A resolução de problemas de computação gráfica exige programação.

Atualmente há diversas APIs de computação gráfica e processamento de imagem disponíveis para diferentes linguagens, e o especialista deve ser capaz não só de utilizar, como também de estender ou reescrever qualquer uma delas. Até mesmo para reutilizar de maneira eficaz uma API existente, você deve compreender sua documentação e, portanto, conhecer os fundamentos de cada função ou método por ela implementado.

Vamos retomar o contexto desta unidade. Seu cliente contratou sua empresa de consultoria para ajudar a montar uma equipe técnica de computação gráfica. Solicitou ainda que sejam definidos padrões de programação, formatos de imagens e vídeos, além da infraestrutura de hardware e software para a equipe.

Você já definiu requisitos técnicos para orientá-lo na contratação de sua equipe de profissionais de computação gráfica. Falta definir os padrões solicitados e criar um projeto piloto para servir de modelo de boas práticas de programação para a nova equipe.

Você deve entregar um relatório contendo um levantamento do conteúdo gráfico já disponível e das linguagens de programação já utilizadas, hoje, pela equipe de tecnologia de informação do seu cliente. No mesmo relatório, você deve definir um conjunto de APIs a serem adotadas pela nova equipe de processamento gráfico, assim como as primeiras estruturas de dados para representar imagens matriciais e vetoriais, apresentando exemplos de código fonte.

Esta seção traz um conteúdo técnico dos fundamentos de imagens, os conceitos de imagens vetoriais e matriciais e os formatos de ambos os tipos de imagens tanto em arquivo quanto em memória. Você conhecerá também

diversos modelos de cor, opacidade e transparência, com exemplos de código para a definição e manipulação de imagens usando linguagem de programação.

Não pode faltar

Caro aluno, nesta seção vamos explorar os fundamentos de imagens. De acordo com o Dicionário Online de Português, uma imagem é a "representação de uma pessoa ou uma coisa pela pintura, a escultura, o desenho etc" (IMAGEM, 2018, [s.p.]). Na computação, uma das bases para tal representação é a geometria.

Comecemos por representar as coisas pela geometria plana, que também é conhecida como geometria Euclidiana, em homenagem a Euclides de Alexandria, matemático grego (CAMARGO; BOULOS, 2005). Com ela, somos capazes de representar as coisas pelo desenho. A geometria plana é o que define pontos, retas, círculos e outras formas geométricas que podem ser representadas sobre o plano. Um círculo de centro no ponto *(a, b)* e raio *r*, é definido pela equação $(x-a)^2+(y-b)^2=r^2$, mas pode ser **descrito** simplesmente pelas duas coordenadas do seu centro e seu raio. Conhecendo a equação do círculo, seu centro e seu raio, um programa de computador pode desenhar esse círculo sobre o plano, assim como uma pessoa o faria utilizando um compasso.

Note que a descrição de um círculo pela informação das coordenadas do seu centro e seu raio não é uma imagem, mas apenas uma descrição. Imagem, de fato, só existirá se o círculo for desenhado sobre o plano. Entretanto, uma imagem pode ser totalmente descrita por equações vetoriais, ou, melhor dizendo, por meio de descritores de imagens, para ser desenhada apenas quando necessário. Imagens descritas com base neste princípio são chamadas **imagens vetoriais** (HUGHES et al., 2013).

O descritor de um círculo em uma imagem vetorial pode ainda ter atributos de cor e transparência e não estar limitado ao plano. A geometria euclidiana já foi estendida para mais do que as duas dimensões da geometria plana. Imagens vetoriais, portanto, podem representar formas geométricas multidimensionais com cores e transparência. Mesmo assim, são apenas equações matemáticas que descrevem a imagem e, para serem visualizadas, precisam ser digitalizadas, ou seja, desenhadas sobre o plano.

Assimile

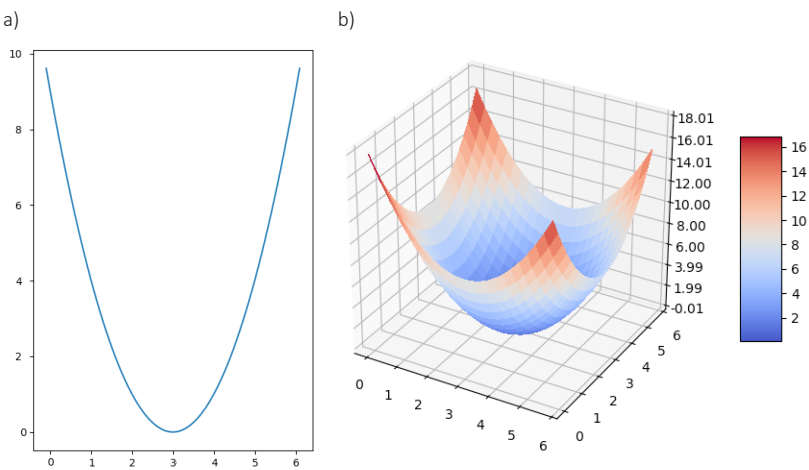
O que chamamos de imagem vetorial é um conjunto de equações matemáticas (descritores) de formas geométricas. Não é possível visualizar as equações matemáticas. Para se visualizar uma imagem vetorial é preciso desenhar as formas descritas por ela sobre o plano, ou seja, fazer sua digitalização, gerando uma imagem digital.

Imagens vetoriais são uma representação sintética da realidade. As fotografias ou as versões digitalizadas de imagens vetoriais são imagens digitais. Para compreender a imagem digital, é preciso relembrar outros conceitos: as funções matemáticas.

Uma função matemática é um mapeamento de elementos de um conjunto, o domínio, para elementos de outro conjunto, o contradomínio. Os elementos do contradomínio, que são mapeados por elementos do domínio, constituem a **imagem** da função. Para definirmos que f é uma função de domínio A e contradomínio B, utilizamos a notação $f : A \rightarrow B$.

A Figura 1.7 (a) ilustra uma função $f : \mathbb{R} \rightarrow \mathbb{R}$. O domínio da função f é o conjunto dos números reais, ou seja, todos os possíveis valores do eixo x (abcissas). O contradomínio de f é também o conjunto dos números reais, ou seja, todos os possíveis valores do eixo y (ordenadas). A imagem de f é o mapeamento representado pela curva $f(x) = (x-3)^2$, uma parábola. Pela definição do conjunto dos números reais, existem infinitos valores num intervalo entre dois valores quaisquer de x. Portanto dizemos que f é uma função **contínua** (SCHEINERMAN, 2011).

Figura 1.7 | Funções contínuas: a parábola $f(x) = (x-3)^2$ em (a) e o paraboloide $f(x, y) = (x-3)^2 + (y-3)^2$ em (b)



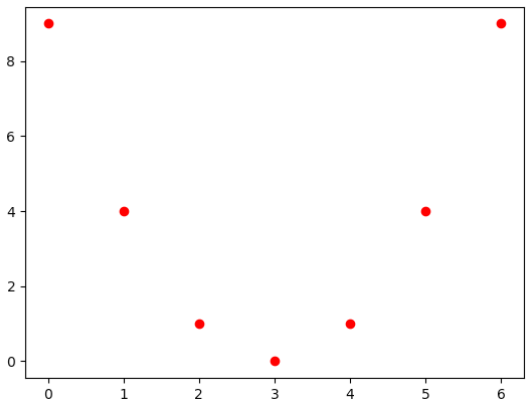
Fonte: elaborada pelo autor.

Podemos criar também uma função contínua em duas dimensões. A Figura 1.7 (b) ilustra uma função $g : \mathbb{R}^2 \rightarrow \mathbb{R}$. O domínio de g são duas dimensões (eixos x e y), e o contradomínio (eixo z), uma dimensão de números reais. A imagem de g é o paraboloide $g(x, y) = (x-3)^2 + (y-3)^2$.

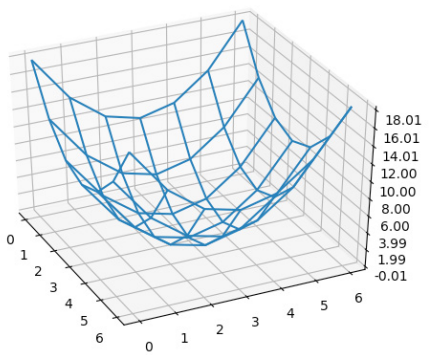
Vamos agora criar uma função h cujos domínio e contradomínio sejam o conjunto dos números inteiros, ou seja, $h:\mathbb{Z}\rightarrow\mathbb{Z}$. A função h está representada na Figura 1.8. Trata-se de uma amostragem da parábola da função f apresentada na Figura 1.7 (a). A Figura 1.8 (b) representa a função $i:\mathbb{Z}^2\rightarrow\mathbb{Z}$, uma amostragem do paraboloide da função g apresentada na Figura 1.7 (b).

Figura 1.8 | Funções discretas: a parábola $h(x)=(x-3)^2$ em (a) e o paraboloide $i(x,y)=(x-3)^2+(y-3)^2$ em (b), com os pontos conectados em um ***wireframe*** para melhor visualização

a)



b)



Fonte: elaborada pelo autor.

A função que mapeia um domínio nos números inteiros para um contradomínio também nos números inteiros é uma função **discreta** (SCHEINERMAN, 2011). Se o domínio da imagem i for um intervalo de inteiros em cada dimensão, i pode ser representada numericamente por uma

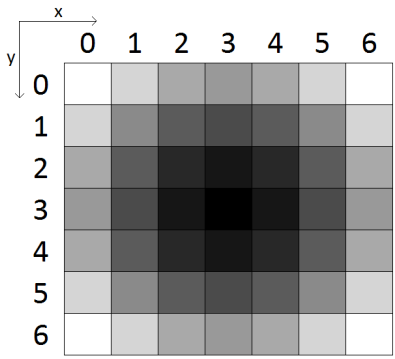
matriz bidimensional, na qual as colunas representam a coordenada *x,* e as linhas representam a coordenada *y*, como mostra a Figura 1.9 (a). Podemos também representar os números da matriz como tons de cinza, partindo do preto, para o valor 0, até o branco, para o valor 18, como na Figura 1.9 (b). A imagem representada por uma matriz é chamada de imagem **matricial**, **bitmap** ou, simplesmente, **imagem digital**. Cada elemento da imagem digital bidimensional é um pixel (acrônimo do inglês *picture element*) (HUGHES et al., 2013).

Figura 1.9 | Representações: matricial (a) e tons de cinza (b) do paraboloide $i(x, y) = (x-3)^2 + (y-3)^2$

a)

y \ x	0	1	2	3	4	5	6
0	18	13	10	9	10	13	18
1	13	8	5	4	5	8	13
2	10	5	2	1	2	5	10
3	9	4	1	0	1	4	9
4	10	5	2	1	2	5	10
5	13	8	5	4	5	8	13
6	18	13	10	9	10	13	18

b)



Fonte: elaborada pelo autor.

Reflita

Coloque-se no contexto em que você foi encarregado de montar uma equipe de computação gráfica para trabalhar no desenvolvimento de conteúdo gráfico para a propaganda dos produtos de uma empresa. Essa equipe vai trabalhar principalmente com imagens vetoriais ou imagens matriciais?

Formatos de imagem tipo bitmap e vetoriais

Imagens digitais (do tipo bitmap ou matriciais) são armazenadas em arquivos que contêm todos os pixels. Os arquivos do tipo BMP são arquivos sem compactação. Se cada pixel ocupar 1 byte (imagem em tons de cinza), uma imagem de 12 megapixels (12MP) ocupará 12MB de um arquivo BMP, e se cada pixel ocupar 4 bytes (imagem colorida com transparência), o arquivo terá 48MB.

Para evitar arquivos muito grandes, as imagens digitais são comprimidas. Exemplos de imagens comprimidas são TIFF, JPEG e PNG. Para um

software utilizar uma imagem BMP, basta copiar sequencialmente os bytes do arquivo para a memória. Para utilizar uma imagem comprimida, é preciso ler todo o arquivo e fazer a descompressão para obter a imagem em bitmap na memória. Diferentes formatos de arquivos comprimidos podem usar diferentes algoritmos de compressão.

Na memória, uma imagem digital nada mais é do que um vetor unidimensional de bytes. Os elementos da matriz bidimensional são dispostos no vetor segundo uma ordem *raster*. A ordem *raster* tem como primeiro elemento o ponto de coordenada (0,0). Todos os elementos da linha 0 são adicionados ao vetor unidimensional, em seguida os elementos da linha 1, e assim por diante. O elemento da linha y, coluna x, se encontrará na posição $y \times xs + x$, onde xs é o tamanho de cada linha. Para a Figura 1.9 (a), os elementos no vetor estarão na sequência {18, 13, 10, 9, 10, 13, 18, 13, 8, 5, 4, 5, 8, 13, 10, 5, 2, 1, **2**, 5, 10, ...}. O elemento da linha 2, coluna 4, é o elemento na posição $2 \times 7 + 4 = 18$, destacado em negrito no vetor unidimensional.

Imagens vetoriais são armazenadas em arquivos com descritores de formas. Não são arquivos visualizáveis. Exemplos de tipos de arquivos vetoriais são SVG, CDR e DWG, mas há diversos outros tipos de arquivos (MURRAY; VANWRYPER, 1996). Para que um software possa utilizar uma imagem vetorial, precisa implementar o formato de armazenamento dos descritores e carregar na memória os descritores em uma estrutura de dados apropriada.

Para exibir a imagem, precisará digitalizá-la, desenhando as formas em uma imagem digital.

O trecho de código a seguir mostra um exemplo de estrutura de dados capaz de descrever uma imagem vetorial na linguagem Python, e a Figura 1.10 exibe a imagem.

```
class Line:
  def __init__(self, x1, y1, x2, y2, color):
    self.start = (x1,y1)
    self.end = (x2,y2)
    self.color = color
class VImage:
  def __init__(self):
    self.descriptor = [] # lista vazia de descritores
```

```
    def addDescriptor(self, d):
        self.descriptor.append(d)
i = VImage()
i.addDescriptor(Line(2,2,4,4,'black'))
i.addDescriptor(Line(4,2,2,4,'red'))
```

Note que a linha vermelha está por cima da linha preta porque foi desenhada depois.

Figura 1.10 | Visualização da imagem digital correspondente ao código fonte acima

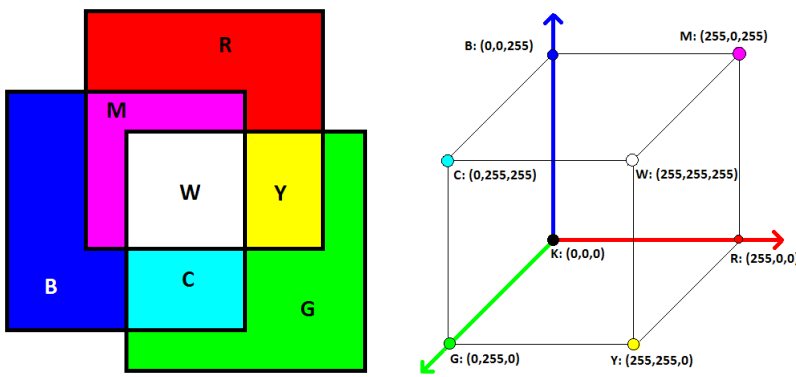
Fonte: elaborada pelo autor.

Modelos de cor e transparência

Até o momento falamos de imagens digitais em tons de cinza. Imagens coloridas usam modelos de cor. O modelo de cor mais conhecido é o RGB (*red*, *green*, *blue*), ilustrado na Figura 1.11.

Figura 1.11 | Modelo RGB: combinações de cores primárias (a) e o cubo de cores (b)

a) b)



Fonte: elaborada pelo autor.

O modelo RGB surgiu a partir de estudos de composição de luzes, que demonstraram que a partir de apenas três cores primárias é possível, ajustando-se a intensidade de cada uma delas, obter todas as outras cores do espectro da luz visível. Os estudos também mostraram que as três cores primárias que

apresentam as melhores combinações para gerar outras cores são as cores vermelho (Red), verde (Green) e azul (Blue), o que deu origem ao modelo RGB.

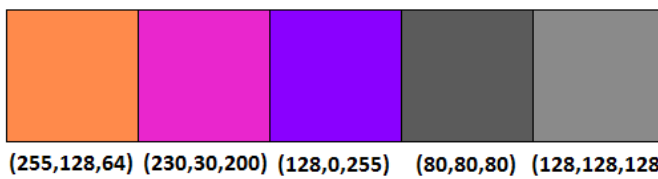
O modelo RGB é um modelo de cor **aditivo**. A combinação das cores primárias gera as cores secundárias, como mostra a Figura 1.11 (a). O verde adicionado ao vermelho produz o amarelo (Y), o vermelho adicionado ao azul produz o magenta (M) e o verde adicionado ao azul produz o ciano (C). Quando adicionados verde, vermelho e azul, obtém-se o branco (W), que é a composição de todas as cores. Uma cor é identificada por uma tupla *(r,g,b)*. Com $r = 0$, não há contribuição da cor vermelha, e com $r = 255$, há o máximo de contribuição da cor vermelha. O mesmo vale para as cores verde (g) e azul (b). Uma cor é, portanto, um ponto no interior do cubo de cores da Figura 1.11 (b), cujos vértices representam as cores primárias (R, G e B), as cores secundárias (C, M e Y), a ausência de cor (K) e a composição de todas as cores (W).

Exemplificando

O cubo RGB da Figura 1.11 (b) é amplamente utilizado no cotidiano de quem cria qualquer conteúdo gráfico. Para se definir a cor de fundo de uma página web, por exemplo, utiliza-se a seguinte linha em HTML5:

```
<h1 style="background-color:rgb(r,g,b);">...</h1>
```

onde (*r,g,b*) são os valores do cubo RGB. Note que quando r=g=b, temos um tom de cinza. A diagonal do cubo RGB (segmento que liga o ponto K ao ponto W, na Figura 1.11 (b)) contém os tons de cinza. Veja alguns exemplos de cores:

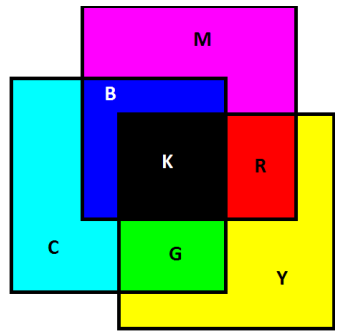


Além das três componentes de cor (R, G e B), uma imagem digital pode ter uma quarta componente: a **transparência**. A componente de transparência permite a sobreposição de uma imagem sobre um fundo permitindo que o fundo não seja totalmente coberto pela imagem sobreposta. A transparência é o oposto da **opacidade**. Uma imagem totalmente opaca cobrirá completamente o fundo, enquanto uma imagem

com algum grau de transparência exibirá tanto o fundo sobreposto quanto a imagem que o sobrepõe, como um tecido transparente.

Por ser um modelo aditivo, o modelo RGB é usado em situações em que a imagem deve ser exibida em uma tela e que, para tanto, as cores são obtidas pela luz. A ausência de luz se torna a cor preta. Isso serve para TVs, monitores, projetores, etc. Quando as cores precisam ser impressas no papel, o papel é branco e, para ser lido, precisa estar iluminado. Para a impressão, portanto, é utilizado um modelo complementar ao RGB, o modelo CMY, que é um modelo de cor **subtrativo**. As cores básicas do CMY são as cores secundárias do RGB: o ciano (C), o magenta (M) e o amarelo (Y). As combinações dessas cores básicas são mostradas na Figura 1.12.

Figura 1.12 | Modelo CMY



Fonte: elaborada pelo autor.

A combinação de todas as cores secundárias produz a ausência de cores, que é o preto (K). A combinação das cores secundárias, duas a duas, gera as cores primárias. O amarelo combinado com o ciano produz o verde, o ciano combinado com o magenta produz o azul, e o magenta combinado com o amarelo produz o vermelho. O modelo CMY é usado nas impressoras e, por esse motivo, os cartuchos de pigmentos são ciano, magenta ou amarelos. É comum, porém, que as impressoras possuam um quarto cartucho, com o pigmento preto, formando o modelo CMYK. Como a obtenção da cor preta pela mistura das cores secundárias requer grandes quantidades de pigmentos coloridos, a presença do pigmento preto gera economia, visto que a cor preta é muito utilizada nas impressões.

Pesquise mais

O modelo RGB pode ser demonstrado na prática com o uso de luzes verde, vermelha e azul, formando-se as composições do modelo aditivo. O vídeo sugerido, a seguir, demonstra as composições foi disponibilizado em: PASCOSCIENTIFIC. **Color Mixer & Accessory Kit, 2011.**

Os modelos de cor RGB e CMY são modelos que usam combinação de cores. Um outro conjunto de modelos de cor representa as cores em três componentes, sendo um componente de luminância ou intensidade e dois componentes de cromaticidade (SOLOMON; BRECKON, 2013). O modelo HSL representa a cor pelos componentes de cromaticidade matiz (H, do inglês *hue*) e saturação (S), e pela componente de luminância (L). É um modelo bastante utilizado no processamento de imagens, isso porque alguns algoritmos de processamento de imagens requerem uma relação de ordem entre os elementos do contradomínio. Uma imagem digital em RGB é uma função $f:\mathbb{Z}^2 \to \mathbb{Z}^3$, na qual as três dimensões do contradomínio são o cubo de cores RGB. Não existe uma relação de ordem entre as cores do cubo RGB. O contradomínio da componente luminância (L) do HSL contém todos os contornos da imagem (como uma fotografia monocromática) e possui a relação de ordem (número inteiro). Assim, a imagem deve ser convertida do RGB para o HSL, o processamento em tons de cinza, aplicado sobre a componente L, e a imagem resultante, convertida de volta para RGB. Há outros modelos de cor que se assemelham ao HSL, com uma componente de luminância e duas de cromaticidade, como o HSI, HSV, Lab e YCbCr (GONZALEZ; WOODS, 2011).

Nesta seção foram apresentados os fundamentos técnicos de imagens vetoriais e matriciais, desde a definição matemática de imagem até a possível representação de uma imagem na memória por meio de uma linguagem de programação. Foram também apresentados os modelos de cor mais usados para computação gráfica e processamento de imagens. A partir de agora será possível aprofundar na programação.

Sem medo de errar

Caro aluno, depois de definir requisitos técnicos para montar a equipe de processamento gráfico do seu cliente, seu trabalho agora é padronizar os formatos de imagens e vídeos e criar um projeto piloto para servir de modelo para a nova equipe. Para criar este projeto piloto, é preciso, antes de tudo, definir as tecnologias a serem usadas: linguagem de programação, bibliotecas e APIs. A escolha das tecnologias deve levar em conta não só aspectos técnicos relativos ao acervo existente de conteúdo gráfico como também aspectos humanos, com base em consulta à equipe de tecnologia da informação do cliente.

Pela descrição do contexto do cliente, ele possui conteúdo gráfico tanto em formato vetorial quanto em formato matricial, seja em imagens ou vídeos. Isso significa que as tecnologias escolhidas deverão cobrir ambos os formatos. Portanto, a linguagem de programação a ser adotada deve dispor de bibliotecas para lidar com ambos os formatos, o que não é uma dificuldade no mundo do processamento gráfico, uma vez que as principais linguagens de programação são suficientemente providas de bibliotecas. Dê preferência a linguagens que possuam

bibliotecas compiladas e não interpretadas, pois o processamento gráfico executa muitos laços de repetição e cálculos de números reais. Com esse requisito claro, solicite a sugestão da equipe atual do seu cliente. Afinal, a nova equipe será incorporada a eles e serão eles que deverão manter atividades futuras de capacitação da equipe e novas contratações.

Uma possibilidade de solução é criar um projeto piloto muito simples que execute as seguintes etapas:

- Abra uma imagem vetorial do acervo de conteúdos gráficos do seu cliente, usando a API de manipulação de imagens vetoriais escolhida.
- Altere a imagem vetorial adicionando um círculo verde e usando um tipo de dados da API.
- Salve em arquivo a imagem vetorial alterada, usando ainda a API.
- Transforme a imagem vetorial em imagem digital, utilizando os recursos de digitalização disponíveis na mesma API.
- Salve em arquivo a imagem digital obtida, usando a segunda API, que é a API de processamento de imagens.
- Desenhe um retângulo de cor vermelha sobre uma parte da imagem digital.
- Salve em novo arquivo a imagem digital com o retângulo adicionado, usando novamente a API de processamento de imagens.

Com esta sugestão, o projeto piloto resultante será capaz de demonstrar a leitura e escrita de arquivos de imagens vetoriais e matriciais, e a manipulação de imagens vetoriais e matriciais. Todos os projetos que virão poderão utilizar seu projeto piloto como *template*.

Ao solucionar este problema, demonstra-se a compreensão dos fundamentos das imagens e dos formatos de imagens.

Avançando na prática

Sistema de informação geográfica com imagens de satélite

Descrição da situação-problema

Sua equipe foi contratada para desenvolver um novo sistema de cadastro georreferenciado de nascentes de água. O trabalho vai se iniciar pela delineação de nascentes pelos próprios proprietários das áreas rurais que as

englobam. O proprietário da área rural deverá marcar sobre uma imagem de satélite as áreas do terreno onde se encontram cada uma das nascentes de sua propriedade em um sistema Web. Para tanto, o usuário deverá visualizar a imagem de satélite pelo sistema, dar zoom no entorno do ponto de interesse e, então, desenhar sobre a imagem de satélite um contorno que inclua o ponto da nascente.

Você já adquiriu as imagens de satélite e agora deve definir em que pontos do sistema você deve utilizar imagens vetoriais e em que pontos você deve utilizar imagens matriciais.

Resolução da situação-problema

O primeiro ponto a ser observado é que as imagens originais do sistema são imagens adquiridas por satélites, por um processo de captura de imagens. São fotografias e, portanto, imagens matriciais. Não há meios de se utilizar imagens vetoriais para as imagens de satélite.

Por sua vez, quando um usuário interage com o sistema, desenhando sobre a imagem de satélite uma área onde se encontra uma nascente, o sistema vai gerar em *background* uma imagem que representa o desenho do usuário. Tal imagem pode ser vetorial ou matricial, pois está sendo criada pelo sistema. No entanto, temos bons argumentos para defender que seja utilizada uma imagem vetorial. As imagens de satélite são imagens matriciais grandes, com alta resolução, enquanto as áreas que contêm nascentes são áreas pequenas. É mais vantajoso utilizar imagens vetoriais neste caso, pois com imagens vetoriais é possível armazenar apenas os pontos marcados pelo usuário, para que possam ser desenhados sobre a imagem de satélite.

Faça valer a pena

1. A representação de cores em imagens é feita com base nos modelos de cor. O modelo RGB é o mais conhecido, seguido pelo modelo CMY (ou CMYK), mas existe um outro grupo de modelos de cor, ao qual pertencem os modelos HSL, Lab e YCbCr. O HSL é um modelo bastante utilizado para o processamento de imagens.

Assinale a alternativa que apresenta a principal diferença entre o modelo RGB e o modelo HSL.

a) O RGB é um modelo aditivo, assim como o modelo CMY, enquanto o HSL é um modelo subtrativo, assim como seus similares Lab e YCbCr.

b) O RGB é um modelo de composição de cores primárias, enquanto o HSL é a composição de um componente de luminância e dois de cromaticidade.

c) O RGB é um modelo que representa as cores como uma composição das cores primárias do espectro visível, enquanto o HSL faz a composição de cores secundárias.
d) O RGB é um modelo com um componente de luminância e dois de cromaticidade, enquanto o HSL é um modelo de composição de cores primárias.
e) O RGB é um modelo subtrativo, assim como o modelo CMY, enquanto o HSL é um modelo aditivo, assim como seus similares Lab e YCbCr.

2. Uma imagem vetorial é um conjunto de equações matemáticas de formas geométricas. As equações matemáticas são definidas no espaço euclidiano contínuo. Já as imagens digitais são um conjunto de pontos em um espaço discreto. Você deseja manter um conteúdo gráfico produzido por designers para a propaganda dos produtos de sua empresa e pode optar por imagens vetoriais ou bitmap. Cada elemento gráfico pode ser aplicado sobre itens pequenos, como um chaveiro, assim como sobre itens grandes, como um outdoor.

Assinale a alternativa que apresenta a melhor escolha com a devida justificativa.
a) Deve-se optar por imagens vetoriais. Como a imagem matricial é um conjunto de equações, não pode ser visualizada e tampouco impressa sem antes ser vetorizada. É preferível, portanto, já armazenar o conteúdo em formato vetorial.
b) Deve-se optar por imagens vetoriais. Como a imagem vetorial é resultante da captura de cenas do mundo real, os resultados de aplicações de imagens vetoriais produzem impressões mais realísticas das imagens.
c) Deve-se optar por imagens matriciais. Como a imagem vetorial é um conjunto de equações, não pode ser visualizada e tampouco impressa sem antes ser digitalizada. É preferível, portanto, já o conteúdo em formato digital.
d) Deve-se optar por imagens vetoriais. Como a imagem vetorial está no espaço contínuo, pode ser desenhada em qualquer resolução de pixels, podendo ser desenhada em baixa resolução ou ampliada para um outdoor sem distorções.
e) Deve-se optar por imagens matriciais. Como a imagem matricial está no espaço discreto, pode ser desenhada em qualquer resolução de pixels, podendo ser ampliada para um outdoor sem distorções.

3. Uma imagem digital bidimensional em tons de cinza é representada na memória por um vetor v unidimensional com $xs \times ys$ elementos, sendo xs o número de elementos em cada linha da imagem e ys o número de linhas da imagem. Suponha que você deseja espelhar esta imagem na horizontal.

Assinale a alternativa que descreva o procedimento que executa corretamente o espelhamento da imagem sem nova alocação de memória.

a)

```
procedimento espelhaH(v, xs, ys):
  para todo y no intervalo [0..ys):
    para todo x no intervalo [0..(xs/2)]:
      tmp = x*ys + y
      v[x*ys+y] = v[x*ys + (ys-y-1)]
      v[x*ys + (ys-y-1)] = tmp
```

b)

```
procedimento espelhaH(v, xs, ys):
  para todo x no intervalo [0..xs):
    para todo y no intervalo [y..(ys/2)]:
      tmp = y*xs + x
      v[y*xs+x] = v[(ys-y-1)*xs + x]
      v[(ys-y-1)*xs + x] = tmp
```

c)

```
procedimento espelhaH(v, xs, ys):
  para todo y no intervalo [0..ys):
    para todo x no intervalo [0..(xs/2)]:
      tmp = y*xs + x
      v[y*xs+x] = v[y*xs + (xs-x-1)]
      v[y*xs + (xs-x-1)] = tmp
```

d)

```
procedimento espelhaH(v, xs, ys):
  para todo x no intervalo [0..xs):
    para todo y no intervalo [y..(ys/2)]:
      tmp = x*ys + y
      v[x*ys+y] = v[(xs-x-1)*ys + y]
      v[(xs-x-1)*ys + y] = tmp
```

e)

```
procedimento espelhaH(v, xs, ys):
  para todo y no intervalo [0..ys):
    para todo x no intervalo [y..xs):
      tmp = y*xs + x
      v[y*xs+x] = v[x*xs+y]
      v[x*xs+y] = tmp
```

Seção 1.3

CGPI: representações da imagem

Diálogo aberto

Caro aluno, nesta seção você será apresentado a soluções concretas de computação gráfica, na forma de algoritmos que você deverá implementar, demonstrando a capacidade de desenvolvimento de softwares para atender completamente às necessidades de seus clientes.

Até o momento você já conhece os requisitos necessários para a contratação de uma equipe de computação gráfica e processamento de imagens e já tem condições de definir estruturas de dados que representem imagens vetoriais e matriciais; porém, falta-lhe apenas demonstrar a capacidade de desenvolver códigos que implementam algoritmos executáveis.

Nesta seção você será apresentado a soluções concretas de computação gráfica, na forma de algoritmos que você deverá implementar, demonstrando a capacidade de desenvolver software para atender completamente às necessidades de seus clientes.

Lembre-se de que seu cliente é uma empresa de grande porte que te contratou para implantar uma equipe técnica de computação gráfica e processamento de imagens, mas exigiu também que você defina as estruturas de dados a serem usadas pela equipe e que apresente um projeto piloto, pequeno, para demonstrar boas práticas de programação com o uso das APIs e estruturas de dados escolhidas.

Você já definiu estruturas de dados e uma linguagem de programação, mas deve ainda definir uma suíte de desenvolvimento para a linguagem de programação escolhida, desenvolver e entregar um programa que cria retas e círculos em uma imagem vetorial, além de implementar algoritmos clássicos de desenhos de retas e círculos em imagens matriciais, demonstrando o uso desejado da suíte de desenvolvimento, das APIs e das estruturas de dados definidas.

Quais são os algoritmos existentes? Vamos conhecer alguns e desenvolver códigos que atendem à demanda?

Bons estudos!

Não pode faltar

Com base nos fundamentos de imagens apresentados na Seção 1.2, vamos agora explorar questões mais concretas a fim de começarmos a desenvolver programas que processam imagens vetoriais e matriciais.

A implementação de programas que processam imagens, bem como as estruturas de dados seguem os conceitos apresentados na Seção 1.2. A partir de agora vamos tornar esses conceitos mais concretos, trabalhando com implementações em Python de algoritmos e estruturas de dados.

A linguagem Python é uma linguagem gratuita provida de um vasto conjunto de bibliotecas gratuitas para computação gráfica e processamento de imagens. Possui sintaxe simplificada, nos moldes de linguagens de script, mas com suporte à orientação de objetos e integração com bibliotecas de alta performance, implementadas em C. Possui, também, bibliotecas gratuitas de manipulação de imagens vetoriais e matriciais bastante intuitivas, o que permite concentrarmo-nos nos conceitos de computação gráfica e processamento de imagens e não em dificuldades da linguagem ou das bibliotecas.

Para o processamento de imagens vetoriais, utilizaremos a biblioteca Pycairo, uma interface Python para a biblioteca gráfica cairo, implementada em C. Para criar uma imagem vetorial no cairo, é preciso criar um objeto Surface (como vimos na Seção 1.2, uma imagem bidimensional é uma função $f(x,y)$, portanto, uma superfície). Há vários tipos de Surfaces, que variam de acordo com o tipo de descritor de imagem desejado. Para se criar uma imagem v em memória, a ser posteriormente escrita em um arquivo do tipo *Scalable Vector Graphics* (SVG) e nome *exemplo.svg*, com resolução de 200x200 pixels, basta uma linha, além de importar a biblioteca.

```
import cairo
v = cairo.SVGSurface("cgpiU1S2.svg", 201, 201)
```

Para o processamento de imagens matriciais, utilizaremos a biblioteca NumPy, também implementada em C e com interface para o Python. NumPy é uma biblioteca que facilita a manipulação de matrizes em Python. Para se criar uma imagem matricial f em níveis de cinza, toda preta, de resolução 200x200 pixels, basta importar a biblioteca e criar uma matriz 200x200 de zeros no tipo inteiro de 8 bits.

```
import numpy
f = numpy.zeros((200,200),dtype=numpy.uint8)
```

Para se criar uma imagem RGB toda branca, basta criar uma matriz 3x200x200 inicializada com os valores 255 em todos os pontos. Para isso, cria-se a imagem com valores 1 e multiplica-se por 255.

```
import numpy
frgb = 255*numpy.ones((3,200,200),dtype=numpy.uint8)
```

A alteração de um ponto se faz usando as coordenadas inteiras da matriz. Para colocar um ponto branco no centro da imagem *f*, basta atribuir o valor 255 ao ponto de coordenada (100,100).

```
f[100,100] = 255
```

Criação de retas em imagens vetoriais e matriciais

Tanto em imagens vetoriais quanto em imagens matriciais, a criação de retas, de fato, não é feita, mas sim a criação de segmentos de retas, visto que as imagens são funções bidimensionais dentro dos limites de um quadro.

A criação de um segmento de reta em uma imagem vetorial é muito simples. Basta acrescentar a descrição de que deve ser adicionado um segmento de reta. A reta deverá ser desenhada posteriormente sobre uma imagem matricial pelo aplicativo que exibirá ou imprimirá a imagem vetorial. Com o cairo é preciso criar um Context para desenhar sobre uma Surface. Considerando a imagem vetorial *v* criada anteriormente, isto é feito como se segue.

```
context = cairo.Context(v)
context.scale(10, 10)
context.set_line_width(0.02)
context.move_to(0.2, 0.2)
context.line_to(0.8,0.6)
context.stroke()
```

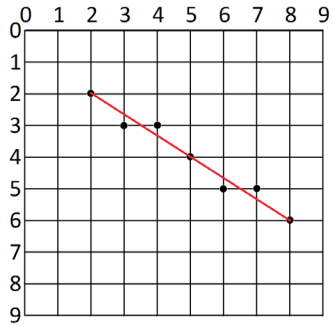
Vamos compreender o código acima. Primeiramente cria-se o Context chamando o construtor da classe Context (`cairo.Context`) e define-se a escala (`context.scale`), que neste caso é a mesma das dimensões da

imagem *v* criada. Define-se a espessura da linha a ser desenhada (`context.set_line_width`) o ponto inicial (`context.move_to`) e o ponto final (`context.line_to`). Os parâmetros dessas três últimas funções são valores entre 0 e 1, que representam o percentual da escala definida pelo `context.scale`.

A criação de segmentos de retas em imagens matriciais exige o desenho de cada ponto pertencente ao segmento. Isso nem sempre é evidente, visto que uma imagem matricial possui pontos apenas em coordenadas inteiras. Para demonstrar a dificuldade, a Figura 1.13 mostra o desenho do segmento de reta acima sobre a imagem matricial.

Cada ponto da imagem matricial está representado pelos cruzamentos das linhas de grade. O segmento de reta contínuo está representado em vermelho. Note que, com esta inclinação, a reta passa apenas por 3 pontos da linha de grade e 4 outros pontos devem ser aproximados. O problema das aproximações ocorrerá para o desenho de praticamente toda forma geométrica. Começaremos por apresentar algoritmos de desenho de retas e círculos.

Figura 1.13 | Representação de um segmento de reta de imagem matricial



Fonte: elaborada pelo autor.

Exemplificando

Se considerarmos uma imagem matricial sendo exibida como fundo de tela em um monitor com 1000 pixels de largura, uma linha de apenas um pixel de espessura pode ficar pouco visível para o usuário. Portanto, pode ser desejável desenhar a linha com uma determinada espessura de *e* pixels, tornando-a mais visível. Para que a linha seja desenhada com uma determinada espessura *e* (`context.set_line_width`), em vez de desenhar cada ponto da linha, deve-se desenhar um disco de diâmetro *e* centralizado em cada ponto da linha.

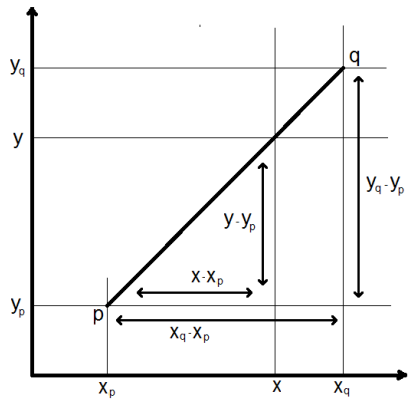
Algoritmo para o desenho de retas

O algoritmo mais conhecido para o desenho de retas é o algoritmo de Bresenham. Vamos criar a função *desenhaReta(f,p,q,g)*, que desenha sobre a imagem matricial *f* um segmento de reta que parte do ponto $p = (xp, yp)$ para o ponto $q = (xq, yq)$, na cor *g*.

Comecemos pelo caso especial em que $xp < xq$, $yp \leq yq$ e $(yq - yp) \leq (xq - xp)$, ilustrado na Figura 1.14. Neste caso, a inclinação da reta é dada por $s = \frac{yq - yp}{xq - xp}$. No espaço contínuo, para qualquer coordenada x, $xp \leq x \leq xq$, é possível encontrar a coordenada y correspondente, usando a inclinação $s = \frac{y - yp}{x - xp}$, ou, isolando y, $y = yp + s(x - xp)$.

O algoritmo mais direto para o desenho de reta consiste em iterar pelos valores inteiros possíveis de x, $xp \leq x \leq xq$, calcular o y correspondente e arredondar y de forma a obter o inteiro mais próximo. Podemos, então, implementar a função *desenharRetaCaso1(f,p,q,g)*. Observe que o acesso ao ponto (x,y) é feito por f[y,x].

Figura 1.14 | Inclinação da reta



Fonte: elaborada pelo autor.

```
def desenhaRetaCaso1(f,p,q,g):
    (yp,xp) = p;  (yq,xq) = q
    s = float(yq-yp)/(xq-xp)
    for x in range(xp,xq+1):
        y = round(yp + s*(x-xp))
        f[y,x]=g
```

Para otimizar a função *desenhaRetaCaso1*, Bresenham eliminou o uso de multiplicações de números reais e arredondamentos, trabalhando apenas com números inteiros. Primeiramente, é preciso considerar que a inclinação da reta é também definida por dois pontos consecutivos $s = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}$. Como a iteração é em x, $x_{i+1} - x_i = 1$, logo, $y_{i+1} = y_i + s$. A inclinação s é

o tamanho do deslocamento em *y* para cada unidade deslocada em *x*. Mas $0 \leq s \leq 1$ para o caso $(yq - yp) \leq (xq - xp)$. Se $s \geq 0{,}5$, y_{i+1} será arredondado para $y_i + 1$, senão, será arredondado para y_i. Se dermos mais um passo em *x*, teremos $y_{i+2} = y_i + 2s$. Podemos, portanto, acumular o deslocamento em *y* em uma variável *d*, mantendo *d* no intervalo $-0{,}5 \leq d < 0{,}5$. A variável *d* é inicializada em 0. A cada iteração de *x*, *d* é incrementada de *s*. Se $d \geq 0{,}5$, significa que teremos um arredondamento para cima de *y*. Podemos incrementar *y* de 1 e decrementar o deslocamento *d* de 1, como feito na função *desenhaRetaCaso2* que segue:

```
def desenhaRetaCaso2(f,p,q,g):
    (yp,xp) = p;  (yq,xq) = q
    y,d = yp, 0.0
    s = float(yq-yp)/(xq-xp)
    for x in range(xp,xq+1):
        f[y,x]=g;  d += s
        if (d >= 0.5):
            y += 1;  d -= 1
```

Para ilustrar a função *desenhaRetaCaso2*, considere a Figura 1.15. Neste caso temos *(xp,yp)=(2,2)*, *(xq,yq)=(6,2)* e $s = \frac{6-2}{8-2} = \frac{2}{3}$. O deslocamento *d* é inicializado em *0*. No primeiro passo, para *x=2*, o ponto *(2,2)* é desenhado e *d* é incrementado em $\frac{2}{3}$; mas como este incremento faz $d \geq 0{,}5$, então *y* é incrementado de 1 e *d* decrementado de 1, passando a valer $-\frac{1}{3}$.

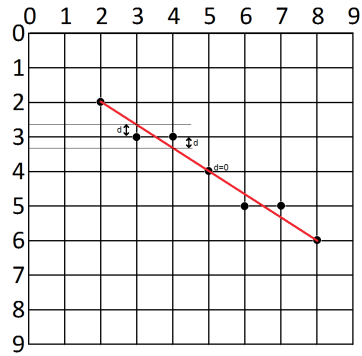
No segundo passo, para *x=3*, o ponto *(3,3)* é desenhado, *d* é incrementado em $\frac{2}{3}$, passando a valer $\frac{1}{3}$, que é menor que 0,5, então *y* e *d* não são alterados.

No terceiro passo, para *x=4*, o ponto *(4,3)* é desenhado, *d* é incrementado

em $\frac{2}{3}$ e passa a valer 1. Novamente, como $d \geq 0{,}5$, então y é incrementado de 1 e d decrementado de 1, passando a valer 0. O próximo ponto a ser desenhado é o ponto *(5,4)* e os passos seguintes seguem a mesma lógica.

Para eliminar ainda mais operações de ponto flutuante, vamos alterar o tipo da variável d. Considere $dx = xq - xp$ e $dy = yq - yp$, a cada iteração d é incrementado de dy/dx e é comparado com 0,5. Podemos incrementar d de dy e compará-lo com $0{,}5dx$ ou, ainda melhor, incrementar d de $2dy$ e compará-lo com dx, pois assim só trabalharemos com números inteiros, como implementado na função *desenhaRetaCaso3* que segue. Trata-se de uma manipulação meramente matemática que produz o mesmo resultado da função *desenhaRetaCaso2* sem uso de números reais.

Figura 1.15 | Plotagem da reta



Fonte: elaborada pelo autor.

```
def desenhaRetaCaso3(f,p,q,g):
    (yp,xp) = p;  (yq,xq) = q
    dx,dy = xq-xp,yq-yp
    s = 2*dy;   c = 2*dx;   d = 0;   y = yp
    for x in range(xp,xq+1):
        f[y,x]=g;  d += s
        if (d >= dx):
            y += 1;   d -= c
```

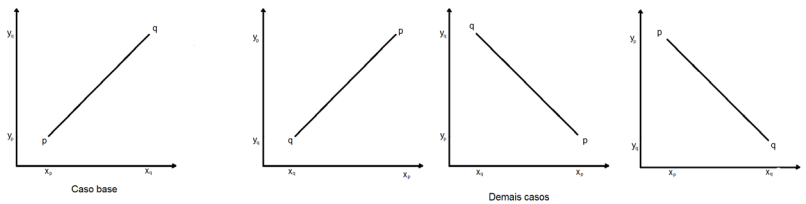
Assimile

Algoritmos de computação gráfica e processamento de imagens exigem muitas iterações. O desenho de uma forma, por exemplo, com um milhão de pontos, exigirá um milhão de iterações, uma para cada ponto.

No nível do hardware, operações com números inteiros são executadas muito mais rápido que operações com números reais. Por esse motivo, o uso de variáveis inteiras deve ser sempre priorizado. Operações com inteiros são executadas mais rápido no processador.

A função *desenhaRetaCaso3* implementa somente o caso especial em que $xp < xq$, $yp \leq yq$ e $dy \leq dx$. Para generalizar essa função, é preciso tratar os casos em que $xp \geq xq$ e $yp > yq$, ilustrados na Figura 1.16, fazendo-se a iteração com decrementos ao invés de incrementos, e tratar os casos em que $dy > dx$, trocando toda a iteração para *y* no lugar de *x*.

Figura 1.16 | Todos os casos para o desenho de reta



Fonte: elaborada pelo autor.

Finalmente obtemos a função *desenhaReta*, que é a implementação final do algoritmo de Bresenham.

```
def desenhaReta(f,p,q,g):
  (yp,xp) = p;  (yq,xq) = q
  dx,stx = xq-xp,1;  dy,sty = yq-yp,1;  d = 0
  if (dx < 0):
    dx,stx = -dx,-1
  if (dy < 0):
    dy,sty = -dy,-1
  if (dx >= dy):
    s = 2*dy;  c = 2*dx;  y = yp
    for x in range(xp,xq+stx,stx):
      f[y,x]=g;  d += s
      if (d >= dx):
```

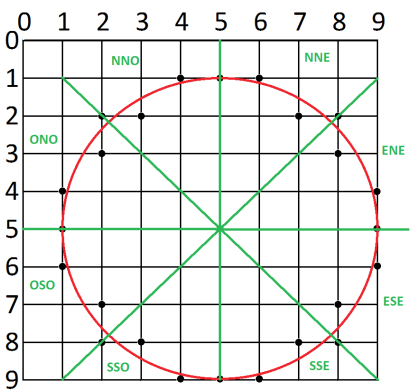
```
            y += sty;  d -= c
    else:
      s = 2*dx;  c = 2*dy;  x = xp
       for y in range(yp,yq+sty,sty):
         f[y,x]=g;  d += s
         if (d >= dy):
           x += stx;  d -= c
```

Pesquise mais

É possível melhorar ainda mais a performance do algoritmo de desenho de retas. Ammeraal e Zhang (2008) apresentam uma otimização ainda melhor para o algoritmo de Bresenham.

AMMERAAL, L.; ZHANG, K. **Computação Gráfica para Programadores Java.** 2. ed. Rio de Janeiro: LTC, 2008. p. 57-60.

Figura 1.17 | O círculo discreto e suas simetrias



Fonte: elaborada pelo autor.

Algoritmo para o desenho de círculo

O algoritmo para o desenho de círculos se comporta de maneira bastante similar ao de desenho de retas, mas para compreendê-lo, é preciso apresentar algumas propriedades do círculo discreto. A equação do círculo contínuo de centro em *(xc,yc)* e raio *r* é $(xc+x)^2+(yc+y)^2=r^2$, em que *(x,y)* é um deslocamento a partir do centro *(xc,yc)*. A Figura 1.17 representa um círculo contínuo em vermelho e os pontos discretos, sobre a malha inteira, que se aproximam do círculo contínuo. Em verde são mostrados os eixos que dividem o círculo em pontos subcolaterais. O círculo discreto apresenta simetria com relação ao eixo *x*, com relação ao eixo *y* e com relação às diagonais. Isso significa que o algoritmo para desenho de círculos só precisa

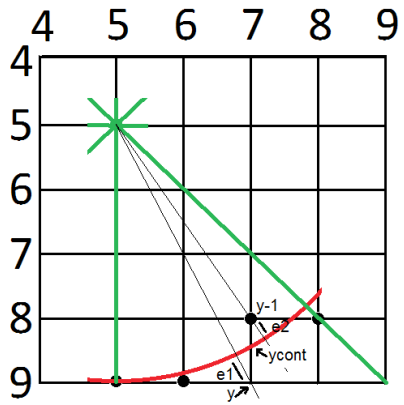
calcular de fato os pontos do arco de um ponto subcolateral. Os outros sete arcos são obtidos por simetria.

Reflita

O círculo contínuo apresenta simetria com relação a qualquer reta que passe pelo seu centro. Por que o círculo discreto só apresenta simetria com relação a essas quatro retas da Figura 1.17? Teria a ver com o fato de serem as únicas retas que passam exatamente sobre a malha discreta?

Vamos desenvolver um raciocínio com base na Figura 1.17 para desenhar apenas os pontos do arco SSE. Vamos iniciar com o desenho do ponto (5,9), sobre o eixo vertical de simetria. Iniciaremos com $x = 0$ e $y = r = 4$, pois assim $(xc+x,yc+y)=(5+0,5+4)=(5,9)$.

Figura 1.18 | Erros no desenho do círculo



Fonte: elaborada pelo autor.

Ao incrementar x, a coordenada y_{cont} do círculo contínuo vai diminuir, ficando em um valor no intervalo $y-1 \leq y_{cont} \leq y$. Portanto, ao arredondar y_{cont}, obteremos y ou $y-1$. Um jeito simples de se descobrir o novo valor de y seria calcular x^2+y^2 e também $x^2+(y-1)^2$ e verificar qual dos dois valores é o mais próximo de r^2. Esta situação é ilustrada na Figura 1.18, que mostra somente o arco SSE extraído da Figura 1.17.

Na Figura 1.18 são mostrados y, y_{cont} e $y-1$, para $x=7$. Se for desenhado o ponto y, o erro será $e1$. Se for desenhado o ponto $y-1$, o erro será $e2$. Neste caso, $e2<e1$, portanto deve ser desenhado o ponto em $y-1$.

Para evitar o cálculo dos quadrados, vamos introduzir três variáveis inteiras. As diferenças entre dois quadrados consecutivos (dx e dy) e o erro do raio inteiro com relação ao raio contínuo e. Como x é crescente e y decrescente, temos $dx=(x+1)^2-x^2=2x+1$, $dy=y^2-(y-1)^2=2y-1$ e $e=x^2+y^2-r^2$.

Como iniciamos com $x = 0$ e $y = 4$, temos $dx = 1$, $dy = 7$, $e = 0$. Ao incrementar x de 1, o erro será incrementado de dx, e dx será incrementado de 2.

Agora é preciso decidir se vamos decrementar *y* ou não. Se decrementarmos *y* de 1, o erro será decrementado de *dy*, e *dy* será decrementado de 2. Como desejamos minimizar o erro, o que precisamos testar é se o erro após o decremento de *y* é menor do que o erro sem o decremento, ou seja, se $|e-dy|<|e|$, então *y* deve ser decrementado. Para eliminar os valores absolutos, podemos utilizar a inequação $(e-dy)^2 < e^2$, que, após algumas manipulações pode-se provar equivalente a $dy < 2e$. Este é o teste que devemos fazer. A função *desenhaSSE* a seguir desenha na cor *g*, sobre a imagem matricial *f*, o arco SSE de raio *r* e centro em *c*.

```
def desenhaSSE(f,c,r,g):
  (xc,yc) = c;  x,y,dx,dy,e = 0,r,1,2*r-1,0
  while (x<y):
    f[yc+y,xc+x] = g #SSE
    x+=1;    e+=dx;   dx+=2
    if (dy < 2*e):
      y-=1;  e-=dy;  dy-=2
```

Agora basta replicar a função *desenhaSSE* para os outros sete arcos simétricos. Finalmente a função *desenhaCirculo* a seguir generaliza *desenhaSSE* e desenha todos os arcos dos pontos subcolaterais.

```
def desenhaCirculo(f,c,r,g):
  (xc,yc) = c;   x,y,dx,dy,e = 0,r,1,2*r-1,0
  while (x<y):
    f[yc+y,xc+x] = g #SSE
    f[yc-y,xc-x] = g #NNO
    f[yc-x,xc+y] = g #ENE
    f[yc+x,xc-y] = g #OSO
    x+=1;    e+=dx;   dx+=2
    if (dy < 2*e):
```

```
    y-=1;  e-=dy;  dy-=2
  if (x>y):
    break
  f[yc+x,xc+y] = g #ESE
  f[yc+y,xc-x] = g #SSO
  f[yc-x,xc-y] = g #ONO
  f[yc-y,xc+x] = g #NNE
```

Com os algoritmos apresentados nesta seção você já é capaz de resolver a situação problema apresentada. Os conceitos geométricos apresentados são também suficientes para que outros algoritmos possam ser compreendidos e para que você seja capaz de resolver outras situações similares. Os próximos passos serão as transformações geométricas e a modelagem tridimensional, que vão consolidar as bases para a síntese de imagens e o processamento de imagens digitais. Ainda temos muito o que explorar!

Sem medo de errar

Ao longo do processo de definição de equipe e procedimentos para o cliente, exige-se que você defina as estruturas de dados a serem usadas pela equipe e que você apresente um projeto piloto pequeno para demonstrar boas práticas de programação com o uso das APIs e estruturas de dados escolhidas. É preciso definir uma suíte de desenvolvimento para a linguagem de programação escolhida, desenvolver e entregar um programa que cria retas e círculos em uma imagem vetorial, além de implementar algoritmos clássicos de desenhos de retas e círculos em imagens matriciais, demonstrando o uso desejado da suíte de desenvolvimento, das APIs e das estruturas de dados definidas.

A solução para esta situação-problema sempre dependerá da linguagem de programação e das APIs de desenvolvimento escolhidas anteriormente. Um dos requisitos para a escolha da linguagem de programação é o próprio conjunto de APIs disponíveis para essa linguagem. A escolha da linguagem e de APIs ocorre quase que concomitantemente.

Uma vez escolhidas linguagem e APIs, para desenvolver o programa que cria retas e círculos em uma imagem vetorial e implementa algoritmos clássicos de desenhos de retas e círculos em imagens matriciais, pode-se:

- Implementar os algoritmos de Bresenham de desenho de reta e círculo, conforme visto nas seções.
- Implementar um programa simples interativo, no qual o usuário escolhe que retas e círculos deseja desenhar.

A interação pode ser pela linha de comando ou, se desejar, de forma mais sofisticada, com interface gráfica. Pela linha de comando sugere-se:

1. O programa pergunta ao usuário se quer desenhar uma reta ou círculo.
2. Após a escolha, o usuário informa os parâmetros para a forma geométrica escolhida: ponto inicial, ponto final, espessura, ponto central, raio ou cor.
3. Com os parâmetros passados pelo usuário, o programa cria a reta ou círculo em uma imagem vetorial e desenha-o em uma imagem matricial, ambos em memória.
4. Os passos de 1 a 3 podem ser repetidos quantas vezes o usuário desejar.
5. Quando o usuário decidir finalizar os desenhos, o programa salva a imagem vetorial em um arquivo SVG e a imagem matricial em um arquivo PNG.

Um programa simples como este é suficiente para demonstrar o uso das APIs e estruturas de dados escolhidas. Junto com os passos anteriores de definição de requisitos técnicos para os profissionais da equipe, a resolução desta situação-problema permite demonstrar a compreensão das áreas da computação gráfica, dos fundamentos das imagens e dos formatos de imagens.

Avançando na prática

Aglomerações em bolhas

Descrição da situação-problema

Você foi contratado por uma rede de farmácias para implementar a exibição de todos os pontos de localização das farmácias da rede sobre um mapa. Cada farmácia é marcada no mapa por um desenho de um alfinete. Você já possui a ferramenta que desenha um alfinete sobre o mapa. A localização de cada farmácia é dada pela posição geográfica aferida por um GPS e, portanto, trata-se de coordenadas contínuas. O marcador deve ser desenhado

sempre do mesmo tamanho, independente do zoom que o usuário faça no mapa. Esse marcador é uma imagem de 32x32 pixels. Acontece que, se o usuário dá um *zoom out* no mapa, muitos dos seus marcadores ficam sobrepostos, fazendo com que seja difícil mostrar para o usuário até mesmo o número de farmácias que se encontram numa determinada região. A solução para isso é criar aglomerados de pontos e representar por um círculo de raio *n* um conjunto de *n* farmácias próximas. Você precisa definir os procedimentos para criar esta ferramenta.

Resolução da situação-problema

Primeiramente é preciso observar que a localização no GPS de uma farmácia é um ponto de coordenadas contínuas, que devem ser convertidas para a resolução espacial da imagem do mapa sendo exibida, para que o alfinete seja desenhado sobre a coordenada discreta correta. Esta conversão é feita com base na escala do mapa (ou resolução espacial), como vimos nesta seção.

Como o alfinete tem dimensões 32x32 pixels, sempre que dois centros de alfinetes estiverem a uma distância menor que 32 pixels, os alfinetes vão se sobrepor. Neste caso você precisa criar um aglomerado de dois alfinetes. Todo alfinete que estiver ainda em distância inferior a este limite deve ser adicionado ao aglomerado.

Após definido o aglomerado, você deve calcular o epicentro de todos os alfinetes aglomerados. Este será o centro do seu círculo.

Sobre este centro você poderá desenhar um círculo de raio *n*, onde *n* é o número de alfinetes aglomerados, utilizando o algoritmo de Bresenham.

Faça valer a pena

1. A computação gráfica e o processamento de imagens utilizam dois tipos básicos de imagens: as imagens vetoriais e as imagens matriciais. O primeiro descreve formas geométricas no espaço contínuo, o segundo representa a imagem como uma matriz de pontos no espaço discreto. Cada um desses tipos é mais adequado para cada uma das subáreas da computação gráfica.

Qual tipo de imagem é utilizado pelos profissionais da subárea de visão computacional e por quê?
a) Imagem vetorial, pois a visão computacional busca extrair informações das imagens, o que é mais fácil de ser obtido dos descritores contínuos de imagens vetoriais.

b) Imagem matricial, pois a visão computacional busca extrair informações das imagens capturadas do mundo real, que são imagens matriciais.
c) Ambos, pois a visão computacional busca extrair informações das imagens, sejam elas capturadas do mundo real ou sintetizadas.
d) Imagem vetorial, pois a visão computacional busca extrair informações de imagens sintéticas, que são imagens vetoriais.
e) Imagem matricial, pois a visão computacional busca simplesmente transformar as imagens por meio da aplicação de filtros.

2. Uma reta no espaço contínuo pode ser descrita simplesmente como dois pontos no plano. A visualização de uma reta, no entanto, é feita em um monitor ou tela que é uma matriz de pontos. Para a visualização de uma reta descrita em uma imagem vetorial é necessário desenhá-la sobre uma imagem matricial. Para o desenho de retas existe o algoritmo de Bresenham.

Sobre o algoritmo de Bresenham, assinale a alternativa verdadeira.
a) O algoritmo de Bresenham é capaz de desenhar apenas retas verticais e horizontais que passam exatamente sobre os pontos de coordenadas inteiras.
b) O algoritmo de Bresenham trabalha necessariamente com variáveis reais, pois exige que sejam feitas aproximações dos pontos da reta contínua para a reta discreta.
c) O algoritmo de Bresenham visa não somente a baixa complexidade assintótica, mas prioriza, também, o uso de variáveis inteiras para reduzir o custo computacional.
d) O algoritmo de Bresenham é capaz de desenhar apenas retas verticais, horizontais, e diagonais, que passam exatamente sobre os pontos de coordenadas inteiras.
e) O algoritmo de Bresenham trabalha exclusivamente com variáveis inteiras, visando reduzir a complexidade assintótica do algoritmo.

3. Um círculo no espaço contínuo pode ser descrito simplesmente como a coordenada do seu centro e seu raio. A visualização de um círculo, no entanto, é feita em um monitor ou tela que é uma matriz de pontos. Para a visualização de um círculo descrito em uma imagem vetorial, é necessário desenhá-lo sobre uma imagem matricial. Para isso, existe o algoritmo de Bresenham.

Sobre o algoritmo de Bresenham para o desenho de círculos em duas dimensões, assinale a alternativa verdadeira.
a) O algoritmo de Bresenham itera sobre metade dos valores possíveis de uma coordenada e um quarto dos valores possíveis da outra coordenada, explorando a simetria vertical.
b) O algoritmo de Bresenham itera sobre todos os valores possíveis de cada uma das coordenadas dos dois eixos, visto que o círculo discreto não apresenta simetrias como o círculo contínuo.

c) O algoritmo de Bresenham itera sobre todos os valores possíveis da coordenada de um dos eixos, sendo que a cada iteração, dois pontos são desenhados com base na simetria horizontal.
d) O algoritmo de Bresenham só itera sobre um quarto dos valores possíveis da coordenada de um dos eixos, pois os outros pontos são calculados pelas simetrias horizontal, vertical e diagonais.
e) O algoritmo de Bresenham só itera sobre metade dos valores possíveis da coordenada de um dos eixos, pois os outros pontos são calculados pelas simetrias horizontal e vertical.

Referências

AMMERAAL, L.; ZHANG, K. **Computação Gráfica para Programadores Java.** 2. ed. Rio de Janeiro: LTC, 2008.

ANGEL E.; SHREINER, D. **Interactive Computer Graphics:** a top-down approach with shader-based OpenGL. 6. ed. Addison-Wesley, 2012.

CAMARGO, I. BOULOS, P., **Geometria analítica:** um tratamento vetorial. 3. ed. São Paulo: Prentice Hall, 2005.

DAVID ROBUSTELLI. Pokemon Go HoloLens Mixed Reality, 2016. Disponível em: https://www.youtube.com/watch?v=0gHp8PuXAIk. Acesso em: 2 out. 2018.

GONZALEZ, R. C.; WOODS, R. E. **Processamento de Imagens Digitais.** 3. ed. [S.l.]: Pearson, 2011.

HUGHES, J. F. et al. **Computer graphics:** principles and practice. 3. ed. [S.l.]: Addison-Wesley, 2013.

IMAGEM. **Dicio –** Dicionário Online de Português. 2018. Disponível em: https://www.dicio.com.br/imagem/. Acesso em: 10 set. 2018.

JOMIRIFE. **Tutorial Blender 2.6 - Esqueleto/Ossos/Armature (Aula 3) (HD), 2012**. Disponível em: https://www.youtube.com/watch?v=-CUlXKMybzU. Acesso em: 2 out. 2018.

MANSSOUR, I. H.; COHEN, M. Introdução à Computação Gráfica. **Revista de Informática Teórica e Aplicada,** Porto Alegre, v. 13, n. 2, p. 43-67, 2006.

MICHAEL JACKSON VEVO. **Michael Jackson - Black Or White (Shortened Version), 2009.** Disponível em https://www.youtube.com/watch?v=F2AitTPI5U0. Acesso em: 2 out. 2018.

MURRAY, J. D.; VANWRYPER, W. **Encyclopedia of Graphics File Formats.** 3. [S.l.]: Ed. O'Reilly, 1996.

PASCOSCIENTIFIC. **Color Mixer & Accessory Kit, 2011.** Disponível em: https://www.youtube.com/watch?v=KZ-mEddsYqo. Acesso em: 5 nov. 2018.

PILLOW. **Pillow 5.2 – the friendly Python Imaging Library fork,** 2018. Disponível em: http://pillow.readthedocs.io. Acesso em: 7 nov. 2018.

PYCAIRO. **Pycairo 1.17 – Python bindings for cairo,** 2018. Disponível em: http://pycairo.readthedocs.io. Acesso em: 7 nov. 2018.

PYTHON SOFTWARE FOUNDATION. **Python 3.7.1 documentation,** 2018. Disponível em: https://docs.python.org/3/. Acesso em: 7 nov. 2018.

SANTOS, B. M.; ARTERO, A. O. Efeitos especiais em computação gráfica – Morphing. **Colloquium Exactarum,** Presidente Prudente, v. 3, n. 2, p. 85-92, 2011.

SCHEINERMAN, Edward R. **Matemática discreta:** uma introdução. São Paulo: Cengage Learning, 2011.

SHERMAN, W. R.; CRAIG, A. B. **Understanding Virtual Reality**: interface, application and design. Morgan Kauffman, 2003.

SOLOMON, C.; BRECKON, T. **Fundamentos de processamento digital de imagens:** uma abordagem prática com exemplos em Matlab. Rio de Janeiro: LTC, 2013.

VAN DER GRAAFF, K. M. **Anatomia Humana.** 6. ed. Barueri: Manole, 2003.

Unidade 2

Geometria do processamento gráfico

Convite ao estudo

Caro aluno, você já foi apresentado aos fundamentos de imagens e conhece tanto imagens vetoriais quanto matriciais. Nesta unidade você irá conhecer as transformações geométricas em diferentes contextos da computação gráfica: reconstrução 3D, animação gráfica e processamento de imagens. Ao final do estudo desta unidade você poderá demonstrar a capacidade de aplicar transformações geométricas e implementar um conjunto delas em modelos de câmera. É a base para a construção de animações gráficas 3D e também para diversas ferramentas de processamento de imagens.

Por trás de toda ferramenta de edição de imagens ou de modelagem 3D há sempre um profissional que desenvolve as ferramentas matemáticas de transformações geométricas de base. Esse profissional é um cientista da computação que precisa ter conhecimentos de geometria e também deve ser capaz de implementar os conceitos da geometria em linguagem de programação. Esse profissional pode ser você.

Considere o seguinte contexto. Sua empresa está desenvolvendo um novo aplicativo de criação de animações 3D e sua equipe de trabalho está encarregada do desenvolvimento da biblioteca de transformações geométricas. Como trata-se de uma biblioteca, o usuário não terá contato direto com o resultado do seu trabalho, mas sim com o resultado do trabalho de outras equipes que utilizarão sua biblioteca.

Para a criação de animações, o usuário irá utilizar uma interface gráfica interativa, na qual ele será capaz de visualizar modelos geométricos, podendo fazer rotações, translações ou aproximações. Será capaz também de definir o ponto de vista do espectador da animação, posicionando câmeras virtuais.

Você deverá desenvolver esta biblioteca em três etapas. A primeira etapa é a implementação das transformações geométricas básicas de escala, rotação, translação e perspectiva. A segunda etapa é a implementação de modelos geométricos tridimensionais, que poderão vir a ser transformados pelas ferramentas criadas na etapa anterior. Por último você deverá implementar modelos de câmera, que utilizarão as transformações geométricas básicas e poderão ser aplicados sobre os modelos geométricos.

As três seções desta unidade apresentam o conteúdo necessário para o desenvolvimento de cada etapa citada acima. A Seção 1 apresenta as transformações geométricas básicas, a Seção 2 apresenta os modelos geométricos tridimensionais e a Seção 3 apresenta os modelos de câmera.

Lembre-se de que por trás de todo aplicativo de computação gráfica ou processamento de imagens existe um profissional de bases teóricas sólidas de geometria e esse profissional pode ser você. Portanto, esteja atento ao conteúdo desta unidade e boa sorte!

Seção 2.1

CGPI: Transformações geométricas

Diálogo aberto

Caro aluno, mais uma vez vamos enfatizar a importância da geometria para o processamento gráfico. Toda animação gráfica depende da aplicação de transformações geométricas sobre imagens digitais ou vetoriais ou sobre modelos tridimensionais. Desde um simples efeito de minimização de janela do seu sistema operacional até a movimentação de uma câmera em um jogo digital são produzidos por transformações geométricas.

As transformações geométricas são implementadas na forma de multiplicação de matrizes. Isto explica porque são tão computacionalmente complexas tais operações, e porque é comum a adoção de uma placa de vídeo dedicada em computadores pessoais. Esta seção aborda os conceitos teóricos das transformações geométricas e é a base para quem deseja implementar aplicativos com qualquer tipo de animação.

Considere que sua empresa esteja desenvolvendo um novo aplicativo de criação de animações 3D e sua equipe de trabalho está encarregada do desenvolvimento da biblioteca de transformações geométricas. O usuário irá utilizar uma interface gráfica interativa, na qual ele será capaz de visualizar modelos geométricos podendo fazer rotações, translações ou aproximações. Será capaz também de definir o ponto de vista do espectador da animação, posicionando câmeras virtuais.

Você deve desenvolver esta biblioteca em três etapas.

A primeira etapa do seu trabalho é a implementação das transformações geométricas básicas. Você está encarregado de traduzir os conceitos de geometria e álgebra linear para código em linguagem de programação. As transformações geométricas a serem implementadas são escala, rotação, translação e perspectiva. Você deve entregar o código fonte do programa que implementa as transformações geométricas.

O conteúdo desta seção é o que você precisa para resolver a primeira etapa. Mãos à obra.

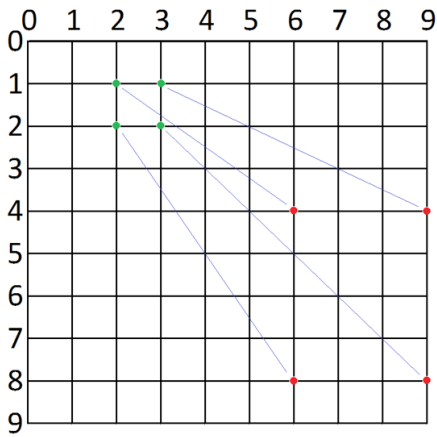
Não pode faltar

Caro aluno, você já conhece as representações de imagens digitais e vetoriais e vamos de agora em diante mostrar como aplicar transformações geométricas sobre essas imagens. Comecemos por compreender as transformações mais simples: escala e translação.

Translação e escala

A Figura 2.1 mostra a transformação de escala de quatro pontos: (2,1), (3,1), (2,2), (3,2). A escala aplicada foi de três vezes, no eixo *x*, e quatro vezes, no eixo *y*: $(s_x,s_y)=(3,4)$.

Figura 2.1 | Transformação de escala. Em verde os pontos originais e em vermelho os pontos após transformação



Fonte: elaborada pelo autor.

A transformação de escala de um ponto (x_1,y_1), por uma escala (s_x,s_y), se dá pelas equações

$$x_2 = s_x x_1$$

$$y_2 = s_y y_1$$

que podem ser representadas na forma de matriz como

$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$$

E pode ser estendido para três dimensões como

$$\begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix}$$

As escalas s_x e s_y podem ser inteiras ou reais. No caso de imagens matriciais o uso de escalas reais pode levar o ponto para um ponto fora da grade inteira e exigir aproximações. Para aplicar a transformação sobre um objeto qualquer, aplica-se a transformação para cada ponto do objeto, como foi feito na Figura 2.1.

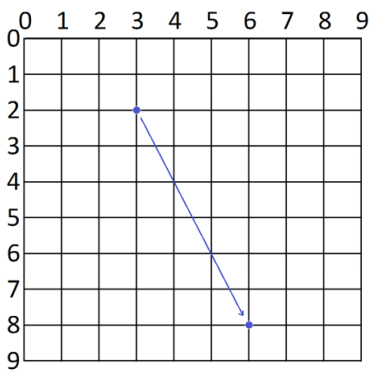
Pela representação na forma de matriz, podemos sempre considerar que o ponto p_2 é resultado da transformação geométrica sobre p_1 pela equação $p_2 = M \cdot p_1$, onde M é a **matriz de transformação** (BOLDRINI et al., 1986). Assim é possível observar também que as transformações são reversíveis aplicando-se a matriz inversa: $p_1 = M^{-1} \cdot p_2$.

Assimile

Toda transformação geométrica de um conjunto de pontos é a execução de uma multiplicação de matrizes do tipo p2 = Mp1 para cada ponto do conjunto. Isso explica porque a computação gráfica exige um hardware dedicado somente ao processamento gráfico: a placa de vídeo.

A segunda transformação mais simples é a translação. A Figura 2.2 representa por uma seta a translação do ponto *(x1,y1)=(3,2)* de *(tx, ty) = (3,6)*, levando-o para o ponto *(x2,y2)=(6,8)*.

Figura 2.2 | Translação do ponto ***(3,2)*** de uma distância ***(3,6)***, levando-o para o ponto ***(6,8)***



Fonte: elaborada pelo autor.

A translação de um ponto *(x1,y1)* por *(tx,ty)*, se dá pelas equações

$$x_2 = x_1 + t_x$$
$$y_2 = y_1 + t_y$$

Que podem ser representadas na forma de matriz como

2D: $\begin{pmatrix} x_2 \\ y_2 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix}$,

3D: $\begin{pmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{pmatrix}$

Para a translação foi necessário acrescentar uma dimensão, fazendo com que a matriz de transformação 2D seja 3x3 e a matriz de transformação 3D seja 4x4. Da mesma forma podemos acrescentar uma dimensão na transformação de escala.

2D: $\begin{pmatrix} x_2 \\ y_2 \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix}$,

3D: $\begin{pmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{pmatrix}$

De forma que toda matriz de transformação, seja de escala, translação ou rotação, sejam de dimensão 3x3 no caso 2D e 4x4 no caso 3D. Assim podemos finalmente definir as matrizes de escala e translação por

2D: $S = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$ $T = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix}$

3D: $S = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ $T = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$

Um exemplo de código da translação 2D em Python é apresentado a seguir, conforme Figura 2.2 O ponto *po* é o ponto original, *pt* é o ponto transladado.

```
import numpy

def matrizT2d(tx,ty):
    return numpy.array([[1,0,tx],
                        [0,1,ty],
                        [0,0,1]])

po = numpy.array([[3],[2],[1]])
pt = numpy.matmul(matrizT2d(3,6),po)
```

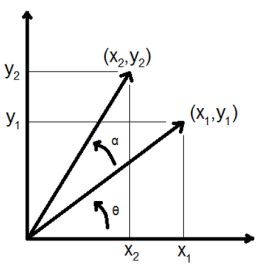
Para transformar um conjunto de pontos, basta construir uma função que recebe uma lista de pontos. O código a seguir implementa a transformação de escala da Figura 2.1.

```
import numpy

def matrizS2d(sx,sy):
    return numpy.array([[sx,0,0],
                        [0,sy,0],
                        [0,0,1]])

def afimN(m,pontos):
    ps = []
    for p in pontos:
        ps.append(numpy.matmul(m,p))
    return ps

m = matrizS2d(3,3)
po1 = numpy.array([[2],[1],[1]])
po2 = numpy.array([[3],[1],[1]])
po3 = numpy.array([[2],[2],[1]])
po4 = numpy.array([[3],[2],[1]])
pontos = [po1,po2,po3,po4]
ps = afimN(m,pontos)
```

Rotação

A rotação de um ponto bidimensional (x_1,x_2) por um ângulo α em torno da origem é representada na Figura 2.3.

Figura 2.3 | Rotação de (x_1,y_1) de um ângulo α em torno da origem, levando-o para o ponto (x_2,y_2)



Fonte: elaborada pelo autor.

Com base nos conceitos da geometria plana e relações trigonométricas (BOLDRINI et al. 1986; KREYSZIG, 2015), vemos que

$$x_2 = r\cos(\theta+\alpha) = r\cos\theta\cos\alpha - r\sin\theta\sin\alpha$$

$$y_2 = r\sin(\theta+\alpha) = r\sin\theta\cos\alpha + r\cos\theta\sin\alpha$$

mas $x_1 = r\cos\theta, y_1 = r\sin\theta$, então

$$x_2 = x_1\cos\alpha - y_1\sin\alpha$$

$$y_2 = x_1\sin\alpha + y_1\cos\alpha$$

Logo, a matriz de rotação 2D em torno da origem é

$$R_\alpha = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

A matriz R pode ser facilmente estendida para três dimensões para as rotações em torno de um dos três eixos. Por ser uma rotação em torno do eixo Z, a coordenada Z não é alterada. A matriz de rotação é igual à matriz 2D, apenas acrescida da dimensão Z.

$$R_{z\alpha} = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Da qual podemos derivar $R_{x\alpha}$ e $R_{y\alpha}$ de rotação em torno dos eixos X e Y, respectivamente.

$$R_{x\alpha} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad R_{y\alpha} = \begin{pmatrix} \cos\alpha & 0 & -\sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ \sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Em Python, a matriz de rotação 2D pode ser implementada com o código a seguir.

```
import math
import numpy
def matrizR(teta):
    return numpy.array([[math.cos(teta),-math.sin(teta),0],
                        [math.sin(teta),math.cos(teta),0],
                        [0,0,1]])
```

Como a rotação utiliza cálculos de senos e cossenos, o uso de números reais é necessário. No caso de imagens digitais, após cada rotação o ponto passa a ter coordenadas reais e precisa ser arredondado. Mesmo em imagens vetoriais, onde o ponto tem coordenadas reais ocorrem aproximações devido

à limitação da representação em ponto flutuante. Por este motivo, transformações geométricas cumulativas devem ser evitadas.

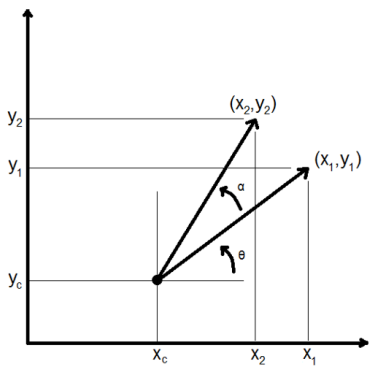
Exemplificando

Executar, na sequência, 3 rotações de 20 graus, uma após a outra, significa realizar as operações $p_2 = R_{20} p_1$, $p_3 = R_{20} p_2$, $p_4 = R_{20} p_3$, onde Rα é a matriz de rotação de α graus. No entanto, o ideal é sempre realizar transformações sobre o ponto original, na forma $p_2 = R_{20} p_1$, $p_3 = R_{40} p_1$, $p_4 = R_{60} p_1$ >>, como forma de reduzir o erro.

Composição de transformações

A rotação bidimensional em torno de um centro fora da origem pode ser obtida pela composição de transformações. A Figura 2.4 ilustra a rotação em torno do ponto $c=(x_c, y_c)$.

Figura 2.4 | Rotação em torno de um centro fora da origem



Fonte: elaborada pelo autor.

A matriz de rotação R_α não se aplica diretamente, mas pode ser usada em conjunto com duas translações, em três passos (GONZALEZ; WOODS, 2011; HUGHES et al., 2013):

Fazer a translação de $-c$, de forma que o ponto c fique coincidente com a origem;

Fazer a rotação pela matriz R_α, em torno da origem;

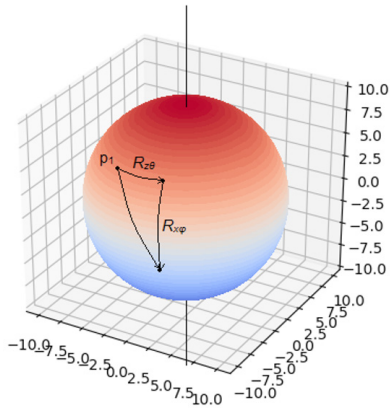
Fazer a translação de c, de forma que o ponto c volte ao lugar correto.

A matriz $R_{c\alpha}$, de rotação de α graus em torno do ponto c, é obtida pela multiplicação de matrizes

$R_{c\alpha} = T_c R_\alpha T_{-c}$. A ordem das matrizes é relevante na composição de transformações. A matriz $R_{c\alpha}$, por exemplo, é equivalente à aplicação das matrizes T_{-c}, R_α e T_c, nesta ordem. A composição de matrizes também será necessária na Seção 3 para a criação de modelos de câmera.

A rotação em torno de um eixo qualquer em 3D, que passe pela origem do sistema de coordenadas, é ilustrada na Figura 2.5.

Figura 2.5 | Rotação 3D, na forma de duas rotações em torno dos eixos *Z* e *X*



Fonte: elaborada pelo autor.

Considerando os códigos anteriores, podemos implementar a rotação do ponto *po=(4,3)* de *teta=2* radianos em torno do centro *c=(1,2)* com a seguinte composição de matrizes:

```
import math
import numpy
po = numpy.array([[4],[3],[1]])
(cx,cy) = (1,2)
teta = 2
m = numpy.matmul(matrizR2d(teta),matrizT2d(-cx,-cy))
m = numpy.matmul(matrizT2d(cx,cy), m)
ps = numpy.matmul(m,po)
```

A superfície da esfera de centro na origem e raio p_1 contém todos os possíveis resultados de rotações de p_1, quaisquer que sejam os ângulos e eixos de rotação que passem pela origem. É possível, no entanto, levar o ponto p_1 para qualquer outro ponto da esfera fazendo-se duas rotações em torno de dois eixos do sistema de coordenadas. Ao invés de fazer a rotação de um ângulo α em torno de um eixo arbitrário, faz-se a rotação $R_{z\theta}$, em torno do eixo Z, e uma rotação $R_{x\varphi}$, em torno do eixo X, conforme mostra a Figura 2.5.

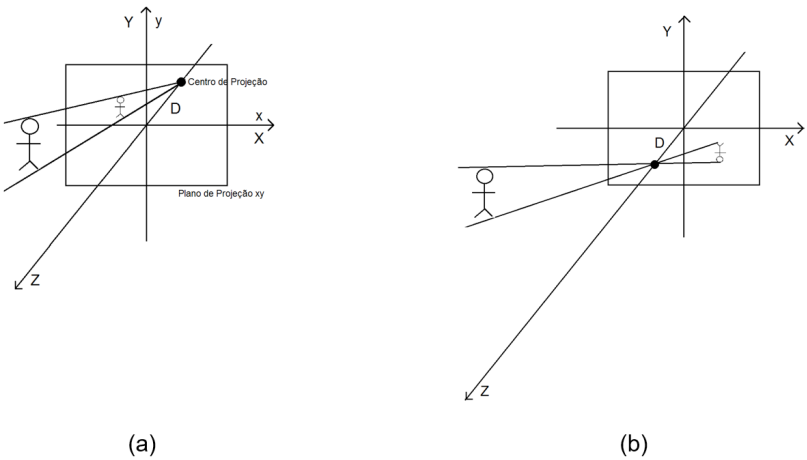
Reflita

Por trabalhar com números reais, as rotações exigem aproximações e apresentam erros. Ao fazer mais de uma rotação, o erro se propaga. A rotação 3D feita como duas rotações, em vez de uma, acumularia mais erro? Ou a aplicação sequencial de Rzθ e Rxϕ poderia ser substituída pela matriz equivalente $R_{\alpha} = R_{z\theta} \cdot R_{x\varphi}$, evitando o acúmulo de erros?

Transformação de perspectiva e projeção

A captura de imagens por meio de câmeras é uma projeção de uma cena no espaço 3D para uma imagem no espaço 2D. A Figura 2.6 ilustra a projeção.

Figura 2.6 | Projeção perspectiva



Fonte: elaborada pelo autor.

Um ponto da cena real 3D é um ponto do sistema de coordenadas *XYZ*. Um ponto do plano de projeção é um ponto do sistema de coordenadas *xy*. Qualquer ponto *p* da cena real é projetado sobre o plano de projeção no ponto de interseção do plano de projeção com reta que liga o ponto *p* ao centro de projeção. Na Figura 2.6(a) o centro de projeção está a uma distância D da origem do sistema de coordenadas *XYZ*, no ponto *(0,0,-D)*, e o plano *xy* é coincidente com o plano *XY*. Na Figura 2.6(b) o centro de projeção está no ponto *(0,0,D)*. Para estas configurações, as matrizes de transformação perspectiva que projeta o ponto *p* sobre o plano de projeção são

Centro em (0,0,-D): $P=\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&\frac{1}{D}&1\end{pmatrix}$, Centro em (0,0,D): $P=\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&-\frac{1}{D}&1\end{pmatrix}$

A configuração da Figura 2.6(a) é normalmente utilizada por desenhistas e na síntese de imagens, enquanto a configuração da Figura 2.6(b) representa a captura de uma câmera, sendo D a distância focal da lente. Nota-se que as matrizes de transformação são similares.

Quando a matriz P é aplicada, obtemos

$$\begin{pmatrix}X'\\Y'\\Z'\\H\end{pmatrix}=\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&\frac{1}{D}&1\end{pmatrix}\cdot\begin{pmatrix}X\\Y\\Z\\1\end{pmatrix}$$

O ponto *(X',Y',Z',H)* é um ponto em coordenadas homogêneas e representa, na verdade, toda a reta que liga o ponto *(X,Y,Z,1)* ao centro de projeção. Para encontrar o ponto *(x,y)*, no plano de projeção, é preciso fazer $x=\frac{X'}{H}$ e $y=\frac{Y'}{H}$.

Pesquise mais

A projeção em perspectiva é também estudada por artistas e desenhistas. Em computação gráfica e processamento de imagens a perspectiva ocorre na captura de imagens por câmeras ou na projeção de objetos 3D em telas. Ambos os casos são perspectiva com um ponto de projeção (ou ponto de fuga). Veja como os artistas utilizam esta perspectiva em (FUSSEL, 2015).

Todas as transformações geométricas apresentadas são implementadas na forma de multiplicações de matrizes. A implementação dessas transformações é trivial em qualquer linguagem de programação. Você pode começar a praticar já.

Sem medo de errar

O contexto em que você se encontra é o de uma empresa que está desenvolvendo um novo aplicativo de criação de animações 3D e sua equipe de

trabalho está encarregada do desenvolvimento da biblioteca de transformações geométricas. A primeira etapa do trabalho é a implementação das transformações geométricas básicas de escala, rotação, translação e perspectiva.

A solução para esta situação nada mais é do que a tradução direta dos conceitos de geometria e álgebra linear para o código na linguagem de programação escolhida. Uma proposta de solução para o problema é:

Defina a linguagem de programação e uma biblioteca com facilidades para a manipulação de matrizes. No caso da linguagem Python, o NumPy, já utilizado na Unidade 1, é uma excelente opção.

Para cada transformação a ser implementada, crie uma função com os parâmetros da transformação.

Como exemplo em Python, segue a implementação da função de rotação em torno da origem

```
import math
import numpy

def matrizR2d(teta):
    return numpy.array([[math.cos(teta),-math.sin(teta),0],
                        [math.sin(teta),math.cos(teta),0],
                        [0,0,1]])

def rotate(p,teta):
    return numpy.matmul(matrizR2d(teta),p)

p = numpy.array([[2],
                 [3],
                 [1]])
q = rotate(p,2)
```

Além de implementar a rotação em torno de um ponto fora da origem, também é interessante implementar a função que executa a mesma rotação para vários pontos, como o código a seguir.

```
def rotateN(ps,teta):
    R = matrizR2d(teta)
    qs = []
    for p in ps:
        q = numpy.matmul(R,p)
        qs.append(q)

    return qs
```

A implementação da função que executa uma mesma transformação geométrica para vários pontos é particularmente útil para transformações compostas. A matriz de transformação é construída uma única vez e aplicada a todos os pontos.

Avançando na prática

Projeção de modelos vetoriais

Descrição da situação-problema

Você foi contratado para implementar as projeções de modelos 3D de um jogo digital na tela do computador. O sistema de coordenadas do mundo do modelo é fixo e, inicialmente, o plano de projeção xy é coincidente com o plano XY do mundo. O jogador tem três formas de interação: rotação do plano de projeção em torno de Z, rotação do plano de projeção em torno de X e translação do plano de projeção. Você deve desenvolver as funções que redesenham as projeções após cada interação do jogador sobre o plano de projeção. Pense na melhor forma de executar diversas transformações geométricas em sequência. Seria melhor executar cada transformação requisitada pelo usuário sobre o resultado da transformação anterior, ou seria melhor manter sempre o modelo em suas coordenadas originais e atualizar a matriz de transformação geométrica, fazendo sempre a transformação do modelo original?

Resolução da situação-problema

A solução para essa situação-problema é uma complementação da implementação das transformações geométricas básicas. Primeiramente temos que considerar a informação de que a câmera inicialmente está com o plano de projeção coincidente com o plano XY do sistema de coordenadas do mundo do modelo. Neste caso, para projetar cada ponto do modelo sobre o plano de projeção, basta aplicar a matriz P de transformação de perspectiva.

Se o jogador interagir solicitando uma translação Tc do plano de projeção, a matriz P não poderá mais ser aplicada diretamente. Será necessário primeiro trazer o plano de projeção de volta ao estado inicial para só então aplicar P. Na prática será aplicada uma matriz de transformação M=PT-c.

Se depois da translação o jogador interagir solicitando uma rotação Rxα do plano de projeção, a matriz P continuará não podendo ser aplicada diretamente. Para trazer o plano de projeção de volta ao estado inicial é preciso agora fazer a rotação Rx(-α), em seguida aplicar a tranlação T-c para só então aplicar P. Na prática será aplicada uma matriz de transformação M= PT-c Rx(-α).

Note que a cada transformação do plano de projeção é acrescentada a matriz de transformação inversa à direita de M.

Finalmente, deve-se manter a matriz M sempre atualizada. A cada solicitação de movimentação da câmera, a matriz M é atualizada de forma a fazer o movimento reverso antes da transformação de perspectiva. A matriz M será sempre aplicada sobre os pontos originais, evitando o acúmulo de erros.

Faça valer a pena

1. Em um aplicativo interativo, o usuário solicitou por 4 vezes consecutivas que fosse realizada a rotação do objeto de 15 graus. O desenvolvedor tinha a opção de implementar rotações consecutivas usando duas possíveis estratégias. A primeira estratégia é implementar realmente de forma sequencial: $p_2 = R_{15}p_1$, $p_3 = R_{15}p_2$, $p_4 = R_{15}p_3$ e $p_5 = R_{15}p_4$, onde Rα é a matriz de rotação de α graus. A segunda estratégia é implementar sempre a rotação de p1, por ângulos cumulativos: $p_2 = R_{15}p_1$, $p_3 = R_{30}p_1$, $p_4 = R_{45}p_1$ e $p_5 = R_{60}p_1$.

Assinale a alternativa que justifica corretamente qual estratégia deve ser adotada pelo desenvolvedor.

a) A segunda estratégia é a melhor, pois o custo computacional de cada rotação é independente do ângulo de rotação, não havendo perdas neste sentido, mas com o acúmulo dos ângulos, evita-se o acúmulo de erros de aproximação.

b) A primeira estratégia é a melhor, pois o custo computacional de cada rotação é independente do ângulo de rotação, não havendo perdas neste sentido, mas as rotações por ângulos menores geram menos erros de aproximação.

c) O custo computacional de cada rotação é menor quanto maior for o ângulo de rotação. A segunda estratégia deve ser adotada porque será mais eficiente e executará mais rápido.

d) O custo computacional de cada rotação é menor quanto menor for o ângulo de rotação. A primeira estratégia deve ser adotada porque será mais eficiente e executará mais rápido.

e) A segunda estratégia é a melhor, pois o custo computacional de cada rotação é independente do ângulo de rotação, não havendo perdas neste sentido, mas as rotações por ângulos maiores geram menos erros de aproximação.

2. Para se executar uma rotação 2D em torno de um centro fora da origem do sistema de coordenadas é preciso fazer a composição de três transformações geométricas. Considere que você deseja fazer a rotação de um ponto por um ângulo α em torno do ponto c=(3,5).

Assinale a alternativa que apresenta a correta composição de matrizes para gerar a matriz de rotação correta para este caso.

a)
$$R_{c\alpha} = \begin{pmatrix} 1 & 0 & -3 \\ 0 & 1 & -5 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 5 \\ 0 & 0 & 1 \end{pmatrix}$$

b)
$$R_{c\alpha} = \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 5 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -3 \\ 0 & 1 & -5 \\ 0 & 0 & 1 \end{pmatrix}$$

c)
$$R_{c\alpha} = \begin{pmatrix} -\cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & -\cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 5 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

d)
$$R_{c\alpha} = \begin{pmatrix} 3 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} -3 & 0 & 0 \\ 0 & -5 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

e)
$$R_{c\alpha} = \begin{pmatrix} -\cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & -\cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 3 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

3. Considere que o sistema de coordenadas 3D do mundo real esteja fixo em um ponto e você esteja movimentando uma câmera no ambiente. A matriz de perspectiva P estudada se aplica apenas ao caso em que o sistema de coordenadas do plano de projeção é coincidente com o plano XY do mundo real. Considere então um ponto w no mundo real, a matriz P e o CCD da câmera como plano de projeção.

Assinale a alternativa correta a respeito da aplicação da matriz *P* sobre o ponto *w* para se obter as coordenadas de *w* no plano de projeção.

a) Não é possível fazer a projeção de w com o uso da matriz P. O sistema de coordenadas 3D não pode ser fixo. Deve sempre se mover junto com o sistema de coordenadas do plano de projeção.

b) A matriz P pode ser aplicada normalmente e a correção será feita quando o ponto em coordenadas homogêneas for convertido para o plano de projeção, com base na distância entre as origens dos sistemas de coordenadas.

c) A matriz P não pode ser aplicada à captura de imagens quando o ponto de projeção está à frente do plano de projeção, mas apenas à síntese de imagens, quando o ponto de projeção está atrás do plano de projeção.

d) É possível utilizar a matriz P, mas antes é preciso alinhar os dois sistemas de

coordenadas (mundo real e plano de projeção), usando rotações e translações, com raciocínio similar ao da rotação em torno de um ponto fora da origem.
e) A matriz P pode ser aplicada normalmente, pois o ponto obtido da transformação de perspectiva é em coordenadas homogêneas, e representa de fato toda a reta que liga o ponto w ao centro de projeção.

Seção 2.2

CGPI: Modelos geométricos

Diálogo aberto

Caro aluno, nesta seção serão apresentadas as formas de representação de objetos tridimensionais por meio de modelos geométricos. A representação de objetos tridimensionais, junto às transformações geométricas, forma a base para a síntese de imagens. Vale lembrar que a síntese de imagens e a computação gráfica estão hoje presentes no cotidiano de todas as pessoas, mesmo as mais leigas em termos de informática. A computação gráfica 3D está presente em filmes, efeitos de televisão, material de ensino, e até mesmo na interação das pessoas com o dispositivo que lhes acompanha a todo momento: o smartphone. Um mundo sem computação gráfica hoje é inimaginável.

Começaremos por apresentar a representação das primitivas geométricas em três dimensões: pontos, retas, curvas e polígonos. Em seguida você conhecerá a representação de superfícies de objetos tridimensionais por meio de pontos conectados, as chamadas malhas tridimensionais. Assim como para as imagens vetoriais 2D, serão apresentadas estruturas de dados para a representação dos modelos tridimensionais.

Vamos retomar a situação em que você faz parte de uma equipe de desenvolvimento da biblioteca de transformações geométricas de um novo aplicativo de criação de animações 3D. Tal biblioteca inclui não só as transformações geométricas básicas como também as representações de objetos tridimensionais. O resultado do seu trabalho pode não ser visível para o usuário final, mas é essencial para a equipe que desenvolverá as visualizações de objetos para o usuário final.

Na Seção 2.1 você já foi apresentado às transformações geométricas básicas de escala, rotação, translação e perspectiva. Agora você deve implementar os modelos geométricos tridimensionais, sobre os quais poderão ser aplicadas as transformações geométricas já desenvolvidas. Os conceitos das imagens vetoriais devem ser implementados em três dimensões. Você deverá implementar estruturas de dados para representar malhas e curvas paramétricas em 3D. Neste momento, ainda não são exigidos algoritmos de desenho ou exibição 2D das formas tridimensionais representadas, mas apenas a definição, em linguagem de programação, das formas descritas. Mesmo assim, você ainda deve se considerar parte de uma equipe de bases sólidas de geometria, que desenvolve as bibliotecas que serão utilizadas por outras equipes. Você deverá entregar código, em linguagem de programação,

implementando estruturas de dados que representem os elementos gráficos tridimensionais apresentados.

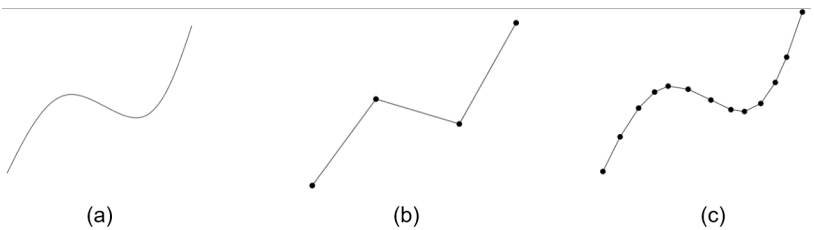
Bom trabalho!

Não pode faltar

Caro aluno, na Seção 2.1 foram apresentadas as transformações geométricas básicas de escala, translação, rotação e projeção perspectiva. Todas as transformações apresentadas são aplicadas sobre um ponto do espaço 2D ou do espaço 3D, contínuo ou discreto. Sabemos, no entanto, que as imagens vetoriais são descritores de formas geométricas, e não apenas um conjunto de pontos. Portanto, temos que levar em consideração que as transformações geométricas estudadas devem ser aplicadas sobre os descritores de imagens vetoriais. Nesta seção exploraremos tais descritores em três dimensões, o que nos permitirá construir modelos geométricos que representem objetos do mundo tridimensional que vivemos.

Vamos começar pela representação de formas geométricas além das retas e círculos apresentados até o momento. Primeiramente, vamos criar linhas não retas. Uma linha qualquer, desenhada sobre um plano, nem sempre pode ser descrita por uma equação trivial. A Figura 2.7(a) mostra uma linha com esta característica.

Figura 2.7 | Linhas e polilinhas



Fonte: elaborada pelo autor.

A linha da Figura 2.7(a) pode ser aproximada por uma sequência de segmentos de reta. A Figura 2.7(b) mostra uma aproximação utilizando 3 segmentos de reta e a Figura 2.7(c) mostra uma aproximação utilizando 12 segmentos de reta. É possível observar que a utilização de um número maior de segmentos de reta (Figura 2.7(c)) resulta em uma melhor aproximação da linha contínua original representada na Figura 2.7(a).

As linhas apresentadas nas Figuras 2.7(a) e (b) são denominadas polilinhas

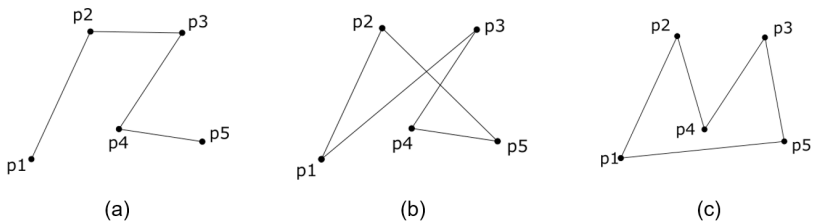
(HUGHES et al., 2013). Uma polilinha é um conjunto de vértices conectados por segmentos de reta (arestas). Com polilinhas é possível desenhar diversas formas geométricas, inclusive polígonos. A Figura 2.8 mostra exemplos de formas geométricas representadas por polilinhas. A Figura 2.8(c) mostra um polígono construído por uma polilinha.

As três formas apresentadas na Figura 2.8 podem ser representadas em linguagem de programação como sequências de vértices, ordenadas. Em Python podemos escrever da seguinte forma (considere já declarados os vértices, por exemplo, como p1=(x1,y1)):

```
polilinhaA = [p1, p2, p3, p4, p5]
polilinhaB = [p1, p2, p5, p4, p3, p1]
polilinhaC = [p1, p2, p4, p3, p5, p1]
```

As polilinhas das Figura 2.8(a), (b) e (c) são representadas pela lista *vértices* e pelas listas de arestas *arestasA*, *arestasB* e *arestasC*, respectivamente.

Figura 2.8 | Formas geométricas representadas por polilinhas



Fonte: elaborada pelo autor.

As polilinhas são aproximações de curvas, mas se conhecermos a equação da curva, podemos no espaço contínuo pôr sua equação. Desta forma a curva poderá ser desenhada em qualquer resolução espacial, por algoritmos similares aos de desenhos de reta e círculo.

A curva da Figura 2.7(a) é uma curva descrita por equações e por isso pôde ser desenhada pixel a pixel, mantendo a aparência de uma curva contínua. As curvas mais utilizadas são curvas paramétricas denominadas *splines* (HEARN et al., 2013).

Splines

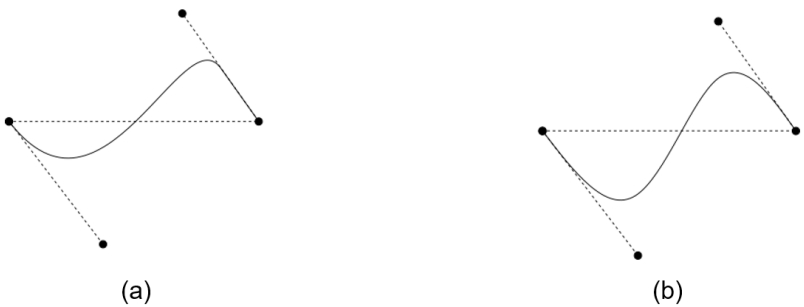
Uma *spline* é qualquer curva composta formada por secções polinomiais satisfazendo algum critério de continuidade nas junções das secções (HEARN et al., 2013). Enquanto as polilinhas são sequências de segmentos de reta, as *splines* são sequências de curvas polinomiais.

Primeiramente vamos especificar uma única secção polinomial. Um polinômio de ordem três é descrito pela equação $f(t) = at^3 + bt^2 + ct + d$, que pode ser reescrita na forma matricial

$$f(x) = [d,c,b,a]\begin{bmatrix}1\\t\\t^2\\t^3\end{bmatrix} = C \cdot T$$

onde T são as potências do parâmetro t e C é a matriz de coeficientes. C pode ser escrita como $C = G \cdot M$, onde G é a matriz geometria, contendo as restrições geométricas da *spline*, e M é a matriz base, uma matriz *4x4* que transforma as restrições geométricas em coeficientes e provê uma caracterização da *spline* (HUGHES et al., 2013). A Figura 2.9 apresenta duas *splines* com as mesmas restrições geométricas (G) e caracterização diferente (M). Finalmente, a equação de uma *spline* é dada por $f(t) = G \cdot M \cdot T$.

Figura 2.9 | Exemplos de ***splines*** de características diferentes e restrições geométricas iguais: (a) curva de Bézier; (b), B-Spline. A linha pontilhada mostra a ordem dos pontos de controle.



Fonte: elaborada pelo autor.

Curvas de Bézier

Curvas de Bézier são *splines* com uma característica específica. Foram propostas por Pierre Bézier para o design de automóveis (HEARN et al., 2013). As curvas de Bézier são muito utilizadas para descrever curvas suaves obtidas a partir de pontos de referência, denominados **pontos de controle**. Os pontos de controle de uma curva de Bézier formam a geometria da *spline* (G), e a matriz base da *spline* (M), que define a característica da curva polinomial, é obtida a partir da definição polinomial dessa *spline*.

A definição polinomial de uma Curva de Bézier é dada por $B(t)=\sum_{i=0}^{n}\binom{n}{i}(1-t)^{n-i}t^{i}P_i, 0\le t\le 1$, onde $\binom{n}{i}=\frac{n!}{i!(n-i)!}$ é o coeficiente binomial e P_i são os pontos de controle.

Pesquise mais

Animações de desenhos de curvas de Bézier com até 5 pontos de controle. (DAVIES, 2010)

DAVIES, J. Animated Bézier curves.

Para uma Curva de Bézier cúbica, ou seja, *n*=3, por exemplo, o polinômio acima se torna

$B(t)=(1-t)^3P_0+3t(1-t)^2P_1+3t^2(1-t)P_2+t^3P_3, 0\le t\le 1$, ou, na forma matricial,

$$B(t)=[P_0;P_1;P_2;P_3]\cdot\begin{bmatrix}1&-3&3&-1\\0&3&-6&3\\0&0&3&-3\\0&0&0&1\end{bmatrix}\cdot T$$

onde $G=[P_0;P_1;P_2;P_3]$ são os pontos de controle, sendo que cada ponto é uma coluna com suas coordenadas no plano (*spline* 2D) ou no espaço (*spline* 3D). A Figura 2.9(a) é uma curva de Bézier de ordem 3, para os quatro pontos de controle nela representados.

Exemplificando

Consideremos como segundo exemplo uma curva de Bézier quadrática. Sua equação é $B(t)=\sum_{i=0}^{2}\binom{2}{i}(1-t)^{2-i}t^{i}P_i$, ou $B(t)=(1-t)^2P_0+2t(1-t)P_1+t^2P_2$, ou ainda, na forma matricial, $B(t)=[P_0;P_1;P_2]\cdot\begin{bmatrix}1&-2&1\\0&2&-2\\0&0&1\end{bmatrix}\cdot\begin{bmatrix}1\\t\\t^2\end{bmatrix}$. A equação da curva de Bézier de ordem 4 é $B(t)=\sum_{i=0}^{4}\binom{4}{i}(1-t)^{4-i}t^{i}P_i$, ou $B(t)=(1-t)^4P_0+4t(1-t)^3P_1+6t^2(1-t)^2P_2+4t^3(1-t)P_3+t^4P_4$. Experimente expressar este último polinômio em forma matricial e também as equações de curvas de Bézier de outras ordens.

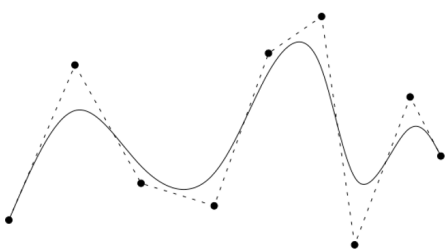
B-Splines

B-Splines são, junto às curvas de Bézier, as *splines* mais usadas. Assim como as curvas de Bézier, B-Splines são geradas pela aproximação de um conjunto de pontos de controle. Curvas B-Splines são mais complexas que curvas de Bézier, mas possuem algumas vantagens. Uma vantagem é que o grau do polinômio da B-Spline pode ser definido independentemente do número de pontos de controle (com algumas limitações). Veremos aqui apenas as B-Splines cúbicas. A função base de uma B-Spline cúbica é

$$b_3(t) = \frac{1}{6}t^3, 0 \leq t \leq 1$$
$$b_3(t) = \frac{1}{6}(-3(t-1)^3 + 3(t-1)^2 + 3(t-1) + 1), 1 \leq t \leq 2$$
$$b_3(t) = \frac{1}{6}(t-2)^3 - 6(t-2)^2 + 4), 2 \leq t \leq 3$$
$$b_3(t) = \frac{1}{6}(-(t-3)^3 + 3(t-3)^2 - 3(t-3) + 1), 3 \leq t \leq 4$$
$$b_3(t) = 0, \textit{senão}$$

e a equação da B-Spline cúbica com pontos de controle P_0, ..., P_n é $B_s(t) = \sum_{i=0}^{n} P_i b_3(t-i)$. A Figura 2.10 apresenta uma B-Spline cúbica com 9 pontos de controle.

Figura 2.10 | B-Spline cúbica com 9 pontos de controle. A linha pontilhada mostra a sequência dos pontos de controle.



Fonte: elaborada pelo autor.

Curvas de Bézier e B-Splines podem ser construídas sobre o plano, mas também no espaço tridimensional, seguindo as mesmas equações. No espaço tridimensional, no entanto, esses descritores de curvas são estendidos para descrever superfícies. São as superfícies de Bézier e superfícies B-Spline. Para um aprofundamento sobre essas superfícies, consulte (HEARN et al., 2013; HUGHES et al., 2013).

Assimile

Os pontos de controle de curvas de Bézier e B-Splines podem ser melhor compreendidos com o auxílio de um aplicativo de desenho vetorial. Utilize o aplicativo livre Inkscape (INKSCAPE, 2018) para desenhar tais curvas:

1. Selecione a ferramenta "Caneta Bézier".
2. Clique sobre a tela de desenho no ponto inicial da curva (P_0).
3. Movimente o cursor livremente. Você verá um segmento de reta sendo desenhado em vermelho.
4. Clique em um segundo ponto e arraste. À medida que você arrasta o cursor, você verá, em vermelho, uma curva polinomial de ordem três ao mesmo tempo que visualizará um segmento de reta em azul. As extremidades do segmento azul são os pontos de controle P_1 e P_2.
5. Solte o botão do mouse. Você verá agora um polinômio em verde e o segmento de reta fixo em azul, ligando os pontos P_1 e P_2.
6. Movimente novamente o cursor livremente. Você verá um segundo polinômio em vermelho.
7. Faça duplo clique no ponto final (P_3). Você irá obter a curva de Bézier para os pontos de controle P_0, P_1, P_2 e P_3.
8. Para desenhar uma B-Spline, altere o modo da ferramenta "Caneta Bézier" na barra de ferramentas do alto da tela à esquerda.

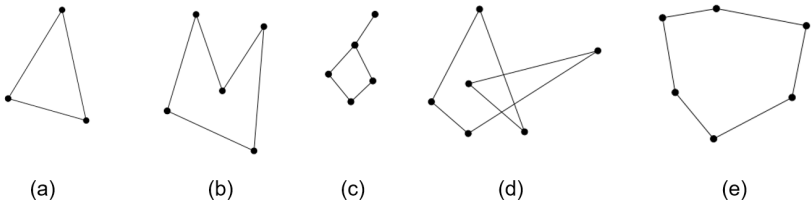
A representação em memória de curvas de Bézier e B-Splines nada mais é do que uma sequência de vértices (pontos de controle), assim como as polilinhas.

Poliedros e malhas

As polilinhas foram definidas sobre o plano e com elas foi possível desenhar polígonos. Sem restrições, polilinhas se estendem para três dimensões, desenhando-se uma sequência de segmentos de reta em três dimensões.

Para estender os polígonos para três dimensões é preciso fazer algumas considerações. A Figura 2.11 apresenta cinco diferentes polígonos. Os polígonos (a), (b) e (e) da Figura 2.11 são **polígonos simples**, pois não há interseção entre nenhum dos segmentos que os formam. Os polígonos (c) e (d) são não-simples. Os polígonos (a) e (e) são ainda **polígonos convexos**, pois qualquer segmento de reta desenhado entre dois pontos de suas bordas está inteiramente no interior do polígono (HUGHES et al., 2013).

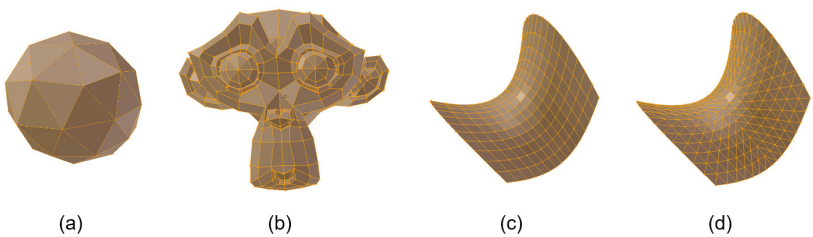
Figura 2.11 | Diferentes tipos de polígonos



Fonte: elaborada pelo autor.

Em 3D temos o interesse de construir sólidos ou superfícies. Em duas dimensões, uma curva contínua é aproximada por uma curva mais simples, a polilinha. Em 3D os sólidos contínuos são aproximados por sólidos mais simples: os **poliedros**. O poliedro é um sólido tridimensional cujas faces são polígonos planares (BERG et al., 2008; ANGEL; SHREINER, 2012). A superfície de um poliedro é uma superfície fechada, que separa o interior do poliedro de seu exterior. Mas uma superfície contínua aberta também pode ser aproximada por um conjunto de polígonos planares. A superfície, aberta ou fechada, composta apenas por polígonos planares é denominada **malha poligonal**. A Figura 2.12 mostra algumas malhas poligonais.

Figura 2.12 | Poliedros e malhas: (a) aproximação da esfera por um poliedro de faces triangulares; (b) macaco desenhado por malha poligonal com diferentes polígonos; (c) cela desenhada por quadriláteros e (d) a mesma cela (c) desenhada por triângulos.



Fonte: elaborada pelo autor.

Representação de malhas em memória

As malhas poligonais são compostas de polígonos planares e podem conter polígonos diversos, como na Figura 2.12(b), que contém triângulos e quadriláteros. Há preferência de uso de polígonos convexos, devido a uma maior facilidade de aplicação das operações necessárias para a exibição do modelo tridimensional em uma tela bidimensional (BERG et al., 2008).

Mesmo que a malha tenha sido criada com o uso de polígonos planares não convexos de diversos lados, é possível transformá-la em malha triangular pela subdivisão dos polígonos em triângulos, como foi feito com a

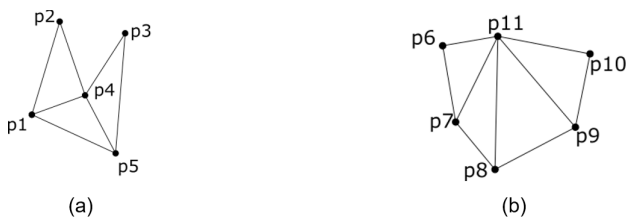
Figura 2.12(c) para gerar a Figura 2.12(d) (OLIVEIRA, 2015). As malhas triangulares são as mais utilizadas por dois motivos: i) simplicidade na representação da malha em memória e ii) o fato de que quaisquer três pontos diferentes no espaço 3D formam um polígono planar convexo, o triângulo, o que não é garantido para polígono com mais lados. Por esses motivos básicos as malhas triangulares já foram extensivamente estudadas.

Reflita

Considere um quadrilátero planar de vértices tridimensionais P_1, P_2, P_3 e P_4. Se você deseja modificar a coordenada do vértice P_4, existe garantia de que o quadrilátero resultante seja planar? Tomemos como exemplo um quadrilátero sobre o plano $z=z_q$. As coordenadas dos vértices seriam $P_1=(x_1,y_1,z_q)$, $P_2=(x_2,y_2,z_q)$, $P_3=(x_3,y_3,z_q)$ e $P_4=(x_4,y_4,z_q)$. Se alterarmos a posição do vértice P_4 para a coordenada (x_4,y_4,z_4), $z_4 \neq z_q$, o quadrilátero resultante será planar? Existiria um plano que inclui os quatro vértices do quadrilátero resultante? Desenvolva o mesmo raciocínio para o triângulo.

Há diversas estruturas de dados possíveis para a representação em memória de uma malha triangular, e a eficiência dos algoritmos de manipulação de malhas depende da estrutura de dados utilizada (OLIVEIRA, 2015). Veremos a seguir a estrutura de dados mais intuitiva, que se assemelha ao que foi apresentado para as polilinhas. Para ilustrar, a Figura 2.13 apresenta triangulações dos polígonos apresentados nas Figuras 2.11(b) e (e).

Figura 2.13 | Malhas triangulares construídas a partir de polígonos da Figura 2.11.



Fonte: elaborada pelo autor.

A malha triangular pode ser representada por duas listas: uma lista de vértices e uma lista de triângulos, como mostra o código em Python a seguir:

```
verticesA = [p1, p2, p3, p4, p5]
tA1 = [p1, p2, p4]
tA2 = [p4, p3, p5]
tA3 = [p1, p4, p5]
triangulosA = [tA1, tA2, tA3]
malhaA = (verticesA, triangulosA)

verticesB = [p6, p7, p8, p9, p10, p11]
tB1 = [p6, p7, p11]
tB2 = [p7, p8, p11]
tB3 = [p8, p9, p11]
tB4 = [p9, p10, p11]
triangulosB = [tB1, tB2, tB3, tB4]
malhaB = (verticesB, triangulosB)
```

O código acima é bastante simplificado e o mesmo formato de definição de malhas permite a criação de malhas poligonais quaisquer, pois os triângulos são simples listas. Para garantir que a malha seja triangular, o ideal é criar uma classe cujas operações restrinjam as listas que definem triângulos a apenas três vértices que façam parte da lista de vértices da malha.

Com o conteúdo desta seção você já é capaz de compreender as *splines* e as malhas poligonais em três dimensões, além de representar essas formas geométricas em memória. Foram apresentadas as *splines* e malhas mais simples e mais utilizadas, mas a base apresentada permite a compreensão de *splines* e malhas mais complexas. A exibição dessas formas será objeto de estudo das próximas seções.

```
class Polilinha2D:

    def __init__(self, vertices):
        self.vertices = vertices

class BezierCubica2D:

    def __init__(self, p1, p2, p3, p4):
        self.controlPoints = [p1,p2,p3,p4]

class BSplineCubica2D:

    def __init__(self, controlPoints):
        self.controlPoints = controlPoints

class MalhaTriangular:

    def __init__(self):
        self.vertices = set()
        self.triangulos = []

    def adicionaTriangulo(self,p1,p2,p3):
        self.vertices.update([p1,p2,p3])
        self.triangulos.append([p1,p2,p3])

    def __repr__(self):
        s = "Vertices: " + str(self.vertices) + "\n"
        s += "Triangulos: " + str(self.triangulos)
        return s
```

Note que na implementação da classe MalhaTriangular foi usado o tipo *set*, que é um conjunto não ordenado de elementos, assim não haverá vértices repetidos. Outra observação importante é que no código proposto acima foram implementadas especificamente curvas de Bézier e B-Splines cúbicas bidimensionais. É possível generalizar essas classes, mas isto poderia complicar os algoritmos de desenho que virão no futuro completar a implementação dessas classes.

Para aplicar uma transformação geométrica básica sobre qualquer uma das formas acima, basta aplicar a transformação para cada vértice ou ponto de controle da forma. A solução deste problema permite, portanto, continuar a demonstrar a capacidade de aplicar transformações geométricas, agora sobre formas tridimensionais.

Avançando na prática

Malhas poligonais contendo polígonos diversos

Descrição da situação-problema

Caro aluno, apesar de terem sido apresentadas as malhas poligonais genéricas, foi dada ênfase nas malhas triangulares, inclusive com sugestão de código em Python para representar uma malha triangular. Você agora foi designado para implementar uma classe que representa uma malha poligonal genérica, que aceita não só triângulos, mas também outros polígonos. Os outros polígonos aceitos, no entanto, continuam sendo polígonos convexos simples e planares. Na sua solução você já deve indicar, na forma de comentário, os pontos onde devem ser acrescentados os devidos testes para verificar se um polígono que está sendo adicionado à malha é convexo, simples e planar. Não devem ser implementados os testes. Você deve entregar código em linguagem de programação com a implementação da classe.

Resolução da situação-problema

A solução para esta situação-problema é uma generalização da classe MalhaTriangular apresentada na solução da situação-problema anterior. A primeira mudança a ser feita é a substituição de qualquer referência a triângulos por referências a polígonos. A mudança mais importante, porém, é no método de adição de polígono, que deve acrescentar os testes que verificam se o polígono sendo adicionado é convexo, simples e planar. Uma proposta de código a ser entregue é apresentada a seguir.

```
class MalhaPoligonal:

    def testaSimplesConvexoPlanar(vertices):
        # implementação dos testes deve ser inserida aqui
        return True

    def __init__(self):
        self.vertices = set()
        self.poligonos = []

    def adicionaPoligono(self,vertices):
        if (testaSimplesConvexoPlanar(vertices)):
            self.vertices.update(vertices)
            self.poligonos.append(vertices)

    def __repr__(self):
        s = "Vertices: " + str(self.vertices) + "\n"
        s += "Poligonos: " + str(self.poligonos)
        return s
```

É possível notar que não há grandes complicações em termos de estruturas de dados para a representação de malhas poligonais genéricas. Os problemas estão nas verificações das características dos polígonos.

Faça valer a pena

1. Polilinhas são linhas formadas por sequências de segmentos de reta. As polilinhas são usadas para aproximar curvas contínuas e também para descrever outras formas geométricas. Um polígono pode ser descrito por uma polilinha cujo ponto final é igual ao ponto inicial. A polilinha não está restrita a duas dimensões, podendo ser também utilizada em três dimensões para representar poliedros.

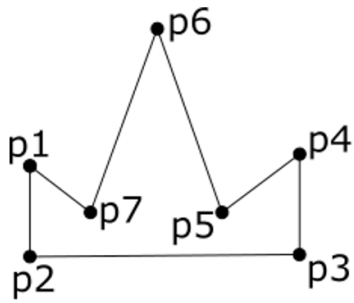
Sobre as polilinhas, polígonos e poliedros, assinale a alternativa correta.
a) Com polilinhas só é possível desenhar polígonos convexos simples. Os polígonos não simples ou não convexos não podem ser desenhados.
b) A estrutura de dados para armazenar uma polilinha é bastante complexa, pois precisa armazenar as inclinações de todos os segmentos de reta que a compõem.
c) Em 3D o número máximo de vértices que garante que o polígono por eles formado seja simples, convexo e planar é três.
d) A estrutura de dados que descreve uma polilinha é mais simples do que aquela que descreve uma B-Spline, pois a B-Spline precisa armazenar um número enorme de pontos.
e) Os poliedros são sólidos cujas faces são polígonos necessariamente convexos e não necessariamente planares.

2. Uma *spline* é qualquer curva composta formada por secções polinomiais satisfazendo algum critério de continuidade nas junções das secções. São curvas paramétricas, ou seja, curvas contínuas definidas por poucos parâmetros. As *splines* mais utilizadas são as Curvas de Bézier e as B-Splines.

Sobre as curvas de Bézier e as B-Splines, assinale a alternativa correta.
a) Curvas de Bézier e B-Splines são *splines* com características idênticas, mas com restrições geométricas diferentes.
b) O número de pontos de controle de uma Curva de Bézier define a ordem do seu polinômio, o que não acontece com a B-Spline.
c) Curvas de Bézier e B-Splines são definidas exclusivamente para o desenho no plano e não são extensíveis para 3D.
d) As B-Splines são mais utilizadas do que as Curvas de Bézier porque suas equações são mais simples.
e) B-Splines e curvas de Bézier são sequências de segmentos de reta, utilizadas para a criação de polígonos convexos simples.

3. Todo polígono simples, convexo ou não, pode ser subdividido em polígonos menores convexos e simples. Considere o polígono da Figura 2.14.

Figura 2.14 | Polígono simples não convexo.



Fonte: elaborada pelo autor.

Esse polígono precisa ser subdividido em triângulos para ser representado por uma malha triangular. Em Python, a malha triangular é declarada por

```
vertices = [p1, p2, p3, p4, p5, p6, p7]
t1 = ...
t2 = ...
t3 = ...
t4 = ...
t5 = ...
triangulos = [t1, t2, t3, t4, t5]
malha = (vértices,triângulos)
```

onde t1, t2, t3, t4 e t5 são os triângulos que compõem o polígono e precisam ser completados.

Assinale a alternativa que apresenta o código de uma correta triangulação do polígono da Figura 2.14.

a)
```
t1 = [p1, p2, p7]
t2 = [p1, p6, p7]
t3 = [p5, p6, p7]
t4 = [p4, p5, p6]
t5 = [p3, p4, p5]
```

b)
```
t1 = [p3, p5, p7]
t2 = [p1, p6, p7]
t3 = [p5, p6, p7]
t4 = [p4, p5, p6]
t5 = [p3, p4, p5]
```

c)
```
t1 = [p1, p2, p7]
t2 = [p2, p3, p7]
t3 = [p5, p6, p7]
t4 = [p3, p5, p7]
t5 = [p3, p4, p5]
```

d)
```
t1 = [p1, p2, p7]
t2 = [p2, p3, p7]
t3 = [p5, p6, p7]
t4 = [p2, p3, p5]
t5 = [p3, p4, p5]
```

e)
```
t1 = [p1, p2, p7]
t2 = [p1, p6, p7]
t3 = [p2, p3, p7]
t4 = [p4, p5, p6]
t5 = [p3, p4, p5]
```

Seção 2.3

CGPI: Modelo de câmera

Diálogo aberto

Caro aluno, estamos avançando nos conceitos da geometria do processamento gráfico. Estudamos até agora os conceitos de modelagem geométrica do ponto de vista da criação e representação do modelo. Esta seção aborda os conceitos relacionados à visualização de uma cena ou modelo 3D. Os conceitos de visualização são embasados em **modelos de câmeras**. O **modelo de câmera** é o que descreve a transformação geométrica que projeta um ponto do mundo real sobre o aparato de uma câmera que pode se mover.

Em um jogo digital de futebol, por exemplo, a todo momento o jogador enxerga o estádio de um ponto de vista diferente, com base no movimento da câmera. Mesmo que as linhas do campo estejam paradas, o jogador as enxerga de forma diferente para cada posição da câmera. A movimentação da câmera é uma sequência de translações e rotações, que, junto à projeção perspectiva, formam o modelo de câmera.

No contexto do desenvolvimento de um novo aplicativo de criação de animações 3D, em que você está inserido, o modelo de câmera será usado para a síntese de imagens. Uma animação é uma sequência de imagens que representam a projeção de objetos 3D em uma câmera, estática ou em movimento.

No aplicativo, o usuário será capaz de visualizar e transformar modelos geométricos e de definir o ponto de vista do espectador da animação, posicionando câmeras virtuais. Lembre-se de que sua equipe está encarregada do desenvolvimento da biblioteca de transformações geométricas, e que o usuário não terá contato direto com o resultado do seu trabalho. Você já implementou ferramentas de transformações geométricas básicas e de representação de modelos 3D, e deve agora incluir funcionalidades que permitam o posicionamento de câmeras virtuais.

Você deverá implementar modelos de câmera que utilizarão as transformações geométricas básicas e poderão ser aplicados sobre os modelos geométricos. Ainda se posicionando como um membro da equipe que detém sólidos conhecimentos de geometria, você deverá implementar funções que executem as transformações geométricas necessárias para que um modelo de câmera seja aplicado tanto para problemas de visão computacional, em que as imagens capturadas por uma câmera devem ser interpretadas, quanto para problemas de síntese de imagens, onde o modelo de câmera é usado

para sintetizar, a partir de modelos tridimensionais, imagens bidimensionais a serem exibidas ao usuário. As implementações dos modelos de câmera deverão incluir algoritmos de projeção de objetos 3D em telas 2D. Deverá ser entregue código em linguagem de programação.

Bons estudos!

Não pode faltar

Caro aluno, nesta seção estudaremos o modelo de câmera e técnicas adicionais de visualização e análise de imagens baseadas nesse modelo. Os conceitos de modelagem geométrica até o momento foram estudados do ponto de vista da criação e representação do modelo. Vamos agora estudar os conceitos geométricos relacionados à visualização de uma cena ou modelo 3D. Os conceitos de visualização são embasados em modelos de câmeras.

A câmera é um dispositivo de captura de imagens baseado em ótica, com um aparato de projeção denominado *Charge Coupled Device* – CCD, que possui sensores de luminosidade e gera uma saída numérica: a imagem digital (BOYLE; SMITH, 1970). A projeção de um ponto do mundo real é uma transformação perspectiva, conforme estudado na Seção 2.1. A transformação estudada não Seção 2.1 só é válida quando os sistemas de coordenadas da câmera e do mundo real são coincidentes. Nas aplicações reais, uma câmera é, a todo momento, transladada ou rotacionada. Neste caso, várias transformações ocorrem. O **modelo de câmera** é o que descreve a transformação geométrica que projeta um ponto do mundo real sobre o aparato de uma câmera que pode se mover (DAVIES, 2012).

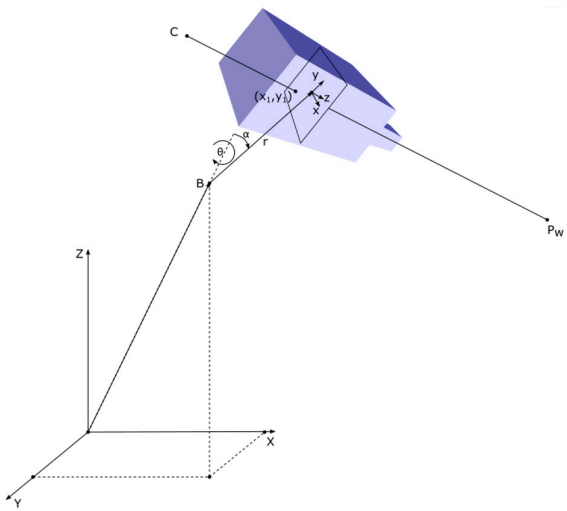
A transformação geométrica que projeto um ponto do mundo real sobre o CCD de uma câmera que pode ser transladada e rotacionada é obtida por um **modelo de câmera** (DAVIES, 2012). Tal modelo é usado em síntese de imagens, para a visualização de modelos tridimensionais, em visão computacional, para o reconhecimento de objetos em imagens digitais, e também em modelagem tridimensional, para construir automaticamente modelos tridimensionais a partir de imagens capturadas por câmeras.

A Figura 2.15 apresenta uma ilustração das transformações geométricas aplicadas sobre a câmera antes que ocorra a transformação perspectiva. A figura ilustra o sistema de coordenadas do mundo real (X,Y,Z) e o sistema de coordenadas da câmera (x,y,z). O ponto P_w tem coordenadas (X_1,Y_1,Z_1) no sistema de coordenadas do mundo real, e é projetado sobre o CCD da câmera no ponto $(x_1,y_1,0)$, que está no sistema de coordenadas da câmera. Trata-se de uma projeção perspectiva com centro de projeção no ponto C, que também existe apenas no sistema de coordenadas do mundo real.

Como não há coincidência dos sistema (x,y,z) da câmera com o sistema (X,Y,Z) do mundo real, não é possível aplicar a transformação perspectiva apresentada na Seção 2.1. É preciso, antes, fazer translações e rotações do ponto P_w para que sua posição relativa ao sistema (x,y,z) seja idêntica à posição relativa ao plano (X,Y,Z). Primeiramente, é preciso fazer a translação T_B, para que o ponto B coincida com a origem do sistema de coordenadas do mundo real.

$$T_B = \begin{pmatrix} 1 & 0 & 0 & -X_B \\ 0 & 1 & 0 & -Y_B \\ 0 & 0 & 1 & -Z_B \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Figura 2.15 | Modelo de câmera



Fonte: elaborada pelo autor.

Em seguida, é preciso fazer uma rotação $R_{X\alpha}$, em torno de X, e uma rotação $R_{Z\theta}$, em torno de Z, para que os eixos X e x fiquem paralelos e na mesma direção, assim como os eixos Y e y, e os eixos Z e z.

$$R_{x\alpha} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad R_{z\theta} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Uma translação T_r coloca as origens de (x,y,z) e de (X,Y,Z), coincidentes e, consequentemente, coloca os dois sistemas de coordenadas coincidentes. Assim é possível aplicar a transformação perspectiva P, onde D é a distância da origem do sistema (x,y,z) ao ponto C (ANGEL; SHREINER, 2012).

$$T_r = \begin{pmatrix} 1 & 0 & 0 & -(X_r - X_B) \\ 0 & 1 & 0 & -(Y_r - Y_B) \\ 0 & 0 & 1 & -(Z_r - Z_B) \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad P = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{D} & 1 \end{pmatrix}$$

Finalmente, para obter a coordenada do ponto P_w, projetado sobre o CCD, aplica-se a transformação geométrica a seguir:

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ H \end{bmatrix} = P \cdot T_r \cdot R_{Z\theta} \cdot R_{X\alpha} \cdot T_B \cdot \begin{bmatrix} X_P \\ Y_P \\ Z_P \\ 1 \end{bmatrix},$$

lembrando que o ponto resultante está em coordenadas homogêneas. Observe também que, para fazer a projeção de todos os pontos de um objeto sobre o plano do CCD, computa-se, antes, a matriz $M_c = P \cdot T_r \cdot R_{Z\theta} \cdot R_{X\alpha} \cdot T_B$, para então aplicá-la a cada ponto do objeto.

Assimile

A transformação geométrica que projeta um objeto, composto por diversos pontos, sobre um plano, é uma matriz 4x4 $M_c = P \cdot T_r \cdot R_{Z\theta} \cdot R_{X\alpha} \cdot T_B$, obtida pela composição das transformações básicas estudadas na Seção 2.1. Não é necessário criar equações específicas para cada projeção.

Modelo de câmera para síntese de imagens e para visão computacional

O modelo de câmera é aplicado de forma diferente nas subáreas de síntese de imagens ou de visão computacional. Em síntese de imagens, faz-se a projeção de um modelo tridimensional sobre o plano. O modelo geométrico é construído sobre um sistema de coordenadas de mundo real definido. No caso de modelos geométricos, o mundo real do modelo de câmera é na verdade virtual, assim como o próprio modelo. Da mesma forma, a posição (translações) e direcionamento (rotações) da câmera são definidos com base no mesmo sistema de coordenadas. A aplicação da matriz M_c acima é direta.

Em visão computacional, por sua vez, busca-se extrair informações do mundo real pelo processamento de imagens digitais, capturadas do mundo real por câmeras de posição e direcionamento desconhecidos. As coordenadas do mundo real, neste caso, são desconhecidas. O que se conhece são as projeções dos objetos do mundo real sobre telas 2D, as imagens digitais. Em visão computacional, portanto, o desafio é descobrir, com base em imagens digitais, quais são os elementos a_{ij} da matriz M_c, na forma

$$M_c = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

Trata-se do problema de calibração automática de câmera (JAIN et al., 1995).

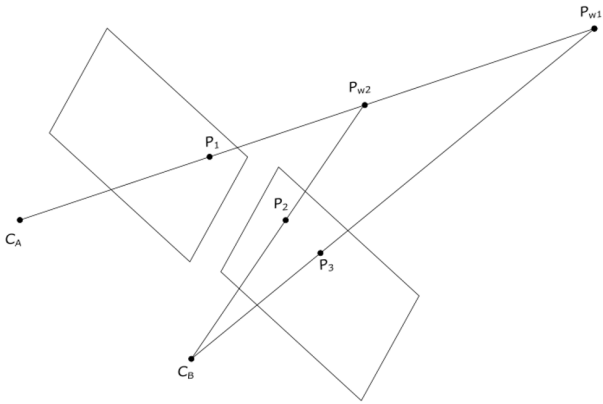
Reflita

Considere a mátria M_c genérica, com os coeficientes a_{ij} desconhecidos. O ponto x_1 da projeção, em coordenadas homogêneas, é $x_1 = a_{11}X_1 + a_{12}Y_1 + a_{13}Z_1 + a_{14}$. Da mesma forma podemos construir as equações de y_1, z_1 e H. Analisando apenas x_1, se for possível conhecer 4 pontos (X_i,Y_i,Z_i) do mundo real e suas 4 respectivas projeções na tela (x_i,y_i), seria possível resolver um sistema de equações lineares para obter a_{1j}?

Visão stereo e reconstrução 3D

Visão stereo é uma técnica que permite a construção de uma cena tridimensional a partir de duas ou mais imagens bidimensionais ou por meio de projeções de padrões (SZELISKI, 2010). A ideia é inspirada na visão humana, que dispõe de dois olhos para construir a noção de profundidade. O sistema é ilustrado na Figura 2.16.

Figura 2.16 | Visão stereo com duas câmeras

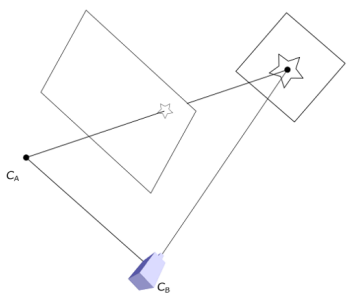


Fonte: elaborada pelo autor.

Na Figura 2.16, considere Tela A, a tela de centro de projeção C_A, e Tela B, a tela de centro de projeção C_B. O ponto P_{w1}, do sistema de coordenadas do mundo, é projetado no ponto P_1 da Tela A, ao mesmo tempo que é projetado no ponto P_3 da Tela B. Todos os pontos da reta que liga P_{w1} a C_A serão projetados no mesmo ponto P_1 da Tela A, como é o caso do ponto P_{w2}. Portanto, olhando apenas o que é projetado na Tela A, não é possível conhecer a profundidade do ponto do mundo real, ou seja, sua distância da Tela A. No entanto, se o mesmo ponto que é projetado na Tela A for reconhecido na Tela B, é possível calcular a profundidade por equações de triângulos. A posição real do ponto P_{w1} é o ponto de interseção da reta que passa por C_A e P_1, com a reta que passa por C_B e P_3. A posição real do ponto P_{w2} é o ponto de interseção da reta que passa por C_A e P_1, com a reta que passa por C_B e P_2.

A visão stereo também pode ser criada por um elemento passivo (câmera) atuando de forma sincronizada com um elemento ativo (projetor). A Figura 2.17 apresenta esta abordagem.

Figura 2.17 | Visão stereo com uma câmera e um projetor



Fonte: elaborada pelo autor.

A Figura 2.17 ilustra um projetor com centro C_B, ângulo de projeção paralelo ao ângulo de captura da câmera de centro C_A, e distância ao centro da câmera C_A conhecida. Nesta configuração, o projetor projeta um padrão (estrela) sobre o objeto do mundo real (quadrado), ou seja, feixes de luz coerente (laser) de forma definida (estrela). A esse padrão se dá o nome de **luz estruturada.** A reflexão do padrão sobre o objeto é capturada pela câmera. Quanto mais distante o objeto do mundo real (quadrado) estiver da câmera, maior será a projeção do padrão sobre o objeto e, consequentemente, maior será sua reflexão sobre o aparato da câmera. Medindo-se o tamanho do padrão capturado pela câmera, é possível calcular a distância (profundidade) do objeto do mundo real com relação à câmera.

Se forem projetados diversos padrões a ponto de preencher toda a tela de captura da câmera, é possível fazer um mapa de profundidade de todos os objetos capturados pela câmera. Este é o princípio utilizado pelo sensor Kinect®, da Microsoft, que marcou uma geração dos jogos digitais. Os dispositivos que usam essa técnica emitem luz laser infravermelho e capturam as imagens com câmeras infravermelho, para que os padrões não sejam visíveis pelo olho humano.

Pesquise mais

O sensor Kinect® é utilizado em videogames para permitir a interação do usuário com o jogo por meio de gestos corporais. O sensor captura imagens por uma câmera comum e também cria um mapa de profundidade que contém as informações de distâncias entre o sensor e os objetos ou jogadores. A informação de profundidade é obtida por visão stereo, com uma câmera e um projetor laser.

CURIOUSINVENTOR. How the Kinect depth sensor **Works in 2 minutes**, 2013.

O Kinect® foi descontinuado pela Microsoft por questões mercadológicas, mas a mesma tecnologia está presente hoje na câmera TrueDepth do iPhone X. É importante observar que a visão stereo com um componente ativo (projetor de luz estruturada) requer que os padrões projetados sejam reconhecidos na imagem bidimensional capturada pela câmera, para que seja possível o cálculo da profundidade (SZELISKI, 2010).

O uso de luz estruturada facilita o trabalho de mapeamento de profundidades, visto que o padrão a ser reconhecido na imagem capturada pela câmera é um padrão conhecido por ter sido projetado em laser sobre os objetos do mundo real. O reconhecimento de padrões em duas imagens digitais capturadas do mundo na forma da Figura 2.16 é mais desafiador,

pois é preciso detectar, em duas imagens, padrões não conhecidos previamente.

Reconstrução 3D

O uso de visão stereo, com duas câmeras ou com um projetor não permite mais do que a geração de um mapa de profundidade dos objetos do mundo real com relação a uma câmera. Não é possível apenas com dois dispositivos reconstruir a forma tridimensional dos objetos reais.

Reconstrução 3D é o processo de mapear, a partir de duas ou mais imagens bidimensionais (ou a partir de imagens e projetores de luz estruturada), as coordenadas do mundo real dos objetos que aparecem nessas imagens. A reconstrução 3D é o processo de criar modelos tridimensionais a partir de imagens capturadas por sensores quaisquer. Para a criação de modelos tridimensionais realistas, várias câmeras ou projetores podem ser usados.

Exemplificando

A reconstrução 3D é usada em modelagem tridimensional para a criação de modelos 3D a partir de imagens capturadas por câmeras comuns, como visto na Seção 1.1 (LIVERPOOL FC, 2013; VIRTUMAKE, 2013). Com várias câmeras é possível criar modelos 3D com *crowd-sourcing*. O vídeo sugerido a seguir exemplifica a reconstrução 3D, uma modelagem tridimensional
de Roma, feita a partir de fotografias disponibilizadas por turistas na rede social Flickr.
ITWC. **Flickr photos used to build Rome in digital 3D**, 2014.

O desafio da reconstrução 3D é o de relacionar cada ponto, de cada imagem capturada, com um ponto de cada outra imagem capturada. O reconhecimento de padrões em duas imagens digitais é um problema de visão computacional, e esse problema é aplicado à modelagem tridimensional, evidenciando o quanto as subáreas da computação gráfica interagem entre si.

O conteúdo desta seção mostrou o quanto o modelo de câmera é essencial para a computação gráfica em três de suas subáreas: síntese de imagens, visão computacional e modelagem tridimensional. Foi possível mostrar ainda que o reconhecimento de padrões, necessário para a visão computacional e modelagem tridimensional, pode exigir técnicas de processamento digital de imagens. O modelo de câmera é, portanto, conteúdo básico para o profissional que irá atuar em qualquer subárea da computação gráfica. A

próxima unidade irá evidenciar ainda mais o modelo de câmera como parte da síntese de imagens. Continue dedicando-se aos estudos!

Sem medo de errar

Caro aluno, no início desta seção lhe foi apresentada uma situação-problema da vida do profissional de computação gráfica. Você faz parte da equipe de desenvolvimento de um novo aplicativo de criação de animações 3D. No aplicativo, o usuário será capaz de visualizar e transformar modelos geométricos e de definir o ponto de vista do espectador da animação, posicionando câmeras virtuais. Sua equipe desenvolve a biblioteca de transformações geométricas.

Seu problema é a implementação dos modelos de câmera, que utilizarão as transformações geométricas básicas e poderão ser aplicados sobre os modelos geométricos. Você deve entregar o código em linguagem de programação, contendo funções que executem as transformações geométricas necessárias para que um modelo de câmera seja aplicado tanto para problemas de visão computacional quanto para problemas de síntese de imagens.

Primeiramente é preciso observar que o modelo de câmera é exatamente o mesmo, seja ele aplicado a problemas de visão computacional, seja ele aplicado a problemas de síntese de imagens. O primeiro trecho de código a ser implementado é a função *matrizMc*, que constrói a matriz de transformação do modelo de câmera com base nos parâmetros mostrados na Figura 2.15: B, α, θ, r e a distância D, do ponto C à origem do sistema de coordenadas da câmera. Uma sugestão de implementação da função *matrizMc* em Python é dada a seguir.

```
from numpy import matmul
def matrizMc(B,alpha,theta,r,d):
    (Xb,Yb,Zb)=B
    (Xr,Yr,Zr)=r
    TB = matrizT3d(-Xb,-Yb,-Zb)
    Rx = matrizRx3d(alpha)
    Rz = matrizRz3d(theta)
    Tr = matrizT3d(-(Xr-Xb),-(Yr-Yb),-(Zr-Zb))
    P = matrizP(d)
    Mc = matmul(P,matmul(Tr,matmul(Rz,matmul(Rx,TB))))
    return Mc
```

Agora é preciso acrescentar as implementações das funções *matrizT3d*, *matrizRx3d*, *matrizRz3d* e *matrizP*, que são as implementações das matrizes das transformações básicas. Para fins de exemplo, o código a seguir mostra a implementação de duas dessas funções.

```
import numpy

def matrizT3d(tx,ty,tz):
    return numpy.array([[1,0,0,tx],
                        [0,1,0,ty],
                        [0,0,1,tz],
                        [0,0,0,1]])

def matrizRx3d(alpha):
    return numpy.array([[1,0,0,0],
                        [0, math.cos(alpha),-math.sin(alpha),0],
                        [0,math.sin(alpha),math.cos(alpha),0],
                        [0,0,0,1]])
```

Coloque todo o código-fonte acima em um único arquivo e carregue esse arquivo no console do Python. Depois basta chamar as funções nele implementadas para que o console do Python imprima as matrizes retornadas pelas funções. Nas atividades práticas desta seção você irá aplicar essas matrizes sobre conjuntos de pontos.

Basta acrescentar agora as implementações das demais funções para se ter a solução completa da situação-problema. Com esta solução você irá demonstrar a capacidade de aplicar transformações geométricas e implementar um conjunto delas em modelos de câmera.

Avançando na prática

Sistema de aferição de comprimentos de tubos por visão computacional

Descrição da situação-problema

Seu cliente é uma indústria que produz tubos metálicos de comprimentos que variam de um a dois metros de comprimento. Ao final da cadeia produtiva, os tubos devem ter suas medidas conferidas. Hoje a aferição do comprimento do tubo é feita por um ser humano, e seu cliente deseja automatizar esse processo. Você foi contratado para prover uma solução com o uso de uma câmera de alta resolução. Você pode incluir elementos na cena que facilitem a sua medição. Você deve apresentar o projeto de instalação física do seu sistema na indústria.

Resolução da situação-problema

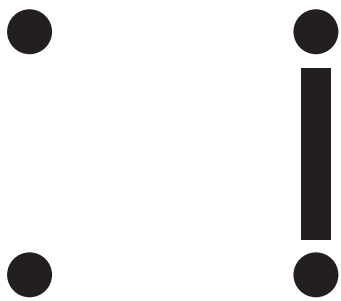
Trata-se de um problema típico de visão computacional. A primeira coisa a ser instalada no ambiente da indústria é a própria câmera. A situação ideal é a colocação da câmera como uma vista de cima do tudo a ser medido. O

motivo é que a visão por cima reduz a distorção do objeto quando sofre a transformação de perspectiva para ser projetado na câmera.

O segundo ponto é que a câmera deverá estar ligada a um computador, mas este não precisa estar próximo, e, portanto, não é uma dificuldade adicional do projeto.

O terceiro ponto é o mais complexo. Um sistema de medição que utiliza câmeras deve fazer medições em uma imagem digital, mas converter essas medidas para medidas do mundo real. Isto significa que é preciso fazer a calibração da câmera. Para permitir esta calibração, sugere-se a colocação do padrão apresentado na Figura 2.18 na mesa ou esteira onde será colocado o tudo a ser medido, em uma posição sempre visível pela câmera.

Figura 2.18 | Padrão para calibração de câmera



Fonte: elaborada pelo autor.

Os discos são facilmente reconhecidos na imagem da câmera e servem como os 4 pontos de coordenadas conhecidas no mundo real. A barra retangular deve ser de medida real conhecida e serve para criar a relação entre a medida do mundo real com a medida na imagem digital.

Desta forma é possível calibrar a câmera.

Faça valer a pena

1. O modelo de câmera é a composição de transformações geométricas básicas de translação, rotação e perspectiva, aplicadas em uma ordem específica. Não é possível aplicar direta e simplesmente a transformação de perspectiva, a não ser em um caso muito específico.

Por que não é possível aplicar diretamente a transformação perspectiva, e qual seria esse caso específico em que é possível?

a) A transformação perspectiva só pode ser aplicada para projeções em câmeras estáticas e somente neste caso específico. Não pode ser aplicada para câmeras em movimento.

b) A transformação perspectiva só pode ser aplicada para projeções de pontos estáticos e somente neste caso específico. Não pode ser aplicada para pontos em movimento.

c) O modelo de câmera é um modelo genérico que pode englobar uma ou mais câmeras. A transformação perspectiva só pode ser aplicada diretamente se houver apenas uma câmera.

d) A transformação perspectiva só pode ser aplicada quando o sistema de coordenadas do mundo real é coincidente com o da câmera e só neste caso ela pode ser aplicada diretamente.

e) A transformação perspectiva só pode ser aplicada quando os eixos do sistema de coordenadas do mundo real são paralelos aos da câmera e só neste caso ela pode ser aplicada diretamente.

2. O modelo de câmera é um modelo que exige o conhecimento prévio da posição e o direcionamento da câmera em termos do sistema de coordenadas do mundo real. Isto é prático para a síntese de imagens, que irá sintetizar imagens a partir de modelos geométricos. Em visão computacional, no entanto, é muito difícil conhecer exatamente a da posição e direcionamento da câmera em termos do sistema de coordenadas do mundo real. Por esta razão, utiliza-se um processo de **calibração de câmera**, que calcula, automaticamente, os parâmetros da matriz M_c do modelo de câmera.

$$M_c = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

Assinale a alternativa que explica como esse processo de calibração de câmera pode ser feito.

a) São colocados no mundo real dois padrões facilmente reconhecíveis na imagem capturada. Como são apenas x e y, conhecendo as coordenadas do mundo real de apenas dois pontos e suas respectivas coordenadas na câmera, monta-se um sistema linear para calcular M_c.
b) São colocados no mundo real quatro padrões facilmente reconhecíveis na imagem capturada. Conhecendo as coordenadas do mundo real de quatro pontos e suas respectivas coordenadas na câmera, monta-se um sistema de equações lineares para calcular os parâmetros de M_c.
c) São colocados no mundo real dois padrões facilmente reconhecíveis na imagem capturada, pois são apenas duas dimensões na câmera. A calibração da câmera consiste em movimentá-la com translações e rotações para alinhar os padrões e poder usar uma matriz M_c predefinida.
d) É colocado no mundo real o desenho do sistema de coordenadas do mundo real, com os eixos X, Y e Z. O desenho é facilmente reconhecível na imagem capturada. Com base no desenho, é possível calcular as matrizes de rotação, translação e perspectiva que compõem M_c.
e) São colocados no mundo real quatro padrões facilmente reconhecíveis na imagem capturada. A calibração da câmera consiste em movimentá-la com translações e rotações para alinhar os padrões reconhecidos na imagem capturada, de forma a poder usar uma matriz M_c predefinida.

3. O Kinect® foi um sensor utilizado em videogames para permitir a interação do jogador por meio de gestos corporais, sem o uso de um joystick. O Kinect® foi descontinuado, mas sua tecnologia está hoje presente em smartphones de última geração. Muitos pesquisadores buscaram utilizar o Kinect® em diversas outras aplicações. No entanto, há uma limitação: o Kinect® não funciona quando utilizado em espaços externos, diretamente iluminados pelo sol.

Assinale a alternativa que melhor explique o motivo de o sensor Kinect® não poder ser utilizado à luz solar.
a) O Kinect® utiliza um sistema de visão stereo com projeção de luz estruturada na frequência do infravermelho. O sol emite raios infravermelho em grande quantidade, que se confundem com a luz estruturada projetada, não sendo possível extrair a informação desejada.
b) Não há impedimento na utilização do Kinect® à luz solar. No entanto, o sensor vem de fábrica calibrado para uso em ambiente interno, e basta fazer a sua calibração para uso em ambiente externo, à luz solar.
c) O Kinect® utiliza um sistema de visão stereo com duas câmeras. A posição de uma câmera com relação à outra é garantida com a precisão de micrômetros. A exposição do equipamento ao calor do sol provoca dilatações que introduzem erros na precisão dos cálculos.
d) Não há impedimento na utilização do Kinect® à luz solar, uma vez que o equipamento não utiliza sensores óticos para a construção do mapa de profundidade. Utiliza

um sonar. A câmera do dispositivo serve para o reconhecimento dos gestos.
e) O Kinect® utiliza um sistema de visão stereo com projeção de luz estruturada no espectro da luz visível. O equipamento emite laser em cores azul, verde e vermelho. À luz solar, a câmera que capta as imagens não é capaz de diferenciar claramente essas três cores.

Referências

ANGEL, E.; SHREINER, D. **Interactive Computer Graphics: a top-down approach with shader-based OpenGL**. 6. ed. Pearson, 2012.

AZEVEDO, E.; CONCI, A.; VASCONCELOS, C. **Computação Gráfica Teoria e Prática** – Vol. 1. 2. ed. Rio de Janeiro, RJ: Elsevier, 2018.

BERG, M. et al. **Computational Geometry: Algorithms and Applications**. 3. ed. Springer, 2008.

BLENDERPOWER. **Modelagem 3D para iniciantes**. Disponível em: https://youtu.be/ubCnbw9XUPo, 2014. Acesso em: 22 nov. 2018.

BOLDRINI, J. L. et al. **Álgebra linear**. 3. ed. Harbra, 1986.

BOYLE, W.; SMITH, G. **Charge coupled semiconductor device**. The Bell System Technical Journal, v. 49, 1970.

DAVIES, E. **Computer & machine vision:** theory, algorithms, practicalities. 4. ed. Elsevier, 2012.

FUSSEL, M. **The Virtual Instructor – One Point Perspective**. 2015. Disponível em: https://youtu.be/bjhkxFDvD78. Acesso em: 7 jan. 2019.

GONZALEZ, R. C.; WOODS, R. E. **Processamento de imagens digitais**. 3. ed. Pearson, 2011.

HEARN, D. D.; BAKER, M. P.; CARITHERS, W. **Computer Graphics with OpenGL**. 4. ed. Pearson, 2010.

HUGHES, J. F. et al. **Computer graphics:** principles and practice. 3. ed. Addison-Wesley, 2013.

INKSCAPE. Inkscape Draw Freely 0.92, 2018. Disponível em: http://inkscape.org. Acesso em: 8 jan. 2019.

JAIN, R.; KASTURI, R.; SCHUNK, B. **Machine vision**. McGraw-Hill, 1995.

KREYSZIG, E. **Advanced engineering mathematics**. 10th ed. Wiley, 2011.

LIVERPOOL FC. **FIFA 14 | Liverpool 3D Team Head Scan**, 2013. Disponível em: https://www.youtube.com/watch?v=ZBIjwU6EbOA. Acesso em: 14 nov. 2018.

OLIVEIRA, S. L. G. **Introdução à Geração de Malhas Triangulares**. São Carlos, SP : SBMAC, 2015 (Notas em Matemática Aplicada; v. 79).

SZELISKI, R. **Computer vision:** algorithms and applications. Springer, 2010.

VIRTUMAKE. **3D Scan - Professional vs Hobby - Artec Eva, Carmine 1.09**, 2013. Disponível em: https://www.youtube.com/watch?v=H3WzY8EWM9s. Acesso em: 14 nov. 2018.

Unidade 3

Computação gráfica tridimensional

Convite ao estudo

Caro aluno, as subáreas da computação gráfica compartilham o embasamento teórico da geometria, mas algumas técnicas são específicas de uma ou mais subáreas. Nesta unidade vamos estudar conceitos e técnicas aplicáveis à subárea de síntese de imagens.

A síntese de imagens lida com a geração de imagens sintéticas, que podem ser utilizadas de forma isolada ou em combinação com imagens reais. Cinema, TV e outros vídeos muitas vezes combinam imagens reais e imagens sintéticas. A síntese de imagens é, portanto, uma subárea mais aplicada, a qual utiliza conceitos e ferramentas que vão além da geometria, mesmo que dela dependam. São conceitos sobre iluminação, sombreamento, tonalização e textura.

Considere que você continua na empresa que está desenvolvendo um aplicativo de criação de animações 3D. Pelo desempenho na equipe de desenvolvimento da biblioteca de ferramentas fundamentais de transformações geométricas, você foi convidado a migrar para a equipe de desenvolvimento de ferramentas de síntese de imagens.

A equipe de trabalho que você passa a fazer parte está encarregada do desenvolvimento das ferramentas de geração de imagens e animações gráficas a partir dos modelos trabalhados pelo usuário da aplicação. Seu trabalho não é com a interface gráfica do usuário do aplicativo de modelagem, mas com o processamento dos modelos criados pelo usuário do aplicativo, para gerar imagens sintéticas a partir de modelos geométricos.

O usuário do aplicativo interage com a interface gráfica para criar curvas, superfícies e sólidos tridimensionais, posicionar e orientar câmeras e fontes de iluminação, além de criar modelos de movimento dos objetos, câmeras e fontes de iluminação ao longo do tempo. O resultado do trabalho de criação do usuário do aplicativo é um modelo tridimensional completo. Finalmente, o usuário poderá solicitar a visualização do modelo a partir de um ponto de vista estático, gerando uma imagem sintética, ou ao longo de uma sequência temporal de pontos de vista de uma câmera em movimento, gerando uma animação 3D.

É no momento da geração de imagens ou animações 3D que as ferramentas desenvolvidas pela sua equipe serão utilizadas. Seu trabalho, por agora, é desenvolver as funções mais básicas de projeção de objetos 3D considerando a possibilidade de oclusões.

Primeiramente você deverá desenvolver o pipeline tradicional de visualização e implementar algoritmos de projeção de faces poligonais planas e retas com oclusão. Em seguida, você deverá incluir iluminação e texturas. Finalmente, você deverá incluir a dimensão temporal, modelando o movimento de uma câmera virtual num modelo 3D.

Esse é apenas o início do trabalho da equipe de síntese de imagens, mas nem por isso pouco desafiador. Esteja sempre bem atento para dominar esses conceitos básicos para que possa ser requisitado para as atividades mais avançadas!

Seção 3.1

CGPI: síntese de imagens

Diálogo aberto

Caro aluno, esta seção aborda os primeiros conceitos da síntese de imagens: o pipeline tradicional de visualização e o algoritmo de *ray casting*, para a projeção de uma cena 3D sobre o plano, considerando os problemas de recortes e oclusão de objetos. Entende-se por pipeline de visualização a sequência de operações e transformações que devem ser executadas para gerar uma imagem 2D a partir de um modelo 3D, considerando como ponto de vista uma das câmeras virtuais presentes no modelo.

Estamos considerando um contexto em que você foi alocado na equipe de desenvolvimento de ferramentas de síntese de imagens, em uma empresa que está desenvolvendo um novo aplicativo de criação de animações 3D. O usuário do aplicativo cria um modelo 3D e solicita a sua visualização a partir de um ponto de vista estático, gerando uma imagem sintética, ou ao longo de uma sequência temporal de pontos de vista de uma câmera em movimento, gerando uma animação 3D. Seu trabalho é desenvolver as ferramentas que permitam essas visualizações.

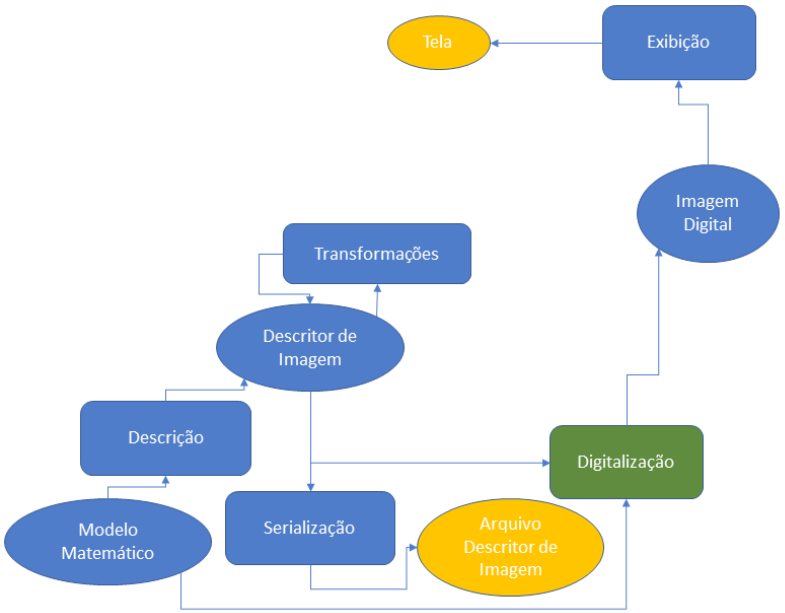
A primeira etapa do seu trabalho é o desenvolvimento do pipeline tradicional de visualização. O pipeline de visualização é uma sequência de operações que cria uma imagem digital bidimensional a partir de transformações de um modelo geométrico tridimensional. O modelo 3D é composto por um conjunto de objetos 3D, entre eles sólidos, curvas, malhas, fontes de iluminação e câmeras virtuais. Todos esses objetos 3D estão posicionados e orientados no sistema de coordenadas do mundo real e devem ser transformados para o sistema de coordenadas de uma das câmeras do modelo e depois criar a projeção do modelo sobre a tela. Esse processamento utiliza as transformações geométricas básicas, mas deve levar em consideração, também, informações de textura e iluminação. Você deverá entregar o código fonte, em linguagem de programação, do pipeline básico abstrato, ou seja, com o uso apenas de interfaces, classes abstratas ou protótipos de funções. O algoritmo de *ray casting* deve ser de fato implementado, mas considerando já existente uma função de cálculo do ponto de interseção de uma reta e um polígono.

Nesta seção você irá conhecer o pipeline de visualização, o algoritmo de *ray casting* e os problemas de recorte e oclusão a ele relacionados. O conteúdo desta seção trata de problemas cujos resultados são aqueles apresentados ao público leigo. Você se aproxima mais do usuário final da computação gráfica, o que torna o aprendizado ainda mais motivador.

Não pode faltar

Caro aluno, nesta seção veremos com mais profundidade as técnicas relacionadas à síntese de imagens. A Figura 3.1 é uma recapitulação da Figura 1.3, com destaque a um processamento específico: a digitalização. A Figura 3.1 é um diagrama extraído de um diagrama maior, que mostra de forma resumida alguns processamentos e artefatos relacionados à síntese de imagens.

Figura 3.1 | Artefatos e processamentos relacionados à síntese de imagens



Fonte: elaborada pelo autor.

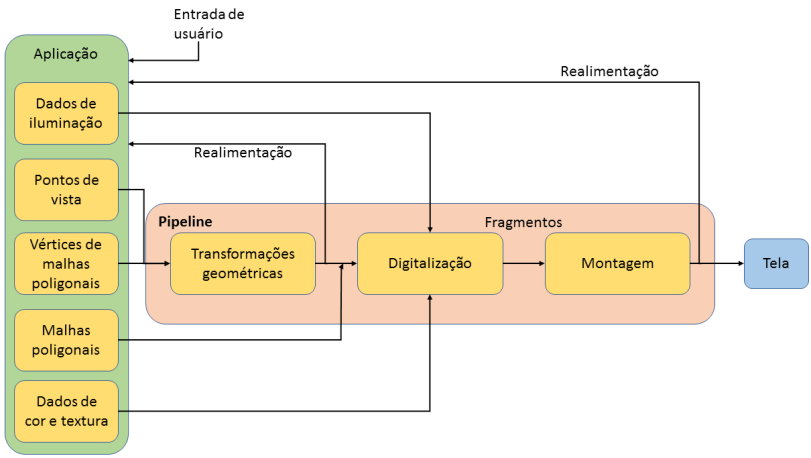
No diagrama da Figura 3.1 podemos notar que o modelo matemático é primeiramente descrito por estruturas de dados adequadas para que possa passar por transformações. Essas transformações são as transformações geométricas estudadas na Unidade 2. Os descritores já transformados unem-se novamente a outras informações do modelo matemático para que possa ser executado o processo de digitalização da imagem. A digitalização é o processo que cria uma imagem digital que será exibida em uma tela.

Agora que já temos mais conhecimento das técnicas de modelagem tridimensional, vamos refazer o diagrama da Figura 3.1 com base em um formato amplamente utilizado na literatura: o **pipeline tradicional de visualização**.

Pipeline tradicional de visualização

A palavra pipeline é utilizada para indicar que se trata de um processo de vários estágios em que a saída de um estágio é a entrada de outro estágio na sequência. A Figura 3.2 mostra o pipeline de visualização segundo Hughes *et al.* (2013). Muitas vezes o pipeline de visualização é também chamado de pipeline de renderização, um termo aportuguesado a partir do inglês "*rendering pipeline*".

Figura 3.2 | Pipeline tradicional de visualização segundo Hughes ***et al.*** (2013)



Fonte: adaptada de Hughes ***et al.*** (2013).

No pipeline da Figura 3.2, o modelo matemático é composto por cinco elementos: dados de textura, malhas poligonais, vértices das malhas poligonais, pontos de vista e dados de iluminação. As malhas poligonais são as descrições de como os vértices das malhas poligonais estão conectados para formar as faces poligonais.

As transformações geométricas das malhas poligonais são aplicadas aos vértices das malhas poligonais. Para representar translações e rotações de um único objeto, são aplicadas transformações geométricas sobre os vértices da malha desse objeto. Para representar o reposicionamento e redirecionamento de uma câmera virtual (ponto de vista), são aplicadas transformações geométricas sobre todos os vértices de todas as malhas que compõem a cena. São aplicadas todas as transformações de um modelo de câmera, exceto a transformação de perspectiva, que será feita no estágio de digitalização. Por essa razão, os vértices e o ponto de vista são os elementos de entrada do estágio de transformações geométricas.

Reflita

A transformação de perspectiva foi estudada para a projeção de um ponto da cena tridimensional. Mas a cena 3D é um conjunto de malhas poligonais definidas no espaço contínuo. Os polígonos podem ser descritos por seus vértices e arestas, mas são compostos por infinitos pontos em suas superfícies. Como seria possível aplicar a transformação perspectiva para infinitos pontos? É necessária uma estratégia mais eficiente.

Aos vértices já transformados são adicionadas informações sobre as faces das malhas poligonais, informações sobre textura e iluminação, para compor a entrada do estágio de digitalização. O processo de digitalização não se resume à projeção perspectiva de pontos, como visto para o modelo de câmera, pois deve levar em consideração a existência de diversos objetos em que um objeto pode estar à frente de outro (oclusões). O processo de digitalização, portanto, é realizado por partes, gerando diversos fragmentos de imagem digital que são a entrada do último estágio do pipeline: a montagem da imagem final.

O pipeline apresenta ainda uma entrada de usuário, que permite a interação com o modelo 3D, seja modificando objetos da cena, seja alterando a posição e direcionamento de câmeras virtuais (pontos de vista). Junto com a realimentação de informações de estágios do pipeline para o modelo, a entrada de usuário permite a criação de animações interativas.

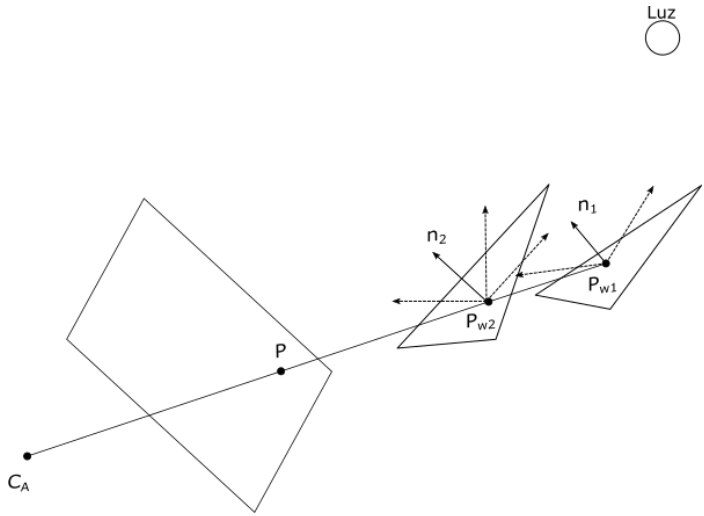
Existem diferentes *application programming interfaces* (APIs) de desenvolvimento que implementam tarefas e até mesmo todo o pipeline gráfico, podendo utilizar recursos disponíveis em hardware nas unidades de processamento gráfico (GPUs). Entre essas APIs, destacam-se a Open Graphics Library – OpenGL (HEARN *et al.*, 2010), multiplataforma, e Direct3D (STENNING, 2014), para plataforma Windows®.

Nesse pipeline, o estágio de transformações geométricas processa dados definidos no espaço contínuo para gerar dados ainda no espaço contínuo. O trabalho de conversão do espaço contínuo para o espaço discreto da imagem digital é uma tarefa do estágio de digitalização, que envolverá, também, cálculos geométricos. A seguir veremos tarefas do estágio de digitalização.

Ray casting e digitalização

Para se compreender o que deve ser feito para a digitalização de uma cena 3D, o primeiro passo após o estudo das transformações geométricas é o estudo dos efeitos de luz, cores e textura. A Figura 3.3 ilustra dois triângulos opacos e uma fonte de luz.

Figura 3.3 | Luz capturada por um ponto da câmera (observador)



Fonte: elaborada pelo autor.

O triângulo 1 da Figura 3.3 tem normal n_1, e o triângulo 2 tem normal n_2. A luz emitida pela fonte pode ser refletida de forma difusa (considerando uma superfície pouco refletiva) sobre cada ponto da superfície de cada triângulo. A reflexão nos pontos P_{w1} e P_{w2} foi representada pelas setas tracejadas. Uma das direções da reflexão da luz é a direção da reta que leva ao centro de projeção C_A, passando pelo ponto P. Como os triângulos são opacos, o ponto P receberá a luz refletida sobre o ponto P_{w2}, somente, visto que esse ponto oculta a reflexão no ponto P_{w1}.

Pesquise mais

A Khan Academy apresenta vídeos que ilustram o processo de visualização.

- PIXAR IN A BOX. **Introdução à renderização**.
- PIXAR IN A BOX. **O que é ray tracing (traçado de raios)**.
- PIXAR IN A BOX. **Renderização 101**.

Com base nesse princípio foi definido o algoritmo de *ray casting*. A estratégia do *ray casting* é projetar uma linha reta R a partir do centro de projeção C_A, passando pelo ponto P, em direção à cena, no sentido inverso do raio (*ray*) luminoso proveniente da fonte de luz. O primeiro polígono intersectado por essa linha reta é o polígono que será observado no ponto P. No caso da Figura 3.3, trata-se do triângulo 2. Para chegar a essa conclusão, o *ray casting* calcula o

ponto de interseção da reta R com cada polígono da cena e verifica qual desses pontos de interseção é o mais próximo do centro de projeção C_A.

Exemplificando

A reflexão da luz pode ser difusa, quando a superfície do objeto é fosca, ou especular, quando a superfície do objeto é brilhante.
PIXAR IN A BOX. **Reflexão da luz**.

Observe que a face de um polígono no espaço contínuo é composta por infinitos pontos, e a reflexão da luz sobre cada ponto é difusa, criando raios em infinitas direções. Seria inviável computar a transformação perspectiva de cada um dos infinitos pontos da cena. A tela de projeção, por sua vez, é matricial, com um número finito de pontos (imagem digital). A estratégia do *ray casting* limita a quantidade de processamento ao número de pixels da imagem, tornando viável a computação das projeções.

Para digitalizar toda a cena, o algoritmo *ray casting* deve ser executado para cada pixel da imagem. O código em Python a seguir apresenta esse algoritmo.

```
def raycastingP(f,model,d):
    (xs,ys) = f.shape
    ca = (0,0,-d)
    d = 0
    for y in range(ys):
        for x in range(xs):
            p = (x,y,0)
            r = ray(ca,p)
            for poly in model.getPolygons():
                pi = intersecao(r,poly)
                dp = distancia(ca,pi)
                if (dp < d):
                    d = dp
                    f(x,y) = getColor(pi,poly)
```

O código acima assume que todos os vértices do modelo já passaram por transformações geométricas e no momento da execução do algoritmo de *ray casting* encontram-se no sistema de coordenadas da tela de projeção. Logo, as coordenadas do ponto C_A, de distância d à origem, são (0,0,-d), e as coordenadas de qualquer ponto (x,y) da tela de projeção são (x,y,0). O código acima considera já implementadas as funções *distancia*, que calcula a distância entre dois pontos 3D; *getColor*, que obtém os dados de cor e textura do ponto no polígono intersectado; *intersecao*, que calcula o ponto de interseção do da reta com o polígono e o método *model.getPolygons()*, que retorna

uma lista de todos os polígonos da cena. Esse código é paralelizável para cada ponto (x,y) e o custo computacional para cada ponto (x,y) é proporcional ao número de polígonos. O cálculo do ponto de interseção do raio com um triângulo (polígono mais simples) pode ser visto na seção 15.4.3 da obra de Hughes *et al.* (2013) e na obra de Möller e Trumbore (1997).

Considere agora que o laço mais externo do algoritmo implementado em Python acima seja a iteração sobre os polígonos. Assim obtemos o código a seguir.

```
def raycastingT(f,model,d):
    (xs,ys) = f.shape
    ca = (0,0,-d)
    d = 0
    for poly in model.getPolygons():
        for y in range(ys):
            for x in range(xs):
                p = (x,y,0)
                r = ray(ca,p)
                pi = intersecao(r,poly)
                dp = distancia(ca,pi)
                if (dp < d):
                    d = dp
                    f(x,y) = getColor(p,poly)
```

A diferença da primeira versão do algoritmo para a segunda pode parecer sutil e irrelevante, mas está fortemente relacionada à capacidade de processamento de cenas com mais ou menos detalhes. Na primeira versão, o processamento de cada reta exige que esteja em memória todo o modelo tridimensional, enquanto na segunda versão, o processamento de cada polígono exige que esteja em memória toda a matriz bidimensional da imagem digital. Se o modelo tridimensional é muito detalhado, ou seja, com um grande número de polígonos, o espaço de memória exigido para o modelo pode ser muito maior do que o espaço de memória exigido para a imagem 2D. Em outras palavras, o processamento pixel a pixel exige muito mais memória do que o processamento triângulo a triângulo. O paralelismo, na segunda versão, está no número de polígonos, enquanto o da primeira versão está no número de pixels da imagem digital.

Exemplificando

Vamos considerar um jogo digital executado em um console para uma TV 4K. A resolução digital 4K é tipicamente 3840×2160 pixels (aproximadamente 8,3 Mpixels). Se cada pixel ocupa 3 bytes (RGB), o total de memória necessário para armazenar toda a imagem digital a ser exibida é de

aproximadamente 25MB. Por outro lado, é muito comum encontrarmos modelos 3D com mais de 60 milhões de vértices e mais de 60 milhões de polígonos (CGTRADER, 2018). Cada vértice são três coordenadas reais, representadas em ponto flutuante de precisão dupla (*double*), que ocupa 4B, cada. Cada vértice, portanto, ocupa 12B. Com 60 milhões de vértices, o modelo ocupará cerca de 720MB. Tente estimar, ainda, quanta memória seria necessária para armazenar um modelo com 19 bilhões de triângulos como o modelo criado por Agus3D (2012). Uma visualização desse modelo é apresentada na Figura 3.4. Fica claro que os modelos 3D ocupam uma quantidade substancialmente maior de memória que a imagem digital a ser exibida, o que faz a segunda versão do algoritmo *ray casting* apresentada mais viável do que a primeira versão.

Figura 3.4 | Renderização do modelo de Agus3D (2012), com 19 bilhões de triângulos. A cena possui 54 mil instâncias de duas árvores, uma com cerca de 200 mil triângulos e outra com cerca de 150 mil triângulos

Fonte: Bart (2012).

Até o momento foram considerados apenas os raios diretamente refletidos sobre a superfície de cada polígono da malha poligonal, sem considerar a possibilidade de existirem outros objetos na cena que contribuam com a luminosidade que incide sobre o pixel por outros caminhos.

O *ray casting* considera que todos os polígonos da cena refletem diretamente a luz proveniente da fonte. Tal restrição impede a digitalização mais realística, pois desconsidera três outras formas de contribuição da fonte de luz para com o pixel projetado: reflexão indireta, sombra e refração.

Na cena pode existir um objeto que se encontra entre a fonte de luz e o polígono mais próximo do centro de projeção, o que geraria uma sombra, e não uma reflexão direta. Há ainda a reflexão indireta, quando o ponto

projetado recebe, além da iluminação direta (ou sombra) da fonte de luz, reflexões de outros objetos da cena. Finalmente, ainda devemos considerar a possibilidade da existência de objetos transparentes, pelos quais a luz proveniente da fonte é refratada.

Pesquise mais

Sombra, reflexão indireta e refração da luz, que diferenciam o *ray tracing* do *ray casting*.

PIXAR IN A BOX. **Raios de luz**.

A generalização do *ray casting* que considera luz indireta, sombras e refração, é *denominada* ***ray tracing*** (ANGEL; SHREINER, 2012). O *ray tracing* é o algoritmo de referência na atualidade. Para implementá-lo é preciso aprofundar as questões geométricas, no caso da reflexão indireta, e físicas, no caso da refração. O aprofundamento pode ser feito por meio da obra de Hearn *et al*., no capítulo 15, da página 493 à 542, e na seção 19.1, da página 633 à 647).

Assimile

O algoritmo de *ray tracing* é uma extensão do algoritmo de *ray casting*. O *ray casting* considera todos os objetos da cena como opacos e trata apenas a reflexão direta da luz sobre o objeto visualizado. O *ray tracing* considera também objetos transparentes e trata a reflexão da luz sobre diversos objetos até chegar no observador. Trata, ainda, de sombras e refração.

Sem medo de errar

Caro aluno, na situação-problema apresentada, você deverá entregar o código fonte, em linguagem de programação, do pipeline tradicional de visualização abstrato, ou seja, com o uso apenas de interfaces, classes abstratas ou protótipos de funções. O algoritmo de *ray casting* deve ser de fato implementado, mas considerando já existente uma função de cálculo do ponto de interseção de uma reta e um polígono.

Para resolver essa situação-problema, primeiramente devemos retomar os conhecimentos da Unidade 2 e implementar as classes abstratas que representam as malhas poligonais. O código em Python a seguir mostra essa representação.

```
import numpy
class Vertice:
    def __init__(self, x, y, z):
```

```
        self.x = x
        self.y = y
        self.z = z

class Poligono:
    def __init__(self):
        self.sequenciaVertices = []

    def setSequenciaVertices(self, sequencia):
        self.sequenciaVertices = sequencia

    def getVertices(self):
        return self.sequenciaVertices

class Malha:
    def __init__(self):
        self.vertices = set()
        self.poligonos = set()

    def getVertices(self):
        return self.vertices

    def getPoligonos(self):
        return self.poligonos

    def addPoligono(self,poligono):
        self.poligono.add(poligono)
        self.vertices.add(poligono.getVertices())
```

Em seguida, você deve implementar classes abstratas que representem as câmeras virtuais, os dados de iluminação, cor e textura e a cena 3D.

```
class CameraVirtual:
    #câmera virtual com centro de coordenadas (0,0,0)
    e centro de projeção (0,0,-d)
    def __init__(self, d):
        self.d = -d #distancia do centro de projeção
        self.posicaoDirecao = None

    def setPosicaoDirecao(self, matriz):
        # matriz é uma matriz 4x4 de transformação afim
        self.posicaoDirecao = matriz

class CoresTexturas:
    # mapeamento de um polígono para sua respectiva
    CorTextura
    def __init__(self):
        self.map = {}
```

```
    def addCorTextura(self,poligono,cortextura):
        self.map[poligono] = cortextura

class FonteLuz:
    # fonte de luz omnidirecional
    def __init__(self, x, y, z):
        self.x = x
        self.y = y
        self.z = z

class Cena:
    def __init__
(self, iluminacao, camera, malhas, corestexturas):
        self.iluminacao = iluminacao
        self.camera = camera
        self.malhas = malhas
        self.corestexturas = corestexturas
```

Cada estágio do pipeline deve ser implementado.

```
def transformacoes(camera, vertices):
    novosVertices = set()
    #novosVertices = aplicação da matriz a todos os
    vértices de vertices
    return novosVertices

def digitalizacao(vertices,poligonos,iluminacao,corestex-
turas):
    fragmentos = []
    # fragmentos = execução da digitalização
    return fragmentos

def montagem(fragmentos):
    imagem = numpy.zeros(1)
    # imagem = execução da montagem
    return imagem

def pipeline(cena,matriz):
    #matriz é a mat
    vertices = set()
    poligonos = set()
    for malha in cena.malhas:
        vertices.union(malha.vertices)
        poligonos.union(malha.poligonos)
    vertices = transformacoes(cena.camera, vertices)
    fragmentos = digitalizacao(vertices,poligonos,cena.
iluminacao,cena.corestexturas)
```

```
    imagem = montagem(fragmentos)
    return imagem
```

A sugestão de implementação acima não possui classes de fato abstratas, mas incompletas. O algoritmo de *ray casting* é exatamente o mesmo apresentado nesta seção.

Avançando na prática

Incrementando o *ray casting*

Descrição da situação-problema

Vimos que o *ray tracing* é uma generalização do *ray casting,* que trata sombras, reflexão indireta e refração da luz. O algoritmo de *ray tracing* é mais complexo, mas você seria capaz de construí-lo de forma incremental, adicionando ao *ray casting*, um a um, os tratamentos de sombra, reflexão indireta e refração.

Você foi designado a implementar o primeiro incremento ao algoritmo *ray casting*: a detecção de sombras. Você deve alterar o algoritmo de *ray casting,* incluindo o tratamento de sombras, sem se preocupar com a eficiência do algoritmo.

Resolução da situação-problema

Como não é exigido um algoritmo eficiente, a solução imediata para o problema é acrescentar um trecho de código após a detecção de que o polígono encontrado na iteração corrente é mais próximo do que os polígonos encontrados em iterações anteriores. Assim, dentro do *if(dp < d),* você pode incluir uma espécie de *ray casting* do ponto encontrado (pi) até a fonte de luz. Se algum triângulo for encontrado a uma distância menor do que a distância de pi até a fonte de luz, conclui-se que há uma oclusão da luz: sombra. O código em Python a seguir apresenta uma possível solução.

```
def raycastingTsombra(f,model,d,luz):
    (xs,ys) = f.shape
    ca = (0,0,-d)
    d = 0
    for poly in model.getPolygons():
        for y in range(ys):
            for x in range(xs):
                p = (x,y,0)
```

```
r = ray(ca,p)
pi = intersecao(r,poly)
dp = distancia(ca,pi)
if (dp < d):
    d = dp
    f(x,y) = getColor(p,poly)

    #detectar sombra
    dpiluz = distancia(pi,luz)
    for polysombra in model getPolygons():
        r = ray(pi,luz)
        ps = intersecao(r,polysombra)
        dp = distancia(pi,ps)
        if (dp < dpiluz):
            f(x,y) = sombra
            break
```

Faça valer a pena

1. O pipeline de visualização da Figura 3.2 apresenta cinco elementos de entrada, dentre eles os "vértices de malhas poligonais" e as "malhas poligonais". Os vértices são entrada para o primeiro estágio do pipeline e as malhas são entrada apenas para o segundo estágio. Considera-se que as informações sobre como os vértices estão conectados na malha para formar os polígonos não são necessárias no estágio de transformações geométricas.

Assinale a alternativa que explica porque apenas os vértices das malhas poligonais são usados no estágio de transformações geométricas

a) A malha poligonal define polígonos no espaço contínuo, contendo infinitos pontos, e é inviável computar transformações geométricas para infinitos pontos.
b) A aplicação das transformações geométricas sobre o polígono, e não apenas sobre seus vértices, pode distorcê-lo, deixando-o não planar.
c) O estágio de transformações geométrica não chega a executar a transformação de perspectiva, que é a única transformação que precisa dos polígonos completos.
d) A aplicação de uma transformação geométrica a todos os vértices de um polígono implica na aplicação da mesma transformação geométrica ao polígono.
e) A transformação geométrica de todo o polígono exigiria grande uso de memória, pois a quantidade de polígonos na cena 3D é maior que o número de pixels da tela.

2. O algoritmo de *ray casting* foi utilizado nos primeiros sistemas de síntese de imagens, e nos dias de hoje o algoritmo utilizado é o *ray tracing*. O *ray tracing* é implementado em hardware pelas unidades de processamento gráfico (GPUs) da atualidade.

Assinale a alternativa que faz uma comparação correta entre os algoritmos de *ray casting* e *ray tracing*.

a) O *ray casting* considera que cada polígono da malha é de cor única e calcula apenas a reflexão dessa cor, enquanto o *ray tracing* processa também texturas.
b) O *ray casting* é um caso especial do *ray tracing,* em que todos os objetos são opacos, recebem incidência direta da luz e não recebem luz indireta, refletida por outros objetos.
c) No *ray casting*, para cada pixel da imagem busca-se o polígono mais próximo intersectado pelo raio, e no *ray tracing*, para cada polígono traça-se o raio até a tela.
d) O *ray casting* é definido para malhas triangulares, enquanto o *ray tracing* é uma generalização do *ray casting*, capaz de processar malhas poligonais quaisquer.

3. Em 2012, Agus3D criou uma cena 3D com 19 bilhões de triângulos. Na época a resolução considerada alta era a Full HD, com 1920×1080 pixels, e o computador de Agus3D demorava 40 minutos para digitalizar a cena.
O algoritmo *ray casting* foi apresentado em duas versões. Na primeira versão, para cada pixel da imagem, busca-se o triângulo primeiro (mais próximo), triângulo atingido pelo raio. Na segunda versão, para cada triângulo da cena, projeta-se o raio até todos os pixels da imagem.

Assinale a alternativa que explica corretamente, com base na cena de 19 bilhões de triângulos, qual é a versão mais adequada do algoritmo de *ray casting*.

a) A primeira versão é a mais adequada porque permite a renderização da cena em telas de alta resolução. A imagem Full HD ocupa muita memória. Ao iterar primeiro pelos pixels, cada pixel já processado pode ser descartado da memória para dar lugar a outro.
b) A primeira versão é a mais adequada porque é a única que pode ser paralelizada. O processamento de cada pixel da imagem é independente do processamento dos outros pixels. O algoritmo pode usufruir dos diversos núcleos da GPU.
c) A segunda versão é a mais adequada porque permite a renderização de cenas complexas. A cena completa com 19 bilhões sequer cabe em 16GB de memória. Ao iterar primeiro pelos triângulos, cada triângulo já processado pode ser descartado da memória para dar lugar a outro.
d) A segunda versão é a mais adequada porque é a única que pode ser paralelizada. O processamento de cada triângulo é independente do processamento dos outros triângulos. O algoritmo pode usufruir dos diversos núcleos da GPU.

e) Tanto a primeira quanto a segunda versão do algoritmo são igualmente adequadas. O corpo do *loop* é o mesmo nas duas versões e o que muda é apenas a ordem de visitação dos elementos (pixels e triângulos). A quantidade de operações não muda.

Seção 3.2

Visualização

Diálogo aberto

Caro aluno, nesta seção continuaremos a apresentar os elementos que compõem o pipeline de visualização. O assunto a ser tratado é a obtenção da cor de um ponto projetado. Para tanto, serão apresentados os modelos de iluminação, tonalização e textura.

Os modelos de iluminação representam a física da reflexão da luz. Esses modelos foram obtidos a partir de estudos sobre como a luz é refletida na superfície de diferentes materiais e como ela chega até o ponto de projeção.

Quando a luz reflete sobre um material, irá projetar na tela a cor do material no ponto de reflexão. Porém o material em computação gráfica é representado por uma malha poligonal. Os modelos de tonalização representam como a cor de cada um dos infinitos pontos no interior de um polígono pode ser calculada com base nas cores de cada um dos finitos vértices do polígono. Os modelos de tonalização buscam atribuir diferentes cores aos pontos do interior do polígono sem que seja necessário armazenar a cor de diversos pontos no modelo 3D.

Finalmente, a textura é utilizada para representar faces de polígonos com muitos detalhes. Em vez de utilizar um modelo 3D com milhões de polígonos para representar os detalhes de uma calça jeans, por exemplo, utiliza-se de fato a imagem digital obtida por fotografia de um tecido jeans real. A cor do ponto no interior do polígono é, então, obtida dessa imagem digital.

Considerando que você já implementou o pipeline básico de visualização de forma abstrata e implementou de fato apenas o algoritmo de *ray casting*, agora você deverá incluir efeitos de iluminação, tonalização e texturas. Você deverá implementar os mais simples modelos de iluminação, tonalização e texturas, além de incorporar ao pipeline de visualização já existente. A incorporação dessas implementações no pipeline é feita como realização de protótipos de funções ou métodos abstratos existentes no pipeline. Você deverá entregar o código fonte da implementação desses três modelos.

Boa sorte e ótimo estudo.

Não pode faltar

Caro aluno, na seção anterior vimos como detectar qual polígono da cena é projetado em um determinado pixel da imagem digital, com base na posição da câmera e da fonte de luz no espaço 3D. Agora vamos estudar como calcular a cor do pixel, considerando a cor original do polígono, sua textura e o tipo de sua superfície, além da distância e do tipo da fonte de luz.

O procedimento para determinar a cor de um pixel sob o efeito de uma ou mais fontes de luz é denominado **modelo de iluminação** (HUGHES *et al*., 2013). Um **modelo de iluminação local** é aquele que considera apenas as reflexões diretas da luz sobre os objetos da cena, como tratado pelo algoritmo de ***ray casting***. Um **modelo de iluminação global** considera todas as possíveis radiações que fluem no ambiente, considerando reflexões indiretas e refração da luz por objetos transparentes, como tratado pelo algoritmo de ***ray tracing***. Vamos nos limitar aos modelos de iluminação local, mais simples, já que os modelos de iluminação global exigem um aprofundamento em teorias da física.

Pesquise mais

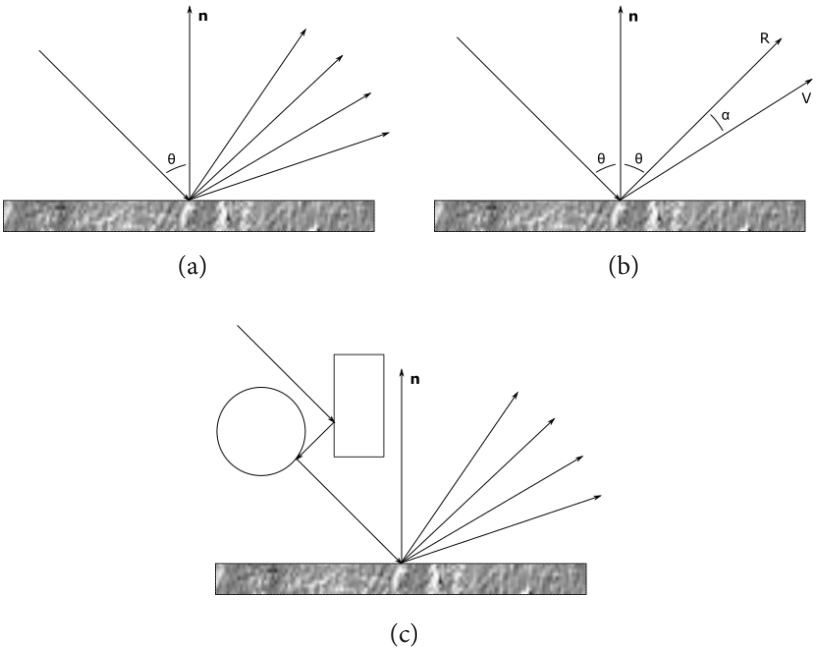
Introdução ao modelo de iluminação global:

WU, S. Traçado de raio**.** *In*: **Síntese de imagens:** uma introdução ao mundo de desenho e pintura dos sistemas digitais.

Um modelo de iluminação local foi proposto por Phong (1975). O modelo Phong é o modelo mais popular, por ser simples e gerar resultados satisfatórios (HUGHES *et al*., 2013). Esse modelo considera a decomposição da luminosidade incidente sobre cada ponto da cena em três componentes: ambiente, difusa e especular.

Já falamos sobre reflexão difusa e especular na Seção 3.1. Agora, vamos explorar um pouco mais em detalhes. A Figura 3.5 ilustra esses três tipos de reflexão.

Figura 3.5 | Três componentes de reflexão do modelo de iluminação de Phong: (a) difusa, (b) especular e (c) ambiente



Fonte: elaborada pelo autor.

A reflexão **difusa** (Figura 3.5(a)) ocorre quando a luz que incide sobre o objeto em um ângulo θ com relação à normal da superfície, reflete em diversos ângulos, com igual intensidade. O quanto cada ponto de projeção receberá de luz proveniente de reflexão difusa dependerá das distâncias do ponto de projeção até o ponto de reflexão e do ponto de reflexão até a fonte de luz, além das características intrínsecas do objeto e da fonte de luz.

A reflexão **especular** (Figura 3.5(b)) ocorre quando a luz que incide sobre o objeto em um ângulo θ com relação à normal reflete em um mesmo ângulo θ com relação à normal. Se o ponto de projeção está sobre a reta R da Figura 3.5(a), receberá a luz refletida em intensidade proporcional às distâncias da fonte de luz ao objeto e do objeto ao ponto de projeção. Se o ponto de projeção se encontra sobre V, sofrerá atenuação que depende do ângulo α.

A reflexão especular proveniente da fonte de luz é dada pela equação $I_s = I_{dir} k_s C_s (\cos\alpha)^s$, onde C_s é a cor especular, k_s é o coeficiente de reflexão especular, s é o expoente de reflexão especular e I_{dir} é a intensidade da cor da luz direta (sem prévia reflexão).

Os parâmetros C_s, k_s e s são intrínsecos ao material e devem constar no modelo 3D. O parâmetro I_{dir} é a cor da fonte de luz com intensidade atenuada pela distância da fonte até o ponto de reflexão. Todos esses parâmetros são definidos empiricamente junto dos parâmetros de reflexão difusa e ambiente que veremos mais adiante.

O expoente s expressa o quão brilhante é o material. Quanto maior o valor de s, mais refletivo é o material, e o ponto de projeção irá perceber mais a cor da fonte de luz do que a cor do objeto. Quanto menor o valor de s, menos refletivo é o material, e o ponto de projeção irá perceber mais a cor do objeto do que a cor da fonte de luz. Um espelho apresenta um expoente s alto, uma bola de tênis, um expoente baixo, e uma bola de bilhar, um expoente intermediário.

O coeficiente k_s expressa o quanto o objeto reflete a luz de forma especular com relação ao quanto reflete de forma difusa. Quanto mais alto o valor k_s, mais o objeto reflete a luz de forma especular. Quanto mais baixo, mais difusa é a reflexão. Mais adiante veremos a relação de k_s com o coeficiente de reflexão difusa.

Para compreender a diferença entre k_s e s, lembremos que a luz que incide sobre o objeto pode ser refletida ou absorvida. O expoente s expressa o quanto de luz é absorvida ou refletida. O coeficiente k_s expressa como ocorre a reflexão, se mais na forma difusa ou especular.

A reflexão difusa proveniente da fonte de luz é dada pela equação $I_d = I_{dir} k_d C_d (\cos\theta)$., onde C_d é a cor difusa, k_d é o coeficiente de reflexão difusa e I_{dir} é a intensidade da cor da luz direta.

Os parâmetros C_d e k_d são intrínsecos ao material, assim como os parâmetros de reflexão especular. O parâmetro I_{dir} é o mesmo da reflexão especular. São todos parâmetros definidos empiricamente. Normalmente os valores de k_s e k_d são percentuais.

Exemplificando

A ferramenta interativa a seguir apresenta sete problemas para praticar e melhor compreender os parâmetros de reflexão difusa ou especular. O primeiro parâmetro (*spotlight*) é a intensidade da fonte luminosa e corresponde a I_{dir}. O segundo parâmetro (*diffuse*) corresponde à relação entre os coeficientes k_s e k_d. O terceiro parâmetro (*specular*) refere-se ao expoente s.

PIXAR IN A BOX. **Praticar:** correspondência das propriedades de iluminação.

Finalmente, a reflexão **ambiente** (Figura 3.5(c)) nada mais é do que uma aproximação global das possíveis reflexões indiretas que possam acontecer na cena. A simplicidade do modelo de iluminação local de Phong está justamente nesse ponto. Em vez de calcular exatamente as reflexões indiretas da luz até chegar ao ponto de projeção, utiliza-se um modelo global.

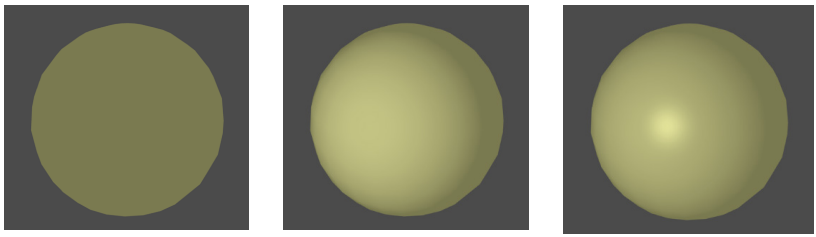
A reflexão ambiente proveniente da fonte de luz é dada pela equação $I_a = I_{ra} k_a C_d$, onde C_d é a cor difusa, k_a é o coeficiente de reflexão ambiente e I_{ra} é a cor da luz refletida no ambiente. A cor da luz ambiente é também definida empiricamente para cada cena e depende do conjunto de objetos da cena e do quão refletivos são esses objetos. O coeficiente k_d normalmente é fixado no mesmo valor de k_d ($k_a = k_d$).

Reflita

Na Seção 3.1 você conheceu os algoritmos *ray casting* e *ray tracing*, sendo que o *ray tracing* é computacionalmente mais complexo do que o *ray casting* por considerar em detalhes todas as reflexões indiretas de luz na cena. Com o modelo de iluminação de Phong, é possível aproximar as reflexões indiretas por um componente de reflexão ambiente e utilizar o *ray casting*. Mas, na prática, qual dos dois é de fato utilizado? Seria uma boa estratégia usar o *ray casting* com o modelo de Phong para pré-visualizações, ao longo do desenvolvimento, e o *ray tracing* para renderizar a cena ou animação final? Pense no desenvolvimento de uma animação longa como um filme completo em computação gráfica.

Agora que você conhece as três componentes de reflexão, podemos definir o modelo de iluminação de Phong para m fontes de luz pela equação $I = I_a + \sum_{i=1}^{m} I_{s,i} + \sum_{i=1}^{m} I_{d,i}$, onde $I_{s,i}$ e $I_{d,i}$ são, respectivamente, as reflexões especular e difusa para a fonte de luz i. Essa equação deve ser aplicada para cada componente R, G e B do sistema de cores. A Figura 3.6 ilustra o resultado dos três tipos de reflexão sobre uma esfera.

Figura 3.6 | Três componentes de reflexão do modelo de iluminação de Phong: (a) apenas reflexão ambiente, (b) reflexão difusa e ambiente e (c) reflexão especular, difusa e ambiente



(a) (b) (c)

Fonte: elaborada pelo autor.

Na Seção 3.1 foi apresentado o código para o algoritmo de *ray casting*, assumindo como já implementada a função *getColor*. Agora podemos implementá-la de fato. O código em Python apresenta a implementação de *getColor* como um método da classe ModeloIluminacaoPhong. Foram consideradas já implementados o método *corAtenuada*, que calcula a intensidade da luz com base na distância da fonte de luz até o ponto de reflexão, e a função *calculaAngulos*, que calcula os ângulos θ e α do modelo de iluminação de Phong.

```
from math import cos

class ModeloIluminacaoPhong:
    def __init__
(self, cs, s, ks, cd, kd, ira, ka, fonteLuz):
        #construtor...

    def componenteEspecular(self,idir,alpha):
        (r,g,b) = self.cs;    (rl,gl,bl) = idir
        r = rl*self.ks*r*pow(cos(alpha),self.s)
        g = gl*self.ks*g*pow(cos(alpha),self.s)
        b = bl*self.ks*b*pow(cos(alpha),self.s)
        return (r,g,b)

    def componenteDifusa(self,idir,theta):
        (r,g,b) = self.cd;    (rl,gl,bl) = idir
        return (rl*self.kd*r*cos(theta),gl*self.
               kd*g*cos(theta),bl*self.kd*b*cos(theta))

    def componenteAmbiente(self):
        (ra,ga,ba) = self.ira; (r,g,b) = self.cd
        return (ra*self.ka*r, ga*self.ka*g, ba*self.ka*b)
```

```
def getColor(self,p,fonteLuz):
    idir = fonteLuz.corAtenuada(p);     (theta,alpha)
    = calculaAngulos(fonteLuz,p)
    (rs,gs,bs) = self.componenteEspecular(idir,alpha)
    (rd,gd,bd) = self.componenteDifusa(idir,theta)
    (ra,ga,ba) = self.componenteAmbiente()
    return ((rs+rd+ra)/3,(gs+gd+ga)/3,(bs+bd+ba)/3)
```

Blinn (1977) propôs uma pequena extensão do modelo de Phong para reduzir a quantidade de cálculos para objetos muito distantes do ponto de projeção (WU, 2009). Como os princípios do modelo de Phong foram mantidos por Blinn, o modelo muitas vezes é referenciado na literatura sob o nome de modelo Blinn-Phong.

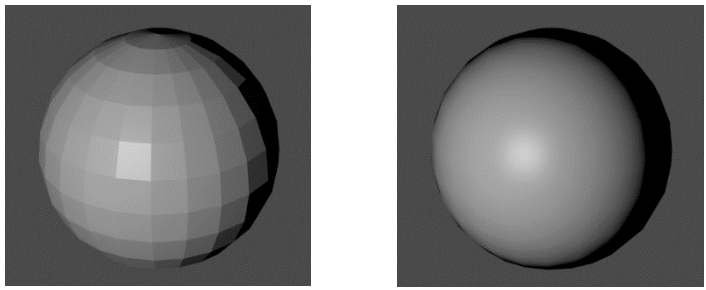
Tonalização

Modelos de tonalização ou *shading* são modelos de iluminação que visam melhorar os resultados visuais da aplicação dos algoritmos de *ray tracing*. No trecho de código em Python apresentado, a cor considerada foi a cor do polígono, e essa cor foi atenuada de acordo com os parâmetros intrínsecos do objeto (se mais especular ou mais difuso) e de acordo com os ângulos de incidência dos raios de luz.

Como vimos, as equações que determinam as componentes especular e difusa da luz dependem de dois ângulos: i) o ângulo do raio traçado a partir do centro de projeção e ii) o ângulo do raio proveniente da fonte de luz. Ambos os ângulos são com relação à normal do polígono no ponto de interseção do raio com o polígono.

Considerando que os modelos poligonais são construídos com polígonos planos, todos os vetores normais do polígono são paralelos, qualquer seja o ponto em seu interior. Logo, todos os raios que atinjam um mesmo polígono serão da mesma cor. Assim, a cor é calculada pelas equações vistas anteriormente apenas para um vértice do polígono e o valor obtido é copiado para todos os outros pontos do mesmo polígono. Este é o modelo de tonalização constante, ilustrado na Figura 3.7(a).

Figura 3.7 | Modelos de tonalização: (a) constante e (b) interpolado



(a) (b)

Fonte: elaborada pelo autor.

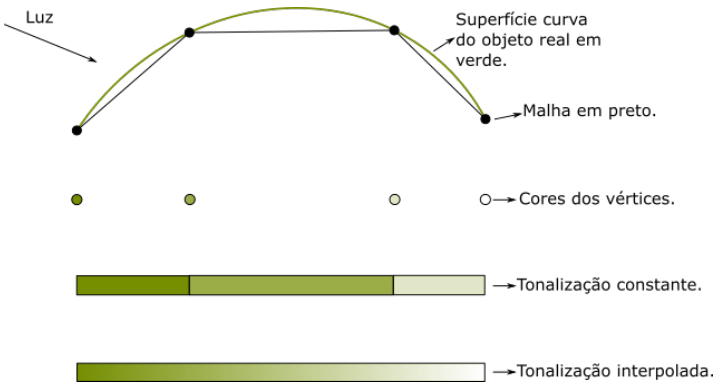
Para que um objeto se torne mais realista do que o que aparece na Figura 3.6(a), há duas medidas a serem tomadas: i) aumentar a resolução do modelo, utilizando um número muito maior de vértices e polígonos ou ii) utilizar o conceito de superfícies curvas e tonalização interpolada. A primeira opção aumenta em muito o custo computacional de visualização 3D e ainda nem sempre resolve o problema. A segunda opção é a mais adotada na prática.

A ideia de superfícies curvas é baseada nos conceitos de *splines*. Em vez de os vértices do polígono representarem polígonos planos, passam a representar superfícies *spline*, construídas com base nos vértices dos polígonos vizinhos. A Figura 3.8 ilustra essa estratégia para uma malha unidimensional, com segmentos de reta no lugar de polígonos.

Assimile

Com os modelos de tonalização interpolada, os polígonos planos são aproximados a superfícies *spline*, permitindo que o objeto seja renderizado com aparência suave, sem que para isso precise ser modelado por uma malha com milhões de polígons.

Figura 3.8 | Ilustração unidimensional da tonalização constante e da tonalização interpolada com a estratégia de superfícies curvas



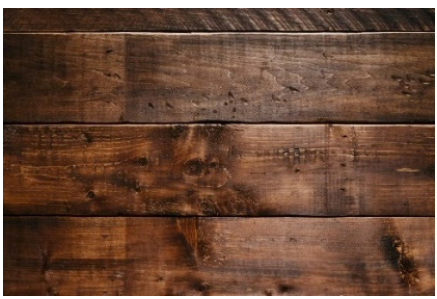
Fonte: adaptada de Hughes ***et al.*** (2013).

Com a tonalização interpolada da Figura 3.8 a cor não está mais associada ao segmento de reta, mas sim aos vértices do segmento. A cor de cada ponto do segmento de reta é obtida por interpolação. Observa-se na figura um dégradé de cores obtido a partir de apenas quatro cores dos vértices. Há diversos modelos de tonalização interpolada, sendo os mais conhecidos os modelos de Gouraud (1971) e de Phong (1975). Esses modelos calculam a cor de um ponto do polígono com base nas cores dos vértices ou dos polígonos vizinhos. A Figura 3.7(b) mostra a renderização, utilizando tonalização interpolada, do mesmo modelo da Figura 3.7(a).

Texturas

As técnicas de tonalização interpolada permitem que objetos sintéticos sejam visualizados com cores mais suaves e realistas. Porém se for de interesse criar um modelo 3D que contenha uma grande parede de madeira, seriam necessários milhões de polígonos. O mais recomendado neste caso é associar ao polígono uma **textura**, como a da Figura 3.9.

Figura 3.9 | Exemplo de textura de madeira



Fonte: freestocktextures.com. Acesso em: 10 jan. 2019.

A textura é uma imagem bidimensional, que pode ser uma fotografia de um objeto real. Essa imagem bidimensional é sobreposta ao polígono como um papel de parede. Para obter a cor de um ponto no interior do polígono faz-se o mapeamento das coordenadas dos vértices do polígono sobre a imagem de textura e os pontos internos do polígono são extraídos dessa imagem. Dessa forma, com um modelo 3D de poucos polígonos é possível criar uma casa de madeira como a da Figura 3.10, utilizando texturas de madeira similares às da Figura 3.9.

Figura 3.10 | Casa de madeira utilizando mapeamento de texturas

Fonte: free3d.com. Acesso em: 10 jan. 2019.

Nesta seção você foi apresentado aos modelos de iluminação, tonalização e textura. As figuras foram produzidas com o software livre Blender e as instruções para produzi-las podem ser encontradas nos capítulos 20, 25, 34 e 37 da obra de Blender (2016). A implementação desses modelos exige muitas linhas de código. Por essa razão, na Seção 3.3 você irá praticar os conceitos desta seção utilizando bibliotecas de desenvolvimento e adicionará a dimensão tempo para criar animações gráficas 3D.

Sem medo de errar

A você foi atribuída a missão de implementar os mais simples modelos de iluminação, tonalização e texturas, para serem incorporados ao pipeline de visualização já existente. A incorporação dessas implementações no pipeline é feita como realização de protótipos de funções ou métodos abstratos existentes no pipeline. Você deverá entregar o código fonte da implementação desses três modelos.

Primeiramente, você precisa completar o código do modelo de iluminação de Phong apresentado nesta seção:

```
from math import cos

class ModeloIluminacaoPhong:
    def __init__
(self, cs, s, ks, cd, kd, ira, ka, fonteLuz):
        self.s = s
```

```
        self.cs = cs
        self.s = s
        self.ks = ks
        self.cd = cd
        self.kd = kd
        self.ira = ira
        self.ka = ka
        self.fonteLuz = fonteLuz

    def componenteEspecular(self,idir,alpha):
        (r,g,b) = self.cs
        (rl,gl,bl) = idir
        r = rl*self.ks*r*pow(cos(alpha),self.s)
        g = gl*self.ks*g*pow(cos(alpha),self.s)
        b = bl*self.ks*b*pow(cos(alpha),self.s)
        return (r,g,b)

    def componenteDifusa(self,idir,theta):
        (r,g,b) = self.cd
        (rl,gl,bl) = idir
        r = rl*self.kd*r*cos(theta)
        g = gl*self.kd*g*cos(theta)
        b = bl*self.kd*b*cos(theta)
        return (r,g,b)

    def componenteAmbiente(self):
        (ra,ga,ba) = self.ira
        (r,g,b) = self.cd
        r = ra*self.ka*r
        g = ga*self.ka*g
        b = ba*self.ka*b
        return (r,g,b)

    def getColor(self,p,fonteLuz):
        idir = fonteLuz.corAtenuada(p)
        (theta,alpha) = calculaAngulos(fonteLuz,p)
        (rs,gs,bs) = self.
componenteEspecular(idir,alpha)
        (rd,gd,bd) = self.
componenteDifusa(idir,theta)
        (ra,ga,ba) = self.componenteAmbiente()
        return ((rs+rd+ra)/3,(gs+gd+ga)/3,
                (bs+bd+ba)/3)
```

Em seguida deve acrescentar um modelo de tonalização. O código fonte acima já trabalha com um modelo de tonalização constante, para toda a malha, já que os parâmetros cs, cd e ira se tornam atributos do modelo. Não há mais nada a fazer.

Finalmente, você deve, então, acrescentar um modelo de mapeamento de texturas. Para isso, você deverá modificar os métodos componenteEspecular, componenteDifusa e componenteAmbiente para verificar se os parâmetros cs, cd e ira são constantes, como já implementado, ou se são provenientes de uma textura. Se for proveniente de textura, você deverá implementar o mapeamento. Para implementar esse mapeamento você deve:

1. Fazer duas rotações 3D no ponto de reflexão para que a normal do polígono que o contém fique paralela ao eixo Z do mundo.
2. Considerar a imagem digital da textura como coincidente com o plano XY do mundo.
3. Mapear a coordenada do ponto à coordenada da imagem de textura e obter a cor do pixel a partir da imagem.

A implementação do mapeamento de textura irá utilizar os conhecimentos da Unidade 2.

Avançando na prática

Criação de modelo 3D com objetos brilhantes e foscos

Descrição da situação-problema

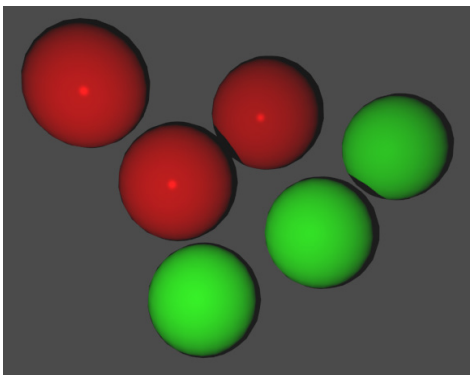
Você foi requisitado para criar uma cena com diversas bolas. Entre as bolas, encontram-se bolas de tênis e de bilhar. O modelo deve ser criado utilizando um software de modelagem. Apresente a descrição do passo a passo para criar esse ambiente e preferencialmente apresente uma imagem que demonstre que você soube aplicar o seu passo a passo no software de modelagem.

Resolução da situação-problema

A sugestão é utilizar o software Blender de modelagem tridimensional. Os pontos da documentação do Blender, necessários para solucionar esta situação, foram citados nesta seção. O passo a passo que você deve executar é:

1. Criar uma cena.
2. Adicionar uma fonte de luz.
3. Criar uma esfera (Esfera 1).
4. Para a Esfera 1 atribuir a cor de uma bola de bilhar, atribuir um alto coeficiente de reflexão especular e também um alto coeficiente de reflexão difusa. Atribuir um alto expoente de reflexão especular.
5. Fazer uma cópia da Esfera 1 (Esfera 2).
6. Para a Esfera 2 atribuir a cor de uma bola de tênis, atribuir um coeficiente de reflexão difusa médio e um baixo coeficiente de reflexão especular. Atribuir um baixo expoente de reflexão especular.
7. Fazer várias cópias da Esfera 1 (bolas de bilhar) e várias cópias da Esfera 2 (bolas de tênis).

A Figura 3.11 mostra um exemplo dessa aplicação utilizando o software Blender.



Fonte: elaborada pelo autor.

Faça valer a pena

1. O modelo de iluminação de Phong é capaz de gerar resultados satisfatórios por uma abordagem bastante simplificada. O modelo de Phong considera que a luz que chega a algum ponto da tela é proveniente de três componentes: uma componente especular, uma difusa e a reflexão ambiente.

Qual é a principal simplificação do modelo de Phong se comparado ao algoritmo de *ray tracing*? Assinale a alternativa correta.

a) Os coeficientes de reflexão especular (k_s) e difusa (k_d) do modelo de Phong não representam fielmente as características refletivas dos objetos.
b) O expoente de reflexão especular (s) foi adicionado ao modelo Phong para substituir as equações de refração em objetos transparentes.
c) O modelo de Phong só é capaz de lidar com o modelo de tonalização (*shading*) constante. Não suporta tonalização interpolada ou textura.
d) O coeficiente de reflexão ambiente (k_a) de Phong deve ser sempre igual ao coeficiente de reflexão difusa (k_d), uma aproximação grosseira.
e) A componente de reflexão ambiente é uma aproximação das múltiplas reflexões indiretas da luz sobre os diversos objetos da cena.

2. O modelo de tonalização constante gera um efeito indesejado, tornando visíveis os polígonos do modelo tridimensional. Para suavizar a superfície do objeto na visualização, utiliza-se a tonalização interpolada, proposta inicialmente por Gouraud e aperfeiçoada por Phong.

Sobre os modelos de tonalização interpolada, assinale a alternativa correta.
a) Os modelos de tonalização interpolada exigem malhas poligonais muito detalhadas, com milhões de polígonos.
b) Os modelos de tonalização interpolada associam a cada ponto do polígono o mesmo valor da cor do 'vértice de referência'.
c) O modelos de tonalização de Phong adiciona ao componente de iluminação difusa um componente de iluminação ambiente.
d) Os modelos de tonalização interpolada calculam a cor do ponto com base nas cores dos vértices do polígono ou de polígonos vizinhos.
e) A interpolação de Phong consiste em calcular a cor do vértice pela somatória das componentes especular, difusa e ambiente.

3. Observe o código fonte a seguir:

```
from math import cos

class Modelo:
    def __init__
(self, cs, s, ks, cd, kd, ira, ka, fonteLuz):
        #construtor...

    def m(self,idir,angle):
        (r,g,b) = self.c;    (rl,gl,bl) = idir
        r = rl*self.k*r*pow(cos(angle),self.e)
        g = gl*self.k*g*pow(cos(angle),self.e)
        b = bl*self.k*b*pow(cos(angle),self.e)
        return (r,g,b)
```

A classe Modelo é a implementação de um modelo de iluminação, tonalização ou textura.

Assinale a alternativa que afirma corretamente qual é esse modelo e o que faz o método *m*.

a) Trata-se do modelo de iluminação de Phong, e o método *m* implementa o cálculo da componente especular, como pode ser observado no uso do coeficiente k e do expoente e de reflexão especular no cálculo de r, g e b.
b) Trata-se do modelo de tonalização de Gouraud, e o método *m* implementa o cálculo da interpolação das cores dos vértices, como pode ser observado no uso do cosseno nas linhas que atribuem valores a r, g e b.
c) Trata-se do modelo de iluminação de Phong, e o método *m* implementa o cálculo da componente difusa, como pode ser observado nas linhas que fazem o mapeamento das cores r, g e b para a imagem de textura.
d) Trata-se do modelo de iluminação de Phong, e o método *m* implementa o cálculo da componente difusa, como pode ser observado no uso do coeficiente k e do expoente e de reflexão difusa no cálculo de r, g e b.

Seção 3.3

CGPI: animação

Diálogo aberto

Caro aluno, nas seções anteriores você foi apresentado aos conceitos teóricos e propôs implementações abstratas do pipeline de visualização. Até o momento você foi apresentado a códigos em linguagem de programação como forma de ilustrar como tais conceitos teóricos poderiam ser programados, mas com a quantidade de código desenvolvida até o momento, ainda há muito a ser desenvolvido para se concluir um pipeline de visualização completo. A implementação de todo o código para a concretização do pipeline de visualização exigiria um grande número de linhas de código.

Por esse motivo, esta seção se apoia em uma biblioteca de processamento gráfico, com a qual você poderá trabalhar focado na realização, em linguagem de programação, dos conceitos teóricos já vistos. Nesta seção você será apresentado a essa biblioteca de processamento 3D e realizará atividades práticas de implementação de um vídeo simples de animação.

Lembre-se que você foi alocado na equipe de desenvolvimento de um aplicativo de modelagem tridimensional. Com base nos argumentos acima, você vai considerar que já possui o pipeline de visualização contendo transformações geométricas e efeitos de iluminação. Ao acrescentar a dimensão tempo será possível criar uma animação gráfica. Nessa última etapa você está encarregado de implementar um modelo de movimento de uma câmera virtual em um modelo geométrico com todos os outros objetos 3D fixos. O modelo de movimento que você deverá implementar é a translação de uma câmera virtual em velocidade fixa ao longo de uma reta no espaço tridimensional. Para gerar a animação gráfica, você deverá considerar o número de quadros a serem gerados. E para cada posição da câmera virtual, você deverá aplicar o pipeline básico de visualização para gerar uma imagem 2D, que será um quadro do vídeo de animação. Você deverá entregar o código fonte que implementa esse tipo de animação.

Você perceberá o quanto as bibliotecas de processamento gráfico podem ajudar a desenvolver animações e isso poderá abrir um novo leque de oportunidades profissionais, seja na indústria de jogos, filmes, ou até mesmo de publicidade e propaganda.

Bom trabalho!

Não pode faltar

As animações gráficas são, talvez, os produtos mais atraentes da computação gráfica. Para produzir uma animação, todos os conceitos de síntese de imagens e modelagem tridimensional são necessários. Além da criação do modelo 3D, é preciso associar a esse modelo uma sequência de transformações ao longo do tempo e, para cada instante, executar o pipeline de visualização para sintetizar um quadro da animação.

Só o pipeline de visualização visto nesta unidade já exige uma grande quantidade de linhas de código. O processo de criação do modelo 3D exige também ferramentas computacionais, de captura de imagens com múltiplas câmeras ou de interação homem-computador para o desenho manual de modelos por artistas.

Jogos digitais, vinhetas de TV ou filmes de animação na qualidade dos que são produzidos hoje só podem ser criados com o auxílio de aplicativos de criação e bibliotecas. O profissional de computação deve compreender os fundamentos da computação gráfica e também deve ter capacidade de aprender a usar as bibliotecas do mercado. Nesta seção você irá praticar o uso de uma ferramenta e uma biblioteca de processamento gráfico e adquirirá essa aptidão.

Há centenas de bibliotecas de computação gráfica tridimensional, gratuitas ou comerciais. Grande parte dessas bibliotecas são de alto nível, construídas utilizando outras bibliotecas, as bibliotecas de baixo nível. As bibliotecas de baixo nível implementam os algoritmos mais fundamentais, como as transformações geométricas, geração de malhas, modelos de iluminação e o pipeline de visualização. As bibliotecas de alto nível disponibilizam ferramentas de mais fácil uso para se criar e manipular objetos, aplicar características a materiais, aplicar efeitos de iluminação e animação (HEARN *et al.*, 2010).

Entre as bibliotecas de baixo nível podemos citar a OpenGL (ANGEL; SHREINER, 2012) e a Direct3D (STENNING, 2014). São bibliotecas utilizadas pelo mercado e bastante populares. OpenGL é multiplataforma e livre, enquanto Direct3D é proprietária e direcionada para sistemas Microsoft Windows®. Ambas (e suas concorrentes) implementam algoritmos otimizados na unidade de processamento gráfico (*Graphics Processing Unit – GPU)*. Há ainda bibliotecas de baixo nível que implementam o estado da arte das pesquisas em computação gráfica, como é o caso da Computational Geometry Algorithms Library – CGAL (THE CGAL PROJECT, 2018).

Reflita

Você acredita que as ferramentas gratuitas como a OpenGL e a CGAL são utilizadas pela mais avançada indústria do cinema para produzir suas animações? Quanto aos algoritmos nelas presentes, são mesmo os mais avançados ou as grandes empresas utilizam algoritmos não publicados mais eficientes do que os já publicados em artigos científicos? Observe que as grandes empresas são muitas vezes patrocinadoras dos eventos científicos e suas equipes técnicas são frequentadoras desses eventos.

O número de bibliotecas de alto nível é bem mais amplo, pois elas se especializam na interação com um público-alvo específico (criação de jogos, projetos de arquitetura, engenharia, animação 3D, publicidade e propaganda). Em geral as bibliotecas de alto nível são adotadas por um aplicativo de criação destinado ao público-alvo, como é o caso do Blender (BLENDER, 2016).

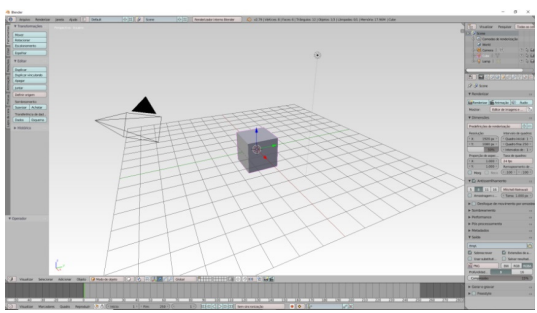
Blender é um aplicativo completo de criação. Implementa modelagem, renderização e animação. É livre, desenvolvido em C, C++ e Python, com uso da OpenGL. Provê uma biblioteca de alto nível em Python, utilizada nesta seção. O aplicativo possui uma interface gráfica interativa, mas buscaremos minimizar o uso dessa interface, priorizando o uso da biblioteca Python.

Processamento de modelos 3D

Para realizar o processamento de modelos 3D em Python com o Blender é essencial que o desenvolvedor tenha acesso ao manual da *Application Programming Interface* (API) Python do Blender (BLENDER, 2016).

Ao inicializar o Blender, o aplicativo cria um modelo inicial contendo um cubo, uma câmera e uma fonte de luz, como mostra a Figura 3.12. Na figura você visualiza menus verticais à esquerda e à direita da tela, e duas áreas. A área maior em posição superior é a visualização 3D, e a área menor em posição inferior, é uma linha de tempo.

Figura 3.12 | Criando um modelo 3D no Blender



Fonte: captura de tela do software Blender 2.79b.

Você pode alterar a ferramenta de uma área clicando no botão do extremo inferior à esquerda da área. Selecionando *Editor de texto* você pode ajustar o tamanho da área e adicionar um novo arquivo de texto clicando no botão +. O arquivo pode ser salvo com extensão .py usando o menu *Texto->Salvar como*. Você pode inserir o bloco de texto a seguir e clicar em *Executar script*.

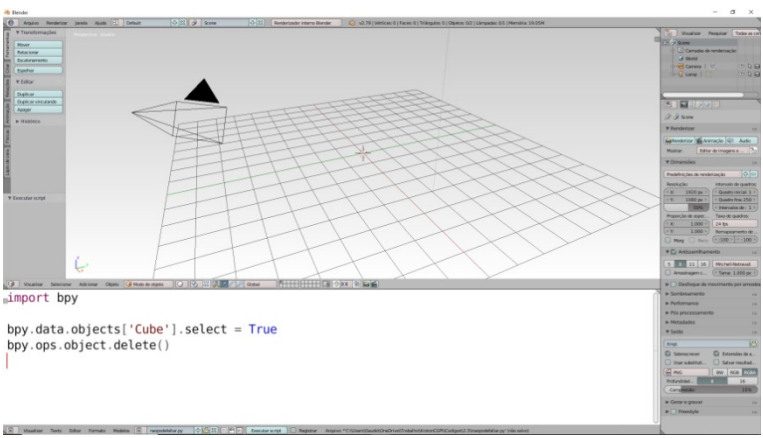
```
import bpy

bpy.data.objects['Cube'].select = True
bpy.ops.object.delete()
```

No código acima, *bpy* é a API Python do Blender. Na página inicial da documentação (https://docs.blender.org/api/current) são apresentados os módulos da API. O módulo *bpy.data* contém os dados do modelo e o módulo *bpy.ops*, um conjunto de operações a serem aplicadas sobre o modelo. A primeira linha do código acima seleciona o objeto de nome 'Cube', e a segunda linha apaga esse objeto. O resultado obtido é mostrado na Figura 3.13.

Assimile

A primeira página da documentação da API Python do Blender apresenta todos os módulos. Ao clicar em um módulo, são apresentados os submódulos. É essencial navegar por esses módulos para compreender a estrutura da API e conhecer possibilidades que vão além das apresentadas nesta seção ou em exemplos da Internet.

Figura 3.13 | Modelo do Blender após remoção do cubo do modelo inicial



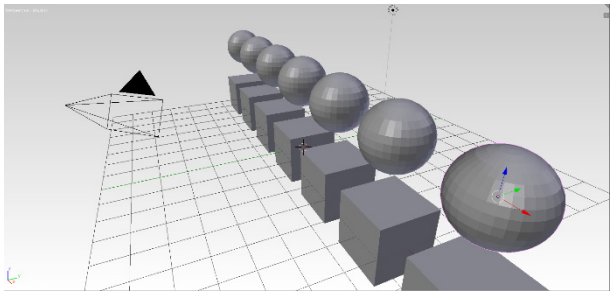
Fonte: captura de tela do software Blender 2.79b.

Note que a API Python permite a interação com o modelo sem restrições. Vamos, agora, acrescentar novos cubos e esferas. Cubos nas coordenadas (i,0,0) e esferas nas coordenadas (i,0,3), variando i e utilizando o módulo *bpy.ops.mesh*, que é o módulo de operações sobre malhas. Adicione um segundo script Python na área de editor de texto para testar o código a seguir (você pode executar um script de cada vez).

```
import bpy

for i in range(-9,10,3):
    bpy.ops.mesh.primitive_cube_
add(location=(i,0,0))
    bpy.ops.mesh.pri
```

O resultado obtido é apresentado na Figura 3.14.

Figura 3.14 | Inserção de objetos



Fonte: captura de tela do software Blender 2.79b.

A função *bpy.ops.mesh.primitive_cube_add* possui diversos parâmetros e permite criar esferas com uma malha com diferentes números de polígonos. Verifique a documentação.

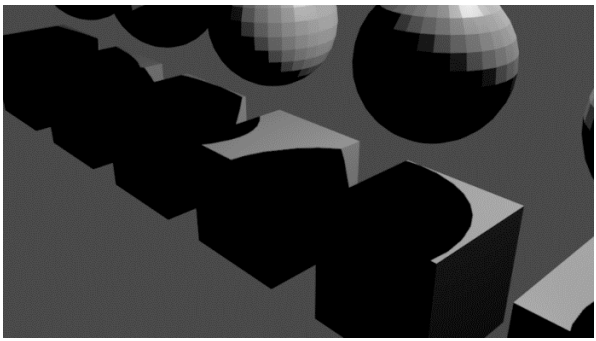
Você pode apagar da cena os objetos por tipo. Para apagar todos as malhas 3D do modelo, sem apagar a câmera e a fonte de luz, crie um terceiro script Python.

```
import bpy

bpy.ops.object.select_by_type(type='MESH')
bpy.ops.object.delete()
```

Você pode obter uma renderização da cena na câmera pelo Python com a função *bpy.ops.render.render(use_viewport=True)*, que realiza o pipeline de

visualização, incluindo transformações geométricas, efeitos de iluminação, *ray tracing*, para gerar um arquivo de imagem 2D no diretório temporário. Para visualizar a imagem renderizada, utilize o *Editor de imagens e UVs* que pode ser acessado pelo mesmo botão que acessa o *Editor de texto*. Pela interface gráfica, simplesmente pressione F12 para exibir o *Editor de imagens e UVs*. Para sair dessa visualização, pressione ESC. A Figura 3.15 mostra a renderização da cena da Figura 3.14.

Figura 3.15 | Renderização da cena.



Fonte: captura de tela do software Blender 2.79b.

Aplicação de transformações geométricas sequenciais

Como apresentado na Unidade 2, a aplicação sequencial de transformações geométricas gera acúmulo de erros. Por esse motivo, a API do Blender sempre realiza transformações geométricas sobre os objetos a partir da origem do sistema de coordenadas do mundo. Um objeto (malha, câmera ou fonte de luz) é representado pelo tipo *bpy.types.Object*, que possui atributos como *bpy.types.Object.location*, que define a translação do objeto a partir da origem, *bpy.types.Object.rotation_euler*, que define a rotação do objeto em torno da origem antes de se realizar a translação. Para transformações geométricas mais complexas é possível alterar o sistema de coordenadas do mundo para o objeto, por meio do atributo *bpy.types.Object.matrix_world*, que é uma matriz 4×4 de transformação afim.

Para se realizar uma sequência de transformações geométricas, com a finalidade de representar um movimento ou deformação do objeto, é preciso calcular previamente todas as localizações e rotações sequenciais aplicadas ao objeto. Basta criar uma lista de localizações ou rotações e alterar os atributos do objeto em um laço de repetição. Experimente executar o script que apaga todos os objetos do tipo malha e criar um novo script com o código abaixo.

```
import bpy

bpy.ops.mesh.primitive_uv_sphere_
add(location=(0,0,0))
esfera = bpy.context.object
locations = [(0,0,-2),(0,0,-
1),(0,0,0),(0,0,1),(0,0,2)]
for l in locations:
    esfera.location = l
```

No script acima, *bpy.context.object* retorna o último objeto selecionado. Ao executar esse script, você verá como resultado apenas o cubo na posição (0,0,2), que é a última posição da lista, mas o objeto foi movido para todas as posições da lista. Para mover o objeto ao longo de uma curva mais complexa, é preciso construir uma lista de posições pela equação da curva. Compete ao desenvolvedor criar essa curva com base no modelo físico que deseja simular.

Exemplificando

Você pode fazer a esfera se mover ao longo de um círculo com coordenada Z constante. Basta lembrar da representação do círculo 2D em coordenadas polares. Para criar um conjunto de localizações sobre um círculo, faça uma iteração por vários ângulos θ, entre 0 e 2π, e acrescente a uma lista *l* a coordenada $(r\cos(\theta), r\sin(\theta), Z)$. Em Python, considerando tt=θ, use *l.append((r*math.cos(tt),r*math.sin(tt),Z))*.

Aplicação de efeitos de iluminação sequenciais

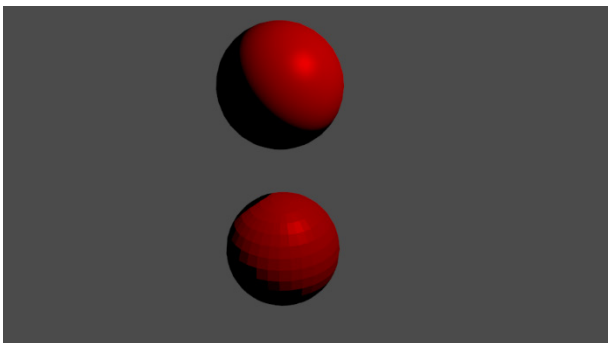
As esferas da Figura 3.15 foram renderizadas utilizando um modelo de tonalização constante. É possível perceber que cada polígono das esferas está com cor constante. Esse modelo de tonalização pode ser alterado para utilizar superfícies curvas, com o uso da função *bpy.ops.object.shade_smooth*. Essa função não altera a malha do objeto, apenas o modelo de tonalização. Também é possível atribuir a um objeto um material, definindo para o material as características de cor e reflexão com base no modelo de iluminação de Phong (1975). O material possui as características de cor e intensidade das componentes de reflexão especular e difusa. A cor e a intensidade da componente de reflexão ambiente são atributos da cena. Vamos apagar novamente as malhas da cena e criar um script Python com características de materiais.

```
import bpy

bpy.ops.mesh.primitive_uv_sphere_
```

```
add(location=(0,0,2))
bpy.ops.object.shade_smooth()
esfera1 = bpy.context.object

bpy.ops.mesh.primitive_uv_sphere_
add(location=(0,0,-1))
esfera2 = bpy.context.object

material = bpy.data.materials.new("brilhante")
material.diffuse_color = (1,0,0)
material.specular_color = (1,0,0)
material.diffuse_intensity = 0.8
material.specular_intensity = 0.8

esfera1.data.materials.append(material)
esfera2.data.materials.append(material)
```

Note que a ambas as esferas foi atribuído o mesmo material, mas à Esfera 1 foi atribuído o modelo de tonalização interpolado. A Figura 3.16 mostra a renderização dos objetos criados acima.

Figura 3.16 | Renderização com tonalização constante e interpolada



Fonte: captura de tela do software Blender 2.79b.

É possível também modificar as características de um objeto ao longo do tempo, para fazer um objeto brilhante ir se tornando mais fosco ou vice-versa, como feito para as posições.

Criação de animação gráfica

A combinação da aplicação de transformações geométricas e efeitos de iluminação sequenciais permite a criação de uma animação gráfica. A animação gráfica é uma sequência de quadros. Para criar uma animação, basta definir um número total de quadros e associar a cada objeto em

movimento um conjunto de quadros de referência. O movimento entre dois quadros de referência é construído pela própria API. O código a seguir cria uma animação do movimento simples de uma esfera. Lembre-se de apagar todos os objetos da cena antes de executar este script.

```
import bpy

cena = bpy.context.scene
cena.frame_end = 50

bpy.ops.mesh.primitive_uv_sphere_
add(location=(0,0,0))
esfera = bpy.context.object

locations = [(0,0,-2),(0,0,-
1),(0,0,0),(0,0,1),(0,0,2)]
quadro = 0
for l in locations:
    cena.frame_set(quadro)
    esfera.location = l
    esfera.keyframe_insert(data_path="location")
    quadro += 10
```

O código acima cria uma animação de 50 quadros, inserindo 5 quadros de referência, um a cada 10 quadros da animação. Os quadros intermediários são criados pela API. Para renderizar a animação use Ctrl+F12 e para visualizá-la, Ctrl+F11.

Saiba mais

Animações podem ser executadas por scripts Python, utilizando o pacote Blender Game Engine (*bge*).

BGE VICIO. **Blender 2.60 Tutorial Python Executar Animações Sem Actuators #11**.

Os mesmos movimentos das malhas 3D podem ser aplicados à fonte de luz e à câmera. Na atividade prática desta seção você irá atribuir movimento à câmera. Todas as APIs de processamento gráfico de alto nível oferecem ferramentas similares à API Python do Blender. Com o conteúdo prático desta seção e os conhecimentos teóricos adquiridos até agora, você tem condições de explorar com facilidade qualquer outra API entre as centenas existentes no mercado. Explore a documentação do Blender e extrapole o que é sugerido na atividade prática desta seção. Surpreenda! O mercado de criação de conteúdo gráfico está sempre à

busca de profissionais de computação para auxiliar os artistas e designers. Você pode ser a diferença.

Sem medo de errar

O contexto em que você está inserido é o de uma equipe de desenvolvimento de um aplicativo de modelagem tridimensional. Você está encarregado de implementar um modelo de movimento de uma câmera virtual em um modelo geométrico com todos os outros objetos 3D fixos, considerando que já possui todo o pipeline de visualição implementado e disponível. O movimento que você deverá implementar é a translação de uma câmera virtual em velocidade fixa ao longo de uma reta no espaço tridimensional. Para gerar a animação gráfica, você deverá considerar o número de quadros a serem gerados e renderizar um quadro para cada posição da câmera virtual, gerando ao final um vídeo de animação. Você deverá entregar o código fonte que implementa esse tipo de animação.

Como você deve considerar já implementadas todas as etapas do pipeline, a solução para esta situação-problema exige o uso de uma biblioteca de processamento gráfico. Nesta seção você foi apresentado à API Python do Blender e essa API é suficiente para resolver o problema.

Ao longo da seção você foi apresentado a um código fonte para criar uma animação simples que movimenta uma esfera ao longo de um caminho descrito por uma lista de pontos. Você precisa simplesmente alterar o código da seção nos seguintes itens:

1. Os pontos da lista devem necessariamente pertencer a uma reta.
2. O objeto a ser movimentado agora é a câmera, e não a esfera.

Vamos começar pelo Item 1. Cabe a você definir a equação da reta. O código fonte a seguir cria uma sequência de pontos sobre a reta $Z = 1{,}5Y + 8$. Essa reta está totalmente contida num plano com X fixo.

```
locations = []
for Y in range(-5,5):
    locations.append((X,Y,1.5*Y+8))
```

Agora vamos modificar o código fonte apresentado na seção para acrescentar o trecho acima e atribuir o movimento à câmera e não aos objetos. O código a seguir soluciona todo o problema.

```
import bpy

cena = bpy.context.scene
cena.frame_end = 100

bpy.ops.object.select_by_type(type='MESH')
bpy.ops.object.delete()
bpy.ops.mesh.primitive_uv_sphere_
add(location=(-7,7,0))

camera = bpy.data.objects['Camera']
(X,Y,Z) = camera.location

#Z=1.5*Y+8
locations = []
for Y in range(-5,5):
    locations.append((X,Y,1.5*Y+8))

quadro = 0
for l in locations:
    cena.frame_set(quadro)
    camera.location = l
    camera.keyframe_insert(data_path="location")
    quadro += 10
```

Esse script pode ser executado e a animação renderizada no Blender.

Com essa solução você irá demonstrar a capacidade de criar animações simples a partir de modelos 3D, e você tem conhecimento suficiente para explorar outras funcionalidades da biblioteca do Blender ou mesmo outras bibliotecas.

Avançando na prática

Movimentar uma fonte de luz de forma circular

Descrição da situação-problema

Você agora foi contratado para criar uma animação que simula a movimentação de uma fonte de luz em torno de uma esfera, e mantém a câmera e a esfera fixas. A animação tem por objetivo ilustrar a movimentação das componentes de reflexão especular e difusa e das sombras, com base na posição da fonte de luz.

Resolução da situação-problema

A resolução dessa nova situação-problema é uma pequena modificação da situação-problema anterior, em dois itens:

1. Os pontos da lista de posições da fonte de luz devem pertencer a um círculo.
2. O objeto a ser movimentado agora é a fonte de luz, e não a câmera.

Vamos começar pelo Item 1, definindo um número de quadros de referência para 16 e criando uma lista de pontos sobre um círculo de raio 5 no plano Z=6.

```
Z = 6; r = 5
locations = []; tt = 0; num=16
for i in range(num):
    locations.append((r*cos(tt),r*sin(tt),Z))
    tt += 2*3.14/num
```

Agora vamos modificar o código fonte apresentado na situação-problema da seção e alterar devidamente o movimento. O código a seguir soluciona todo o problema.

```
import bpy
from math import cos, sin

bpy.ops.object.select_by_type(type='MESH')
bpy.ops.object.delete()

cena = bpy.context.scene

bpy.ops.mesh.primitive_uv_sphere_
add(location=(0,0,0))

luz = bpy.data.objects['Lamp']

Z = 6; r = 5
locations = []; tt = 0; num=16
for i in range(num):
    locations.append((r*cos(tt),r*sin(tt),Z))
    tt += 2*3.14/num

cena.frame_end = 10*num
quadro = 0
for l in locations:
    cena.frame_set(quadro)
```

```
luz.location = l
luz.keyframe_insert(data_path="location")
quadro += 10
```

Faça valer a pena

1. Há centenas de bibliotecas de processamento gráfico disponíveis no mercado. Algumas são bibliotecas de baixo nível, que implementam as operações fundamentais que incluem transformações geométricas, modelos de iluminação e algoritmos de *ray tracing*. Outras são de alto nível, que implementam funções voltadas a um público-alvo específico.

Sobre as bibliotecas de processamento gráfico, assinale a alternativa correta.

a) OpenGL e Direct3D são bibliotecas de alto nível porque implementam algoritmos em GPU e com isso conseguem ser mais eficientes.
b) Aplicativos de arquitetura ou engenharia utilizam bibliotecas de alto nível que, por sua vez, utilizam bibliotecas de baixo nível.
c) Profissionais de criação 3D, inclusive designers, publicitários e artistas, precisam dominar as bibliotecas de baixo nível.
d) Bibliotecas de baixo nível podem ser desenvolvidas em linguagens de alto nível como PHP e JavaScript, pois não interagem com o hardware.
e) Bibliotecas de alto nível são assim denominadas porque são capazes de renderizar animações complexas como os filmes de aimação.

2. A API Python do Blender permite que seja aplicado a cada objeto da cena um conjunto de características relativas ao tipo de reflexão de cores do objeto. São processadas as componentes especular, difusa e ambiente do modelo de iluminação de Phong.

Assinale a alternativa correta com relação à atribuição de características de iluminação ao objeto.

a) Todas as componentes de reflexão, difusa, especular e ambiente, são características exclusivas de cada objeto.
b) A componente de reflexão ambiente é específica de cada objeto, mas todos os objetos compartilham a reflexão especular, que é definida para a cena 3D.
c) Para se acrescentar ao objeto informações sobre os tipos de reflexão é preciso criar um material e vinculá-lo ao objeto.
d) A função *bpy.ops.object.shade_smooth()* é o que atribui ao objeto o modelo de iluminação de Phong, com as três componentes.
e) As componentes de reflexão especular, difusa e ambiente, são atributos da cena e devem ser definidos para o objeto *bpy.context.scene*.

3. A API Python do Blender é uma biblioteca de alto nível que permite que os usuários do aplicativo desenvolvam animações com o uso do potencial de uma linguagem de programação. A API implementa modelos de iluminação, com limitações. Em algumas situações o Blender serve apenas como ferramenta de criação de protótipos.

Com relação à API Python do Blender, assinale a alternativa correta:

a) A API está sempre atualizada com as implementações dos algoritmos do estado da arte em computação gráfica, mas é limitada porque não implementa o uso de GPUs.
b) A API implementa apenas processamento de cenas estáticas, não permite a criação de jogos ou outro tipo de animação gráfica.
c) A API implementa o modelo aproximativo de iluminação de Phong, com componentes de reflexão especular, difusa e ambiente.
d) A API apresenta tanto o modelo de iluminação constante quanto o modelo de iluminação por interpolação, mas seus algoritmos de interpolação são precários.
e) A API só permite a criação de modelos com uma única câmera e uma única fonte de luz, além de não suportar objetos com transparência.

Referências

ANGEL, E.; SHREINER, D.; **Interactive Computer Graphics:** a top-down approach with shader-based OpenGL. 6. ed. Pearson, 2012.

BART. **19 Billion Triangles Render**. 2012. Disponível em: https://goo.gl/X13tG7. Acesso em: 10 dez. 2018.

BLENDER. **Blender:** 3D contente creation noob to pro. Wikibooks, jul. 2016. Disponível em: https://goo.gl/TgiYnu. Acesso em: 19 fev. 2019.

BLINN, J. Models of light reflection for computer synthesized pictures. **SIGGRAPH Comput. Graph.**, v. 11, n. 2, p. 192-198, jul. 1977.

CGTRADER. **3D models for VR / AR, 3D printing and computer graphics**. Disponível em: http://cgtrader.com. Acesso em: 10 dez. 2018.

GOURAUD, H. Continuous shading of curved surfaces. **IEEE Transaction on Computers**, v. 20, n. 6, p. 623-629, jun. 1971.

HEARN, D. D.; BAKER, M. P.; CARITHERS, W.; **Computer Graphics with OpenGL**. 4. Ed. Pearson, 2010.

HUGHES, J. F.; VAN DAM, A.; MCGUIRE, M.; SKLAR, D. F.; FOLEY, J. D.; FEINER, S. K.; AKELEY, K. **Computer graphics:** principles and practice. 3. ed. Addison-Wesley, 2013.

MÖLLER, T.; TUMBORE, B. **Fast, Minimum Storage Ray-Triangle Intersection**. Journal of Graphics Tools, n. 2, p. 21–28, 1997.

PHONG, B. Illumination for Computer Generated Pictures. **Communications of the ACM**, n. 18, p. 311-317, jun. 1975.

STENNING, J. **Direct3D Rendering Cookbook**. Packt Publishing, 2014.

The CGAL Project. **CGAL User and Reference Manual**. 4.13. ed. CGAL Editorial Board, 2018.

WU, S. **Síntese de imagens:** uma introdução ao mundo de desenho e pintura dos sistemas digitais. Disponível em: https://bit.ly/2PW2QEb. Acesso em: 15 dez. 2018. Notas de aula, 2009.

Unidade 4

Processamento digital de imagens

Convite ao estudo

Caro aluno, em unidades anteriores, você foi apresentado a conceitos básicos comuns a todas as subáreas da computação gráfica e também à síntese de imagens, adquirindo a capacidade de aplicar transformações geométricas e criar animações 3D. Em se tratando do amplo contexto de computação gráfica e processamento de imagens, você ainda precisa explorar a visão computacional e o processamento de imagens com mais detalhes.

Lembre-se de que o objetivo do processamento de imagens é a aplicação de filtros sobre imagens já em formato digital para a obtenção de elementos visuais diferentes dos elementos presentes na imagem capturada. A visão computacional visa extrair informações de imagens capturadas do mundo real. A extração de informações não é trivial sobre as imagens capturadas, por isso a visão computacional precisa abrir mão de filtros de processamentos de imagens. Há uma grande interseção entre as duas subáreas, sendo que o que diferencia as duas são as características dos filtros aplicados e o produto esperado com a aplicação de cada filtro.

Com o conteúdo desta unidade, você saberá realizar a segmentação de imagens usando *watershed* e filtros no domínio espacial e da frequência, ferramentas comuns ao processamento de imagens e à visão computacional.

Você tem experiência em computação gráfica e trabalhou com síntese de imagens e animações 3D, mas tem despertado o interesse em outros tipos de problemas relacionados à computação gráfica e ao processamento de imagens. Seu interesse agora é a análise de imagens capturadas do mundo real para extrair informações delas. Você buscou oportunidades no mercado e acaba de ser contratado por uma empresa de visão computacional.

A empresa que você acaba de se integrar realiza diversos projetos de análise de imagens e foi contratada por uma grande produtora e exportadora de cafés especiais. O projeto em que você foi alocado visa avaliar se existem correlações entre a qualidade de bebida de um lote de grãos de café e o que pode ser obtido como características de imagens capturadas das sementes de café do mesmo lote. O objetivo do seu cliente é substituir a análise sensorial da bebida, até então realizada por profissionais altamente especializados, ainda assim, com um certo grau de subjetividade, por uma análise perfeitamente objetiva baseada em imagens.

O cliente separou amostras de um lote de café e enviou para duas equipes: uma equipe interna, que realiza análise sensorial, e sua empresa, que trabalha com análise de imagens. Sua empresa é responsável por levantar características de imagens capturadas por raio X das sementes, e dividiu o trabalho em três etapas: a primeira etapa é a preparação da amostra e captura das imagens de raio X; a segunda etapa é a segmentação dos grãos na imagem; e a terceira etapa é a medição de características. Você é responsável pela segunda etapa.

A segmentação de imagens é uma tarefa que nem sempre é trivial. Muitos pré-processamentos típicos de processamento de imagens são necessários para que seja possível realizar uma boa segmentação. Você deverá avaliar, para a aplicação em específico, os filtros de processamento de imagens necessários para que a segmentação seja bem-sucedida e você possa entregar o produto requerido para a terceira etapa do trabalho.

Primeiramente, você deverá explorar a aplicação de filtros de processamento de imagens no domínio espacial; em seguida, avaliar os filtros no domínio da frequência; e, finalmente, realizar a segmentação dos grãos. Essas atividades serão tratadas nas três seções desta unidade.

Seção 4.1

CGPI: filtros de imagens digitais

Diálogo aberto

Caro aluno, nesta seção, você deverá relembrar seus conhecimentos sobre os tipos de imagens, vetoriais e matriciais. Na Unidade 3, você trabalhou com modelos 3D, uma extensão das imagens vetoriais, e com a síntese de imagens matriciais a partir dos modelos. A partir de agora trabalharemos apenas com o processamento de imagens matriciais (digitais).

Considere o contexto em que você está em uma empresa de soluções de visão computacional e precisa segmentar imagens de raio-X de sementes de café. Para o bom funcionamento dos algoritmos de segmentação as imagens precisam ser pré-processadas com filtros de processamentos de imagens. Esses filtros podem ser aplicados no domínio espacial ou no domínio da frequência. Nesta seção, você focará nos filtros no domínio espacial, que é a primeira atividade que precisa realizar.

Você deve considerar que seu objetivo final é realizar a segmentação de imagens de sementes de café capturadas por raio X, mas a aplicação de algoritmos de segmentação sobre as imagens que você recebeu não geram resultados satisfatórios. Portanto, você precisa realizar pré-processamentos, aplicando filtros que alterem as imagens de forma que os algoritmos de segmentação funcionem. Você deve aplicar diferentes filtros de suavização e de realce de bordas, evidenciando para a sua equipe o que você considera como o melhor filtro de pré-processamento no domínio espacial. Você deve entregar um relatório com a aplicação dos filtros e o código fonte que permite a criação do relatório.

Nesta seção, você aprenderá sobre os filtros de média, mediana e Sobel, utilizando ou não a operação de convolução. Ao observar os efeitos de cada filtro, poderá solucionar o problema que lhe foi conferido. Seu cliente está apostando em uma hipótese de alta correlação dos dados que você obterá com os dados de análise sensorial de café, o que lhe renderia redução de custos. O domínio do conteúdo desta seção é o primeiro passo para o seu sucesso.

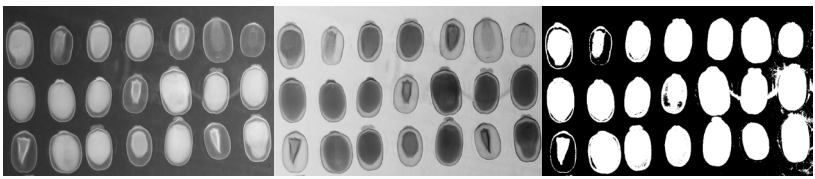
Não pode faltar

Caro aluno, esta unidade é uma introdução à subárea de processamento de imagens. Como as ferramentas de processamento de imagens são também utilizadas pela subárea de visão computacional, você também verá exemplos

de aplicações de visão. Processar uma imagem significa alterar os valores de seus pixels para se criar um novo, visando a uma alteração do conteúdo dessa imagem. O objetivo pode ser apenas a obtenção de um efeito visual, direcionado a seres humanos, ou pode ser o realce de informações para um processamento posterior por computador, em uma aplicação de visão computacional.

Os efeitos visuais de diversos filtros podem ser explorados em diversos aplicativos de edição de imagens, hoje presentes até mesmo em dispositivos móveis. A Figura 4.1 mostra o efeito da aplicação de dois filtros sobre uma imagem em níveis de cinza (a): um filtro de inversão (b) e um filtro de limiar (*threshold*) (c).

Figura 4.1 | Processamento ponto a ponto: (a) imagem original, (b) negativo e (c) limiar



(a) (b) (c)

Fonte: elaborada pelo autor.

As imagens da Figura 4.1 estão em níveis de cinza. São imagens de única camada, em que cada pixel assume valores inteiros de 8 bits, no intervalo [0,255]. Todos os exemplos desta seção serão dados com imagens em níveis de cinza, para facilitar a compreensão dos filtros. O processamento de imagens coloridas é feito pela aplicação de filtros em níveis de cinza em canais da imagem. Uma estratégia é aplicar o filtro independentemente em cada canal RGB. Outra estratégia é a transformação da imagem para HSL, aplicação do filtro sobre o canal L e transformação inversa, aplicando o filtro sobre a luminância da imagem, apenas.

Assimile

O processamento de imagens coloridas utiliza os mesmos filtros de processamento de imagens do processamento em níveis de cinza. Os filtros são aplicados sobre os canais da imagem, podendo a imagem estar nos formatos RGB, HSL ou qualquer outro. Para facilitar a compreensão dos filtros, eles são estudados com exemplos em níveis de cinza.

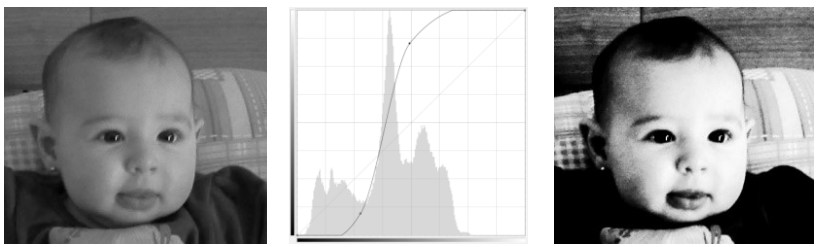
Os filtros da Figura 4.1 são **filtros espaciais** de **processamento ponto a ponto** (GONZALEZ; WOODS, 2011), ou seja, o valor de cada pixel da imagem filtrada é obtido por uma equação envolvendo apenas o pixel de mesmas coordenadas espaciais na imagem original. O filtro de inversão que

gera a Figura 4.1(b) é dado pela equação $g(x,y) = c_{max} - f(x,y)$, onde f é a imagem original, c_{max} é o máximo valor possível de f, e g é a imagem resultante. O filtro de limiar é dado pela equação $g(x,y) = c_{max}$, se $f(x,y) > l$, e $g(x,y) = 0$, senão, onde f é a imagem original, c_{max} é o máximo valor possível de f, l é o limiar e g é a imagem resultante. Esses filtros podem ser implementados em apenas uma linha em Python e NumPy.

```
def inverte(f):

    return 255*numpy.ones(f.shape,dtype=numpy.
    uint8) - f


def limiar(f,th):

    return 255*((f>th).astype(numpy.uint8))
```

Outro filtro de processamento ponto a ponto que podemos citar é o filtro de **curva de cores**. A Figura 4.2 mostra a aplicação de um filtro de curva de cores e a respectiva curva.

Figura 4.2 | Curva de cores: (a) imagem original, (b) curva de cores e (c) aplicação da curva de cores sobre a imagem original



(a) (b) (c)

Fonte: elaborada pelo autor.

A imagem na Figura 4.2(b) mostra, em cinza intermediário, o **histograma** da imagem da Figura 4.2(a), em cinza claro, a curva de cores original (reta com inclinação de 45 graus), e em preto, a curva de cores aplicada sobre a Figura 4.2(a) para obter a Figura 4.2(c). O eixo x corresponde aos níveis de cinza da imagem original. O histograma é uma distribuição de frequências dos níveis de cinza. Para cada nível de cinza da imagem original, o histograma mostra o número de pixels da imagem que estão com aquele valor. Nota-se um pico próximo ao centro do eixo x, o que significa que há uma grande quantidade de pixels nesta faixa de valores.

O eixo y corresponde aos níveis de cinza da imagem filtrada. A curva de cores é, portanto, um mapeamento de cores da imagem original para cores da imagem processada. Na curva da Figura 4.2(b), nota-se que uma estreita faixa de valores em x, em torno do pico do histograma, é mapeada para uma larga faixa de valores de y. Nesta faixa de valores, está sendo aplicado um alargamento do **contraste**, para que os valores nesta faixa se tornem mais diferenciados. Nas demais faixas, está sendo feita uma redução do contraste. Note que a blusa do bebê, de cores escuras, na Figura 4.2(a), aparece quase que de cor uniforme na Figura 4.2(c).

```
def curvaDeCores(f,curva):
    (ys,xs) = f.shape
    g = numpy.zeros(f.shape, dtype=numpy.uint8)
    for y in range(ys):
        for x in range(xs):
            g[y,x] = curva[f[y,x]]
    return g
```

Reflita

Você entende que o filtro de curva de cores é uma generalização dos filtros de inversão e limiar? É possível criar os filtros de inversão e limiar passando para a função curva de cores uma curva específica para cada caso? Como seriam essas curvas?

Convolução e correlação

Uma grande variedade de aplicações exige filtros espaciais que calculam o pixel de saída não somente com base no pixel de entrada, mas com base em uma vizinhança desse pixel (DAVIES, 2012). A Figura 4.3 mostra o exemplo numéricos da aplicação de um desses filtros: o filtro de média.

Figura 4.3 | Exemplo numérico de filtro espacial por convolução: (a) imagem original; (b) filtro de média; (c) aplicação do filtro de média sobre (a)

	0	1	2	3	4	5	6
0	18	13	10	9	10	13	18
1	13	**8**	**5**	**4**	5	8	13
2	10	**5**	**2**	**1**	2	5	10
3	9	**4**	**1**	**0**	1	4	9
4	10	5	2	1	2	5	10
5	13	8	5	4	5	8	13
6	18	13	10	9	10	13	18

(a)

$$\frac{1}{9} \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

(b)

	0	1	2	3	4	5	6
0	-	-	-	-	-	-	-
1	-	9	6	5	6	9	-
2	-	6	**3**	2	3	6	-
3	-	5	2	1	2	5	-
4	-	6	3	2	3	6	-
5	-	9	6	5	6	9	-
6	-	-	-	-	-	-	-

(c)

Fonte: elaborada pelo autor.

Na Figura 4.3, em (a), foram destacados os pixels da janela 3×3 com centro na coordenada (2,2). O filtro em (b) é sobreposto a esta janela e é feita a multiplicação pixel a pixel dos elementos do filtro pelos elementos da imagem para se obter o valor do pixel (2,2). No exemplo, temos $\frac{8*1+5*1+4*1+5*1+2*1+1*1+4*1+1*1+0*1}{9} = 3{,}33$, que é a média dos pontos da janela. Por arredondamento, o valor do pixel (2,2) da imagem (c) é 3.

A operação de somatória das multiplicações ponto a ponto de uma janela é matematicamente definida pelas operações de **convolução** e **correlação** (KREYSZIG, 2011). Para compreender as equações, vamos considerar f e g em 1D. A correlação de f por g é dada por $(f \circ g)(x) = h(x) = \int_{-\infty}^{\infty} f(u) \cdot g(x+u)du$, e a convolução de f por g é dada por $(f * g)(x) = h(x) = \int_{-\infty}^{\infty} f(u) \cdot g(x-u)du$. Veja que, para cada valor de x da função correlação, é calculada a integral da multiplicação de todos os pontos de f por todos os pontos de g, sendo o centro de g deslocado até o ponto x (x+u). Para cada valor de x da função convolução, é calculada a integral da multiplicação de todos os pontos de f por todos os pontos de g, sendo g refletida com base em seu centro, e seu centro deslocado até o ponto x (x-u).

A convolução e a correlação discretas em 1D são definidas, respectivamente, pelas equações $(f * g)[x] = h[x] = \sum_u f[u] \cdot g[x-u]$ e $(f \circ g)[x] = h[x] = \sum_u f[u] \cdot g[x+u]$, que podem ser estendidas para 2D como $(f * g)[x,y] = h[x,y] = \sum_v \sum_u f[u,v] \cdot g[x-u, y-v]$ e $(f \circ g)[x,y] = h[x,y] = \sum_v \sum_u f[u,v] \cdot g[x+u, y+v]$. Na convolução 2D, portanto, o filtro g é refletido horizontal (x-u) e verticalmente (y-v).

A equação da correlação 2D é exatamente a operação ilustrada na Figura 4.3. Como o filtro de média da Figura 4.3(b) é simétrico horizontal e verticalmente, a convolução é igual à correlação. No entanto, se o filtro não for simétrico, as duas operações darão resultados diferentes.

Exemplificando

Vamos considerar um filtro representado pela matriz $S = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$. Considere f a imagem da Figura 4.3(a). A correlação $f \circ S$ no ponto (2,2) é a somatória das multiplicações ponto a ponto de S e f na janela, ou seja, 8*1 + 5*2 + 4*1 + 5*0 + 2*0 + 1*0 + 4*(-1) + 1*(-2) + 0*(-1) = 16. Já a convolução <<Eqn014.wmf>> no mesmo ponto é a somatória das multiplicações ponto a ponto de S_r e f na janela, onde $S_r = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$ é S refletido horizontal e verticalmente. No caso, temos 8*(-1) + 5*(-2) + 4*(-1) + 5*0 + 2*0 + 1*0 + 4*1 + 1*2 + 0*1 = -16.

A diferença entre a convolução e a correlação é apenas um sinal, mas as aplicações podem diferir. A correlação é mais usada na estatística (BUSSAB; MORETTIN, 2017), para busca de padrões (filtro g) em uma determinada função (f). Em processamento de imagens, a convolução é mais usada devido a uma propriedade que será apresentada na Seção 4.2. As convoluções são a base para as redes neurais convolucionais, utilizadas para aprendizado de máquina, no contexto do aprendizado profundo (*deep learning*) (PONTI; COSTA, 2017).

Vamos nos concentrar, portanto, na convolução discreta em duas dimensões. O código a seguir implementa a função *conv2d*. Considere *import numpy as np*.

```
def conv2d(f,g): # g em dimensões ímpares
    gr = np.fliplr(np.flipud(g)) # simetria
    (yf,xf)=f.shape;   (yg,xg)=g.shape
    hshape = (yf-(yg-1),xf-(xg-1)) # sem bordas
    h = np.zeros(hshape,dtype=np.int32) # permite
    números negativos
    for y in range(yf-(yg-1)):
        for x in range(xf-(xg-1)):
            j = f[y:y+yg,x:x+xg] # janela de f nas
            dimensões de g
            h[y,x] = sum(sum(gr*j))
    return h
```

O filtro de média pode ser aplicado por convolução com o código a seguir, considerando o tamanho s (ímpar) do filtro como parâmetro.

```
def filtroMedia(f,s): # s deve ser ímpar
    return conv2d(f,np.ones((s,s))/(s*s))


h = filtroMedia(f,s).astype(np.uint8)
```

Filtros por convolução

Qualquer matriz é um filtro se for feita sua convolução com a imagem. No entanto, atribuir valores arbitrários aos elementos do filtro levará a resultados também arbitrários. A literatura apresenta diversos filtros criados com embasamento matemático e que geram resultados úteis, como é o caso do filtro de média, suavizante. Vejamos alguns desses filtros a seguir.

O filtro **laplaciano** é definido como $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$. Trata-se de um filtro que mede a diferença de um pixel com relação à média de seus vizinhos, produzido por uma equação de segunda derivada nos eixos x e y. Este filtro detecta as regiões de borda, como mostra a Figura 4.4. Como há valores negativos no resultado da convolução, a Figura 4.4(b) mostra o resultado normalizado no intervalo [0,255], sendo que os pixels em cinza médio são os pixels de regiões homogêneas; os pixels mais escuros são os pixels que possuem valor mais baixo que a média da sua vizinhança; e os pixels mais claros são os que possuem valor mais alto que a vizinhança.

A Figura 4.4(c) mostra a negação do módulo do resultado, que permite melhor visualização, sendo que os pixels pretos são aqueles com valores mais diferentes (mais altos ou mais baixos) da média da vizinhança.

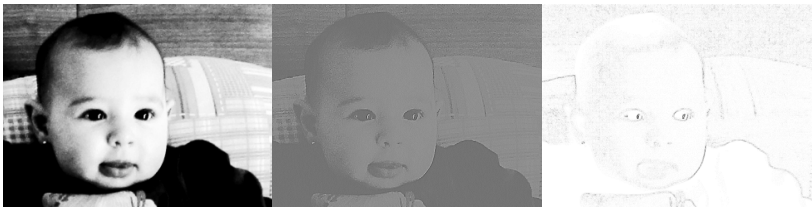
Figura 4.4 | Filtro laplaciano: (a) imagem original; (b) resultado normalizado da aplicação do laplaciano; e (c) negação do módulo do resultado da aplicação do laplaciano

(a) (b) (c)

Fonte: elaborada pelo autor.

O filtro de **aguçamento** é também um filtro que realça pixels em regiões de bordas. É definido por $\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$. É muito similar ao laplaciano, como pode ser visto na Figura 4.5.

Figura 4.5 | Filtro de aguçamento: (a) imagem original; (b) resultado normalizado da aplicação do filtro; e (c) negação do módulo da diferença (a)-(b)

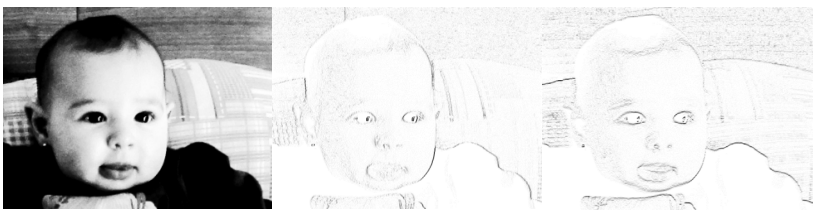


(a) (b) (c)

Fonte: elaborada pelo autor.

O filtro de **Sobel** é um dos mais conhecidos filtros de realce de borda. Foi construído com base no cálculo da primeira derivada da imagem. São filtros de Sobel qualquer rotação do filtro $UD = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$, por exemplo, os filtros $LR = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$ e $D = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -2 \end{bmatrix}$. O filtro UD realça bordas horizontais; o LR, verticais; e o D, diagonais. A Figura 4.6 mostra a aplicação dos filtros UD e LR.

Figura 4.6 | Filtro de Sobel: (a) imagem original; (b) aplicação do filtro LR de Sobel; e (c) aplicação do filtro UD de Sobel

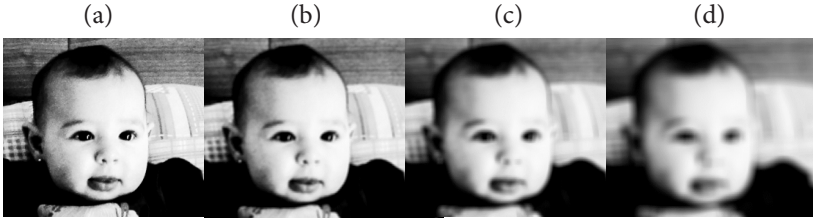


(a) (b) (c)

Fonte: elaborada pelo autor.

Além de filtros que realçam as bordas, há também os filtros que as suavizam (*blur*). É o caso do filtro de média ilustrado numericamente na Figura 4.3. Quanto maior o filtro, maior é a suavização. A Figura 4.7 mostra a aplicação do filtro de média de três tamanhos.

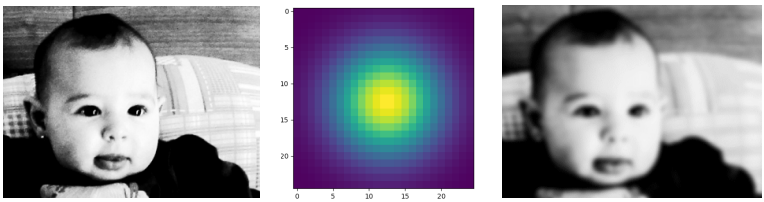
Figura 4.7 | Filtro de média: (a) imagem original; (b) filtro 7×7; (c) filtro 15×15; e (d) filtro 25×25



Fonte: elaborada pelo autor.

O filtro suavizante mais popular é o filtro **gaussiano**. Ao invés de se fazer a média dos pixels de uma vizinhança quadrada, faz-se a média ponderada da vizinhança com base em uma distribuição gaussiana 2D. Quanto mais próximo do centro do filtro, mais relevância tem o pixel para a média ponderada. A Figura 4.8 mostra a aplicação de um filtro gaussiano de tamanho 25×25. Compare a Figura 4.8(c) com a Figura 4.7(d) e perceba que o filtro gaussiano não gera efeitos indesejáveis nas bordas como o filtro de média.

Figura 4.8 | Filtro gaussiano: (a) imagem original; (b) aplicação do filtro LR gaussiano; e (c) aplicação do filtro UD gaussiano



Fonte: elaborada pelo autor.

Você pôde perceber que existem inúmeras possibilidades de criação de filtros por convolução. Uma forma de explorar os efeitos visuais de diversos filtros é utilizando um aplicativo de edição de imagens como o Gimp (2019). Ao escolher um filtro, você pode pesquisar sua definição matemática e implementá-lo de forma simples em seu próprio programa, com o auxílio de uma implementação existente da convolução. Bom trabalho!

Sem medo de errar

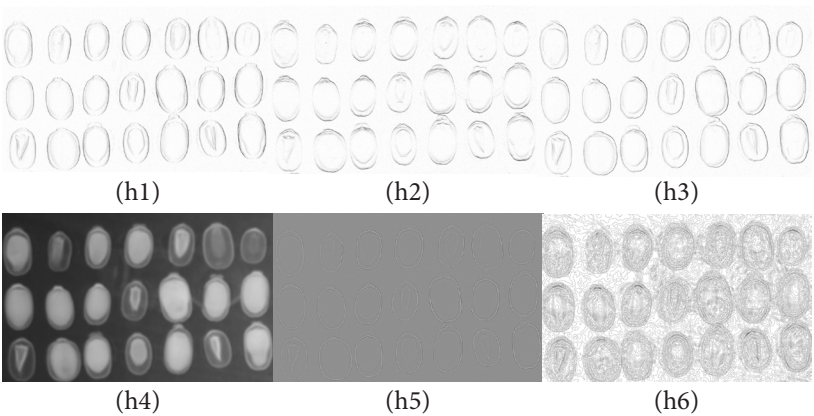
A atividade que lhe foi delegada é a criação de um relatório contendo diversas aplicações de filtros sobre uma imagem de raio X de sementes de café. O objetivo final é segmentar as sementes, mas será preciso fazer pré-processamento, de forma que os algoritmos de segmentação funcionem. Você deve aplicar diferentes filtros de suavização e de realce de bordas, evidenciando para a sua equipe o que você considera como o melhor filtro de pré-processamento no domínio espacial. Você deve entregar um relatório com a aplicação dos filtros e o código fonte que permite a criação do relatório.

A imagem de raio X de sementes de café foi apresentada na Figura 4.1(a). Não foi esta a imagem utilizada ao decorrer da seção. Em seu lugar, foi utilizada a imagem do bebê. O que você deve fazer é o mesmo que foi feito com a imagem do bebê: aplicar diversos filtros e avaliar visualmente o que faz o melhor.

Veja um exemplo de código para gerar um relatório como o esperado. Os resultados dos filtros aplicados são apresentados na Figura 4.9.

```
f = np.asarray(Image.open('semente.png'))

h1 = inverte(normaliza(abs(filtroSobelLR(f))))
h2 = inverte(normaliza(abs(filtroSobelUD(f))))
h3 = inverte(normaliza(abs(filtroSobelLR(f)+abs(filtro-
SobelUD(f)))))
h4 = filtroGaussiano(f,15).astype(np.uint8)
h5 = normaliza(filtroLaplaciano(f))
h6 = inverte(normaliza(abs(filtroLaplaciano(h4))))
```

Figura 4.9 | Aplicação de diversos filtros sobre a imagem de semente



Fonte: elaborada pelo autor.

Foram aplicados o filtro de Sobel de realce de bordas verticais (h1), horizontais (h2) e ambos combinados (h3). Também, os filtros gaussiano (h4), laplaciano (h5) e o laplaciano da imagem suavizada (h6).

Nos experimentos da Figura 4.9, o melhor resultado, aparentemente, é h3, que tem todos os contornos das sementes melhor definidos. No entanto, para resolver a contento a situação-problema, você deve explorar outras combinações para chegar a uma proposta mais segura. Assim, você pode apresentar as imagens obtidas, como foi apresentado na Figura 4.9, e demonstrará que sabe aplicar filtros no domínio espacial.

Avançando na prática

Limiar automático

Descrição da situação-problema

Sua equipe está tendo problemas com a definição arbitrária de valores de limiar para uma aplicação que envolve a operação de limiar sobre suas imagens. Você foi designado para desenvolver um primeiro operador de limiar automático, baseado em histograma. Para definir qual é o valor do limiar, você deverá encontrar o valor de limiar que o coloque acima dos valores da metade dos pixels e abaixo dos valores da outra metade, ou o máximo que você puder se aproximar disso. Você deve apresentar o código fonte desse operador.

Resolução da situação-problema

Primeiramente, é preciso construir um histograma. Como já informado, o histograma contém, para cada nível de cinza, o número de pixels com aquele nível de cinza. Para imagens de 8 bits, basta criar uma lista h de 256 posições e percorrer todos os pixels da imagem. Se o nível de cinza do pixel visitado é g, incrementar h[g], como no código a seguir.

```
def histograma(f):
    h = [0]*256
    (ys,xs) = f.shape
    for y in range(ys):
        for x in range(xs):
            h[f[y,x]]+=1
    return h
```

Depois, basta encontrar o centro dessa distribuição. A somatória dos valores do histograma é igual ao número total de pixels da imagem. O centro c é a metade disso. Basta, então, percorrer o histograma a partir do nível de cinza g=0 e acumular o número de pixels à medida que se incrementa g. Quando o acumulado ultrapassa o valor c, foi encontrado o centro, conforme código a seguir.

```
def encontraLimiar(h):
    meio = sum(h)/2
    acumulado = 0
    for lim in range(len(h)):
        acumulado += h[lim]
        if acumulado > meio:
            break
    return lim
```

Finalmente, seja f a imagem a ser processada, chama-se *limiar(f,encontraLimiar(histograma(f)))*, utilizando a função limiar apresentada ao longo da seção.

Faça valer a pena

1. Os filtros de processamento de imagens no domínio espacial podem ser filtros de processamento ponto a ponto ou filtros de processamento de vizinhança. Dentre os filtros de processamento de vizinhança, os filtros por convolução são os mais usados.

Sobre os filtros de processamento e filtros por convolução, assinale a afirmativa correta.

a) Nos filtros por convolução, o valor de um pixel da imagem filtrada é obtido pelo cálculo da média dos valores dos pixels na vizinhança do pixel de mesmas coordenadas na imagem de entrada.
b) Nos filtros de processamento ponto a ponto, o valor de um pixel da imagem filtrada é obtido pelo cálculo da média dos valores dos pixels na vizinhança do pixel de mesmas coordenadas na imagem de entrada.
c) Nos filtros de processamento ponto a ponto, o valor de um pixel da imagem filtrada é calculado com base apenas no valor do pixel de mesmas coordenadas na imagem de entrada.
d) Nos filtros por convolução, o valor de um pixel da imagem filtrada é calculado com base apenas no valor do pixel de mesmas coordenadas na imagem de entrada.
e) Nos filtros por convolução, o valor de um pixel da imagem filtrada é obtido pelo cálculo do produto vetorial da matriz do filtro pela matriz da vizinhança do pixel de mesmas coordenadas na imagem de entrada.

2. Observe as três imagens a seguir:

(a) (b) (c)

Fonte: elaborada pelo autor.

A imagem (b) é o negativo da imagem (a), e a imagem (c) é resultado da aplicação de um limiar sobre a imagem (a). Os filtros negativo e limiar são largamente utilizados em processamento de imagens e visão computacional.

Sobre os filtros das imagens que constam no texto-base, assinale a alternativa correta.

a) O negativo é um filtro de processamento de vizinhança, em que o pixel da imagem filtrada é obtido pela subtração do pixel de mesma coordenada na imagem de entrada pela média de seus vizinhos.

b) Tanto o negativo quanto o limiar podem ser obtidos em O(n), onde n é o número de pixels da imagem, pois são filtros de processamento ponto a ponto.

c) O negativo pode ser obtido em O(n), pois trata-se apenas da inversão dos níveis de cinza da entrada, mas o limiar tem custo computacional mais alto devido ao cálculo do limiar ótimo com base no histograma.

d) O negativo é um caso especial da aplicação de curvas de cores, enquanto o limiar é um filtro específico que só pode ser obtido por convolução.

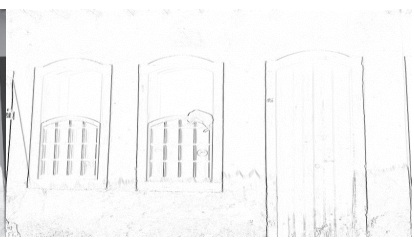
e) Tanto o negativo quanto o limiar são casos especiais da aplicação de curvas de cores, que é um filtro por convolução, de complexidade $O(n^2)$.

3. A seguir, são apresentados filtros de convolução conhecidos na área de processamento de imagens digitais.

$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 2 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -2 \end{bmatrix}$
1	2	3	4	5

Considere as imagens a seguir:

(a)



(b)

Fonte: elaboradas pelo autor.

A imagem (b) é o valor ab soluto normalizado e invertido do resultado da aplicação de um dos cinco filtros por convolução acima sobre a imagem (a).

Qual filtro foi aplicado sobre a imagem (a) para obter a imagem (b)? Assinale a alternativa correta.

a) 1
b) 2
c) 3
d) 4
e) 5

Seção 4.2

CGPI: filtros no domínio da frequência

Diálogo aberto

Caro aluno, na seção anterior, você foi apresentado aos filtros de imagens no domínio espacial. Os filtros por convolução fazem parte desse conjunto de filtros. Se você chegou a implementar e testar os códigos de filtros por convolução em Python dados como exemplo, deve ter percebido um tempo relevante de processamento. Há muitas formas de melhorar a performance dos filtros de convolução e, nesta seção, você conhecerá a principal: o teorema da convolução.

Com o teorema da convolução, você pode aplicar a uma imagem a Transformada de Fourier, que leva a imagem do domínio espacial para o domínio da frequência. Os filtros por convolução no espaço podem ser implementados de forma muito mais eficiente no domínio da frequência, como será apresentado neste material.

Vamos retomar o contexto em que você se encontra. O projeto em que você foi alocado visa avaliar se existem correlações entre a qualidade de bebida de um lote de grãos de café e o que pode ser obtido como características de imagens capturadas das sementes de café do mesmo lote. Você é responsável por apenas uma etapa desse projeto e deve segmentar grãos de café, mas, para que os algoritmos de segmentação funcionem, deve preparar a imagem aplicando filtros de pré-processamento. Você já avaliou os filtros no domínio espacial e, agora, deve avaliar os filtros no domínio da frequência.

Em sua avaliação anterior, você apresentou um relatório e o código fonte que aplica diferentes filtros de suavização e de realce de bordas no domínio espacial, evidenciando para a sua equipe o que considera como o melhor. Sabe-se, porém, que os filtros no domínio espacial são computacionalmente muito custosos, e que a utilização de filtros no domínio da frequência é mais eficiente. Seu trabalho, agora, é demonstrar a aplicação, no domínio da frequência, dos mesmos filtros até então aplicados no domínio espacial. Você deve entregar um relatório mostrando a diferença da aplicação dos filtros no domínio da frequência com relação à aplicação no domínio espacial, com o código fonte que permite a criação do relatório.

Com o conteúdo desta seção, você conhecerá o potencial da Transformada de Fourier e de outras transformadas de imagens. Assim, terá um novo leque de opções de processamento de imagens para utilizar em problemas reais.

Não pode faltar

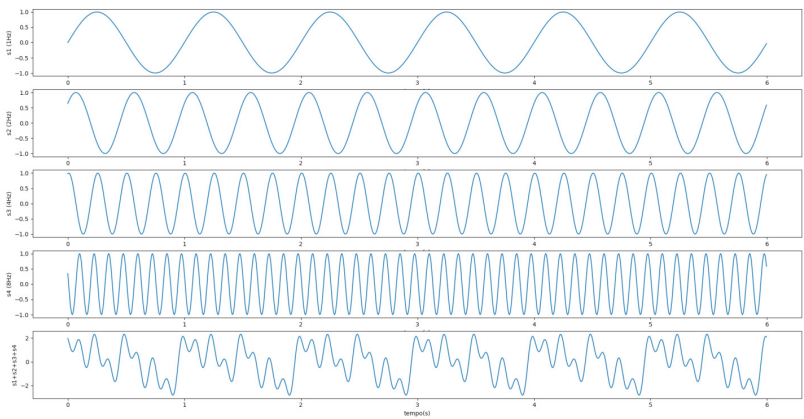
Caro aluno, na seção anterior, você aprendeu como são construídos diversos filtros de processamento de imagens. Além dos filtros ponto a ponto, conheceu a correlação e a convolução e foi apresentado aos filtros por convolução. Se você chegou a implementar e testar os códigos que lhe foram apresentados, deve ter percebido que o cálculo da convolução requer um tempo considerável de processamento.

É claro que há formas de melhorar a performance dos códigos já apresentados, pois foram expostos da forma mais didática possível, para que você tenha facilidades de ler e compreender o algoritmo. Mas, o que mais é capaz de reduzir o tempo de cômputo da convolução é o que você verá nesta seção: a Transformada de Fourier e o Teorema da Convolução.

No início do século XIX, o matemático francês Joseph Fourier afirmou em um trabalho científico que qualquer sinal periódico pode ser expresso como uma combinação ponderada de funções seno e cosseno com diferentes períodos ou frequências (SOLOMON, 2013).

A ideia de Fourier é ilustrada na Figura 4.10, a qual apresenta quatro sinais senoidais (s1, s2, s3 e s4) de mesma amplitude (1) e diferentes frequências e fases. A somatória desses quatro sinais (s1+s2+s3+s4) também é um sinal periódico, o que pode ser visto na imagem.

Figura 4.10 | Ilustração da série de Fourier: somatória de quatro senoides



Fonte:

Não importa o nível de complexidade da função, se ela for periódica, pode ser expressa como uma somatória de funções seno e cosseno de diversas frequências e amplitudes. A essa somatória dá-se o nome de **série de Fourier** (KREYSZIG, 2011).

Reflita

Na Figura 4.10, é possível ver a periodicidade do sinal combinado porque as frequências de s1 a s4 são valores inteiros múltiplos uns dos outros. O sinal combinado ficou com a mesma frequência de s1 (1Hz), que é a menor frequência. Mas, e se as frequências não fossem valores inteiros múltiplos, a frequência do sinal combinado seria maior ou menor?

Transformada de Fourier

Sabendo que uma função periódica qualquer, no domínio espacial, pode ser decomposta em uma combinação de senos e cossenos, é preciso definir, agora, uma forma de calcular essa decomposição. A **Transformada de Fourier** é a operação que realiza esse cálculo. A transformada de Fourier contínua unidimensional é definida por $\Im\{f(x)\} = F(u) = \int_{-\infty}^{+\infty} f(x) \cdot e^{-j2\pi ux} dx$ (GONZALEZ; WOODS, 2011), em que j denota a componente imaginária do conjunto dos **números complexos**. Pela fórmula de Euler (SWOKOWSKI, 1995), $e^{jx} = \cos x + j \sin x$, então $\Im\{f(x)\} = F(u) = \int_{-\infty}^{+\infty} f(x) \cdot (\cos(2\pi ux) - j \sin(2\pi ux)) dx$.

A integral da transformada de Fourier elimina a variável espacial x, e a função resultante é uma função F(u). A variável u aparecerá sempre como argumento das funções seno e cosseno, portanto F(u) é da forma $A(\cos(u) + j \sin(u))$ para um valor fixo de u, em que A é constante. Assim, a função F(u) representa o conjunto de senos e cossenos que, combinados, compõem a função f(x). Como u está multiplicado por 2π, u representa exatamente as frequências desses senos e cossenos. Por esse motivo dizemos que F(u) está no **domínio da frequência** (SHAPIRO; STOCKMAN, 2001).

Assimile

A transformada de Fourier transforma uma função f(x) no domínio espacial em uma função F(u), em que os valores de u representam as frequências dos senos e cossenos da série de Fourier que, quando combinados, geram a função f(x). Por isso dizemos que F(u) está no domínio da frequência.

A função f(x) pode ser recuperada a partir de F(u), sem perdas, pela **transformada inversa de Fourier**, definida por $\Im^{-1}\{F(u)\} = f(x) = \int_{-\infty}^{+\infty} F(u) \cdot e^{j2\pi ux} du$. Considerando que $f(x) = \Im^{-1}\{F(u)\}$, dizemos que $f \leftrightarrow F$ é um par de transformadas de Fourier.

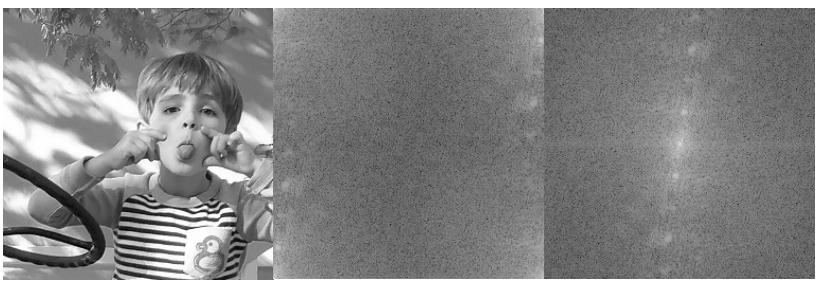
Transformada discreta de Fourier

A definição da transformada de Fourier unidimensional contínua pode ser estendida para funções discretas e bidimensionais, para se definir a **Transformada Discreta de Fourier** (DFT – *Discrete Fourier Transform*). Uma explicação sucinta de como deduzir a DFT é apresentada na Seção 5.14, da obra de Solomon (2013), e uma explicação mais detalhada pode ser encontrada nas Seções 4.3 e 4.5, da obra de Gonzalez e Woods (2011).

A DFT e sua inversa (iDFT), em uma dimensão, são dadas por $F(u)=\sum_{x=0}^{M-1} f(x)\cdot e^{\frac{-j2\pi ux}{M}}, u=0,1,2,...,M-1$ e $f(x)=\frac{1}{M}\sum_{x=0}^{M-1} F(u)\cdot e^{\frac{j2\pi ux}{M}}, x=0,1,2,...,M-1$, respectivamente. Note que os valores de u são também discretos. Em duas dimensões, a DFT e sua inversa, para uma imagem digital M×N, são definidas por $F(u,v)=\sum_{y=0}^{N-1}\sum_{x=0}^{M-1} f(x,y)\cdot e^{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}$ e $f(x,y)=\frac{1}{MN}\sum_{v=0}^{N-1}\sum_{u=0}^{M-1} F(u,v)\cdot e^{j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}$.

Para um par de DFTs $f(x,y)\leftrightarrow F(u,v)$, se f(x,y) é de dimensões M×N, F(u,v) também o é. No entanto, o valor de cada pixel (contradomínio da imagem) em F(u,v) pertence ao conjunto dos números complexos. Como não é possível exibir uma imagem de números complexos, exibe-se sua magnitude $|F(u,v)|=\sqrt{R^2(u,v)+I^2(u,v)}$, em que R e I são as componentes real e imaginária, respectivamente, a qual chamamos de **espectro de Fourier**. A Figura 4.11 mostra uma imagem (a), seu espectro de Fourier (b) e seu espectro de Fourier com o centro deslocado para o centro da imagem. As magnitudes são mostradas em escala logarítmica.

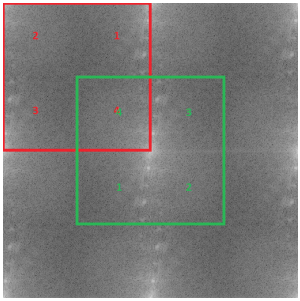
Figura 4.11 | Espectro de Fourier: (a) imagem original, (b) seu espectro de Fourier e (c) seu espectro de Fourier com centro deslocado



(a) (b) (c)

Fonte: elaborada pelo autor.

Figura 4.12 | Deslocamento do centro da DFT para melhor visualização. Em vermelho, janela com o centro da DFT em (0,0); em verde, janela com o centro da DFT em (xs/2,ys/2)



Fonte: elaborada pelo autor.

Para produzir a imagem da Figura 4.11(c), é preciso deslocar o centro da DFT da Figura 4.11(b) do ponto (0,0) para o centro da imagem em (xs/2,ys/2), em que xs e ys são as dimensões x e y da imagem, respectivamente. É preciso lembrar que a transformada de Fourier é aplicável a funções periódicas. O que se obtém na Figura 4.11(b) é apenas um período da transformada. A Figura 4.12 ilustra como é feito o deslocamento do centro.

Exemplificando

Veja, na Figura 4.12, que a janela de visualização com centro deslocado (verde) possui os quatro quadrantes da janela produzida pela DFT (vermelha) em posições deslocadas. A função *dftshift* a seguir faz esse deslocamento com base nos quadrantes.

```
def dftshift(mag): # mag é a magnitude da DFT
    sh = np.zeros(mag.shape, dtype=np.uint8) # shift
    (ys,xs) = mag.shape
    yc = int(ys/2); xc = int(xs/2)
    sh[0:yc,xc:xs] = mag[ys-yc:ys,0:xs-xc] # q1 = q3
    sh[0:yc,0:xc] = mag[ys-yc:ys,xs-xc:xs] # q2 = q4
    sh[yc:ys,0:xc] = mag[0:ys-yc,xs-xc:xs] # q3 = q1
    sh[yc:ys,xc:xs] = mag[0:ys-yc,0:xs-xc] # q4 = q2
    return sh
```

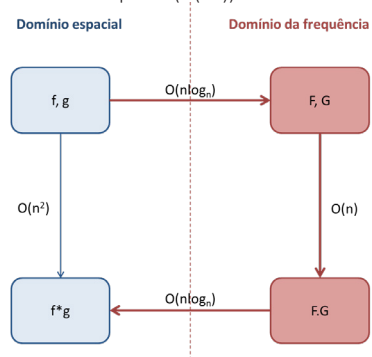
A implementação da transformada discreta de Fourier na força bruta é a iteração nos pontos de F(u,v), e para cada ponto (u,v), o cálculo da dupla somatória, conforme o código a seguir.

```
def dftForcaBruta(f):
    f = np.asarray(f).astype(np.float32)
    F = np.zeros(f.shape, dtype=np.complex64)
    (N,M) = f.shape
    for v in range(N):
        for u in range(M):
            for y in range(N):
                for x in range(M):
                    F[v,u] = f[y,x]*np.exp(-1j*2*np.
                    pi*(u*x/M + v*y/N))
    return F
```

A implementação força bruta é ineficiente, sobretudo quando se trata de implementação em linguagem de alto nível, como é o caso do Python. Considerando n=MN, o número total de pixels da imagem, o algoritmo acima tem complexidade da ordem de $O(n^2)$. Para uma fotografia de 12 Mpixels, comum em smartphones de entrada, seriam necessárias cerca de $12M^2$, ou 144 trilhões de iterações, que incluem cálculos de exponenciais, multiplicações e adições. Não seria nada prático trabalhar no domínio da frequência com implementações tão lentas.

Felizmente, uma excelente otimização foi descoberta na década de 1940: a transformada rápida de Fourier (FFT – *Fast Fourier Transform*) (DANIELSON; LANCZOS, 1942). A otimização que permitiu o algoritmo da FFT é detalhadamente explicada por Brigham (1988), mas também pode ser encontrada em Gonzalez e Woods (2011). O resultado é um algoritmo que calcula a DFT com complexidade $O(n\log_n)$. Com isso, torna-se vantajoso utilizar o teorema da convolução e filtrar as imagens no domínio da frequência, como ilustra a Figura 4.13.

Figura 4.13 | Filtragem no domínio da frequência. Transformar a imagem e o filtro para o domínio da frequência, multiplicar e transformar o resultado de volta é mais rápido (O(n-logn)) do que aplicar o filtro por convolução no domínio espacial (O(n2))

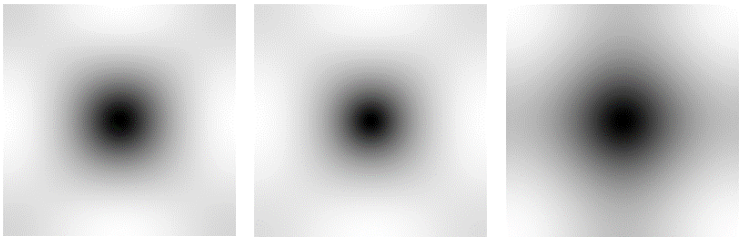


Fonte: elaborada pelo autor.

Filtros no domínio da frequência

Com o teorema da convolução, qualquer filtro por convolução, conhecido no domínio espacial, pode ser aplicado no domínio da frequência. A Figura 4.14 mostra os filtros de aguçamento e laplaciano no domínio da frequência. As frequências mais baixas estão próximas ao centro das imagens. Esses filtros são passa-alta, porque atenuam (multiplicam por valores baixos, representados em tons de cinza mais escuros) as frequências baixas e destacam (multiplicam por valores altos, representados em tons de cinza mais claros) as altas frequências.

Figura 4.14 | Filtros de aguçamento (a) e laplaciano (b)

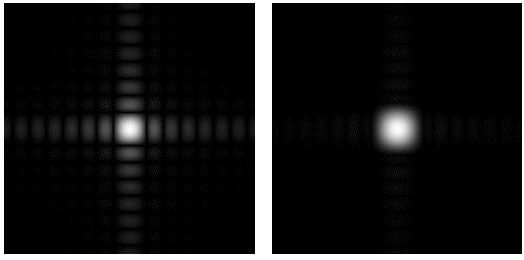


$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \qquad \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix} \qquad \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Fonte: elaborada pelo autor.

A Figura 4.15 mostra os filtros de média (a) e gaussiano (b). Ambos são de tamanho 15×15 no domínio espacial e são filtros passa-baixa, que atenuam as frequências altas e destacam as frequências baixas, mas o filtro de média mantém ainda muitos componentes de alta frequência, o que não acontece com o filtro gaussiano, o que explica a diferença na suavização gerada por esses dois filtros, como mostrado na seção anterior.

Figura 4.15 | Filtro de média (a) e filtro gaussiano de tamanho 15×15

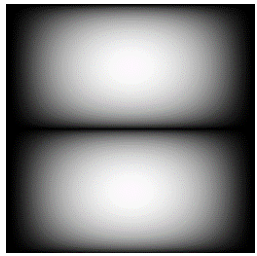


(a) (b)

Fonte: elaborada pelo autor.

Finalmente, a Figura 4.16 mostra dois filtros de Sobel, de realce de bordas. Ambos são filtros passa-alta, mas com realces diferentes na horizontal e na vertical.

Figura 4.16 | Filtros de Sobel de realce de bordas horizontais (a) e verticais (b)



$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \qquad \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Fonte: elaborada pelo autor.

Você poderia criar um filtro diretamente no domínio da frequência, mas, neste caso, você deve estar atento a uma propriedade da transformada de Fourier: a simetria conjugada. Essa propriedade diz que a transformada de Fourier de uma função real bidimensional é simétrica com relação à sua diagonal, ou seja, F(u,v) = F(-u,-v). Se, ao aplicar o filtro G no domínio da frequência ($F \cdot G$), esta propriedade for preservada, garante-se que a iDFT de $F \cdot G$ ($f * g$) será uma função real. Se esta propriedade não for preservada, $f * g$ será uma função complexa. Ao criar um filtro diretamente no domínio da frequência, é importante observar essa propriedade.

Outros exemplos de uso do domínio da frequência

Existem outras transformadas que levam as imagens do domínio espacial para o domínio da frequência. A **transformada cosseno discreta** (DCT, do inglês *Discrete Cosine Transform*) é um exemplo. A DCT é uma transformada similar à DFT, utilizada, principalmente, para a compressão de imagens (formato JPG) e vídeo (formato MPEG).

Pesquise mais

O vídeo sugerido a seguir explica de forma didática o uso da DCT no formato JPG. Ele cita o modelo YCbCr de cores, com uma componente de luminância (Y) e duas componentes de cromaticidade (Cb e Cr), como o modelo HSL visto na Unidade 1. Cita, também, o algoritmo de compactação de Huffman, que está fora do escopo de processamento de

imagens. Para compreender o vídeo, é suficiente saber que o algoritmo de Huffman é de compactação de dados quaisquer, utilizado para a criação de arquivos do tipo ZIP ou RAR. No JPG, Huffman é aplicado após a compactação das componentes de frequência da DCT, nosso foco. Os 12 minutos iniciais do vídeo são suficientes.
COMPUTERPHILE. **JPEG DCT, Discrete Cosine Transform (JPEG Pt2)**. 22 maio 2015.

Além da DCT, podemos citar as transformadas de Hadamard, Walsh e wavelets (SHAPIRO; STOCKMAN, 2001). Todas podem ser usadas para compressão de imagens ou outros sinais unidimensionais, como os sinais de áudio. As transformadas wavelets são largamente utilizadas em pré-processamento de imagens para fins de segmentação, assunto da próxima seção.

Como todas essas transformadas usam dos mesmos princípios de decomposição do sinal em sinais periódicos conhecidos, como os senos e cossenos da transformada de Fourier, compreendê-las em estudos complementares não será um problema para você que já conhece a transformada de Fourier. Novas possibilidades se abrem e você é capaz de explorá-las!

Sem medo de errar

Você está alocado em um projeto de segmentação de sementes e sua atividade atual é preparar a imagem, aplicando filtros de pré-processamento. Você já avaliou os filtros no domínio espacial e, agora, deve avaliar os filtros no domínio da frequência.

Sabe-se que os filtros no domínio espacial, apresentados no relatório da atividade anterior, são computacionalmente muito custosos, e que a utilização de filtros no domínio da frequência é mais eficiente. Seu trabalho, agora, é demonstrar a aplicação, no domínio da frequência, dos mesmos filtros até então aplicados no domínio espacial. Você deve entregar um relatório mostrando a diferença da aplicação dos filtros no domínio da frequência com relação à aplicação no domínio espacial, com o código fonte que permite a criação do relatório.

A partir dos resultados da atividade anterior e com o conhecimento do teorema da convolução, esta atividade se torna trivial. Na atividade anterior, você concluiu que o melhor filtro, dentre os avaliados, foi a soma dos valores absolutos dos filtros de Sobel vertical e horizontal. Vamos aplicar esse mesmo filtro utilizando o teorema da convolução. Primeiramente, vamos criar os filtros no domínio espacial (*kernels*).

```
def kernelSobelUD(f):
    return np.array([[-1,-2,-1],
                     [ 0, 0, 0],
                     [ 1, 2, 1]])


def kernelSobelLR(f):
    return np.array([[-1,0,1],
                     [-2,0,2],
                     [-1,0,1]])
```

Agora, para uma imagem f, vamos aplicar esses filtros no domínio da frequência. Utilizaremos a implementação da FFT disponível no NumPy.

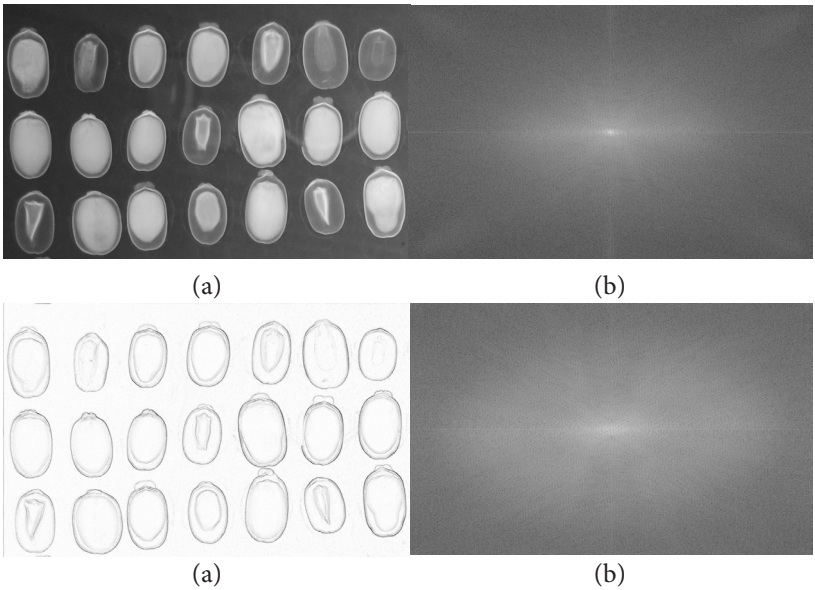
```
def filtroTeoConv(f,g):
    (gy,gx) = g.shape
    gg = np.zeros(f.shape) # filtro nas dimensões de f
    gg[0:gy,0:gx] = g
    F = np.fft.fft2(f)
    G = np.fft.fft2(gg)
    return np.fft.ifft2(F*G)
```

O código acima ajusta as dimensões do filtro g para as dimensões da imagem f, e o código a seguir carrega a imagem de entrada f e aplica o filtro combinado.

```
f = np.asarray(Image.open('semente.png'))

sobelLR = abs(filtroTeoConv(f,kernelSobelLR(f)))
sobelUD = abs(filtroTeoConv(f,kernelSobelUD(f)))
hS = inverte(normaliza(sobelLR+sobelUD))
```

O resultado é apresentado na Figura 4.17.

Figura 4.17 | Aplicação combinada dos filtros de Sobel de realce de bordas vertical e horizontal: (a) imagem original; (b) espectro de Fourier de (a); (c) imagem filtrada; e (d) espectro de Fourier de (c), mostrando o realce das altas frequências



(a) (b)

(a) (b)

Fonte: elaborada pelo autor.

Percebe-se que é o mesmo resultado obtido na seção anterior. Para concluir seu relatório, basta implementar a aplicação, no domínio da frequência, dos demais filtros aplicados na seção anterior, estendendo o código acima nos mesmos moldes. Desta forma, você demonstrará que sabe aplicar filtros no domínio da frequência.

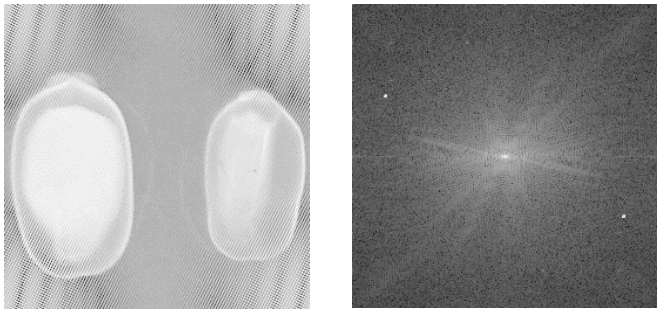
Avançando na prática

Remoção de ruído

Descrição da situação-problema

O scanner de raio X com o qual você está trabalhando insere um ruído intrínseco nas imagens de sementes de café que podem prejudicar o seu trabalho. Por se tratar de um ruído intrínseco, você deve estudá-lo diretamente. A Figura 4.18 mostra a imagem ruidosa e seu espectro de Fourier.

Figura 4.18 | Imagem ruidosa (a) e seu respectivo espectro de Fourier (b)



(a) (b)

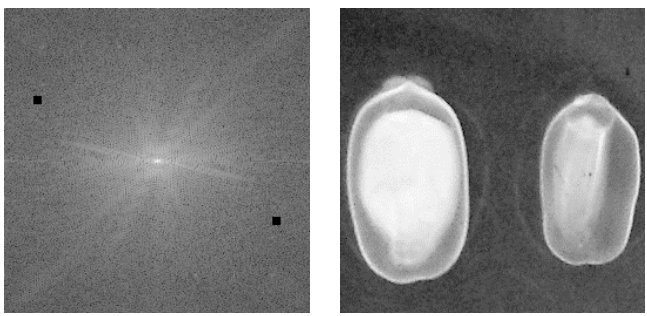
Fonte: elaborada pelo autor.

Observa-se que o scanner insere uma textura sobre a imagem. Você deve verificar seu espectro de Fourier e propor um filtro construído manualmente, diretamente no domínio da frequência, para remover esse ruído.

Resolução da situação-problema

Nota-se, na Figura 4.18(b), que há dois pontos brilhantes de alta frequência isolados no espectro de Fourier. Nota-se, também, que a reta que liga esses dois pontos está no sentido da textura que aparece na Figura 4.18(a). Para remover o ruído, você deverá aplicar um filtro construído de forma manual para literalmente apagar esses pontos brilhantes do ruído. A Figura 4.19 mostra o espectro de Fourier com esse ruído apagado e a imagem obtida como resultado pela transformada inversa de Fourier.

Figura 4.19 | Remoção manual de ruído



Fonte: elaborada pelo autor.

Há uma diferença no brilho, também resultante da remoção do ruído.

Faça valer a pena

1. O Teorema da Convolução, de Fourier, permite um ganho computacional no processamento de imagens e outros sinais digitais. A convolução é uma operação de custo computacional na ordem de $O(n_2)$, enquanto a transformada de Fourier pode ser calculada em tempo $O(n\log_n)$ pelo algoritmo da FFT (*Fast Fourier Transform*).

O que diz o Teorema da Convolução? Assinale a alternativa correta.

a) A operação de convolução é uma operação separável e, portanto, paralelizável, de forma que a Transformada Discreta de Fourier pode ser computada em tempo $O(n\log_n)$.

b) A convolução $f * g$ no domínio espacial é equivalente à multiplicação ponto a ponto $F \cdot G$ no domínio da frequência, sendo $f \leftrightarrow F$ e $g \leftrightarrow G >$, pares da transformada de Fourier.

c) A Transformada Discreta de Fourier é uma operação separável e, portanto, paralelizável, de forma que pode ser computada em tempo $O(n\log_n)$ pela FFT.

d) A convolução $f * g$ no domínio espacial é equivalente à multiplicação ponto a ponto $f \cdot G$ no domínio da frequência, sendo $g \leftrightarrow G$ um par da transformada de Fourier.

e) O cômputo da convolução é mais rápido no domínio da frequência, ou seja, é mais rápido calcular $F * G$ do que $f * g$, sendo $f \leftrightarrow F$ e $g \leftrightarrow G$, pares da transformada de Fourier.

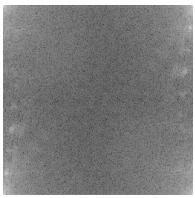
2. Observe as imagens a seguir:

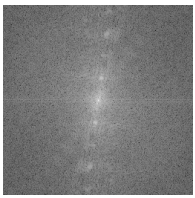
(a) (b)

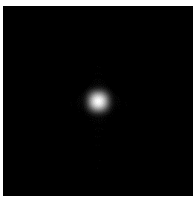
Fonte: elaborada pelo autor.

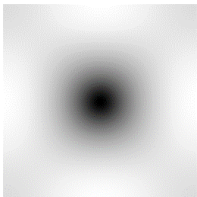
A imagem (b) foi obtida pela aplicação de um filtro sobre a imagem (a), o qual é um filtro de suavização (*blur*) gaussiano, construído no domínio espacial, mas aplicado no domínio da frequência, com base no teorema da convolução.

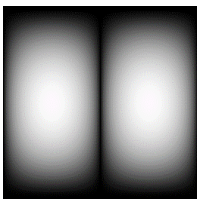
Considere g o filtro no domínio espacial aplicado sobre a imagem (a) e o par de transformadas de Fourier $g \leftrightarrow G$. Assinale a alternativa que mostra a magnitude de G.

a) 

b) 

c) 

d) 

e) 

3. A seguir, são apresentados filtros de convolução conhecidos na área de processamento de imagens digitais.

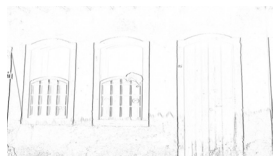
$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \qquad \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

1 2

Considere as imagens a seguir:

(a)

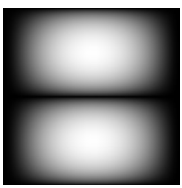
(b)

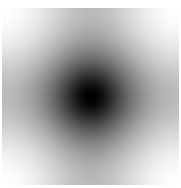


(c)

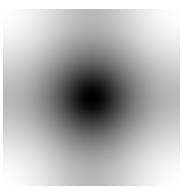
Com base nos seus conhecimentos sobre os filtros acima a respeito dos resultados que eles apresentaram e da Transformada de Fourier, assinale a alternativa que mostra os espectros de Fourier dos Filtros 1 e 2, respectivamente.

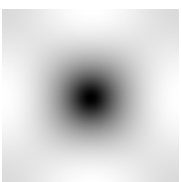
a)



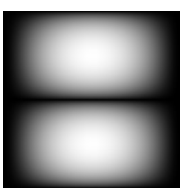


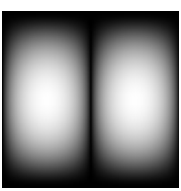
b)



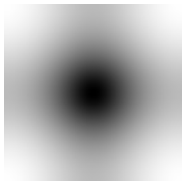


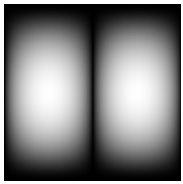
c)



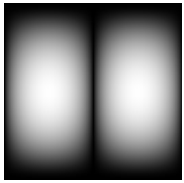


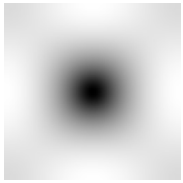
d)





e)





Seção 4.3

CGPI: segmentação de imagens

Diálogo aberto

Caro aluno, a aplicação de filtros sobre imagens é muito útil para aplicativos de edição e também como pré-processamento para sistemas de visão computacional, que buscam extrair informações das imagens. Outra ferramenta muito importante é a segmentação, que consiste em delinear objetos ou regiões de interesse nas imagens.

Se você deseja obter ferramentas que reconheçam pessoas em fotografias, como as existentes nas redes sociais, a primeira coisa a fazer é encontrar e delinear as faces de cada pessoa que aparece na fotografia, ou seja, você precisa realizar uma segmentação das faces nela, que é uma etapa essencial para as atividades de extração de informações de imagens digitais. Nesta seção, você conhecerá algumas técnicas de segmentação e poderá aplicá-las em uma atividade prática.

O projeto em que você está alocado visa avaliar se existem correlações entre a qualidade de bebida de um lote de grãos de café e o que pode ser obtido como características de imagens capturadas das sementes de café do mesmo lote. Antes de medir características da semente, você deve segmentá-las. Mais uma aplicação em que a segmentação de imagens é necessária.

Após estudar os filtros de processamento de imagens no domínio espacial e da frequência, só lhe falta realizar a segmentação das sementes.

Você já apresentou em relatórios a proposta de aplicação de filtros de pré-processamento sobre as imagens de semente de café e convenceu sua equipe a aprovar sua proposta. Demonstre, agora, que seu trabalho anterior foi correto e aplique um algoritmo de segmentação para finalizar o produto desta etapa do trabalho. Demonstre que é possível segmentar as sementes com o algoritmo de watershed, por ele ser fácil e rápido. Você deve entregar a imagem segmentada e o código fonte.

Esta seção possui atividades práticas, as quais auxiliarão na compreensão de todo o conteúdo da unidade. Execute-as sempre, consultando a documentação das ferramentas, buscando fixar os conceitos, e não somente implementar o que é pedido. Assim, você estará pronto para enfrentar novos desafios de processamento de imagens. Bom trabalho!

Não pode faltar

Caro aluno, a segmentação é o processo que subdivide uma imagem em objetos ou partes que a constituem. É a detecção e o delineamento de regiões de interesse. Matematicamente, o problema da segmentação de imagens pode ser modelado de diversas formas, e a literatura em torno do tema é muito rica.

Para escolher uma técnica de segmentação, o profissional deve considerar a precisão desejada, o tempo de processamento e, principalmente, a eficácia da técnica para as características da imagem a ser segmentada. Sim, cada tipo de problema de segmentação de imagem pode demandar uma técnica diferente. A segmentação e o acompanhamento (*tracking*) de pessoas ou objetos em movimento, em tempo real, por exemplo, exigem um algoritmo rápido, mas não necessariamente preciso, pois o objetivo é apenas seguir um objeto. Por sua vez, um sistema de medição de características de objetos, como o exemplo das sementes de café, exige um algoritmo preciso, mas não necessariamente tão rápido, por se tratarem de imagens estáticas.

As técnicas de segmentação de imagens podem ser subdivididas em dois grupos: **baseada em regiões** e **detecção de bordas**. Cada grupo possui, ainda, subdivisões considerando a base matemática utilizada para modelar o problema de segmentação. As técnicas baseadas em regiões podem utilizar desde modelos simples, como o de cálculo de limiar, até modelos complexos, como problemas em grafos (PENG; ZHANG; ZHANG, 2013). As técnicas de detecção de bordas podem ser tão simples como a aplicação de filtros de Sobel, ou tão complexas como o treinamento e aplicação de *deep learning* (GARCIA-GARCIA *et al.*, 2018).

Em geral, as técnicas baseadas em regiões buscam associar, em uma mesma região, pixels que se pareçam, de acordo com critérios de **similaridade**, enquanto as técnicas de detecção de borda buscam a **dissimilaridade**, que são os pixels da fronteira ou separação entre regiões. A seguir, estudaremos algumas dessas técnicas.

Segmentação baseada em gradiente

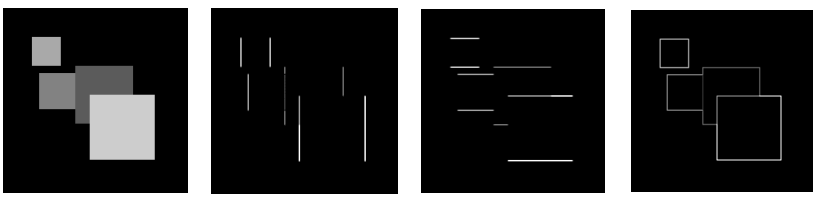
As técnicas de segmentação baseadas em gradiente visam à detecção das dissimilaridades no entorno de cada ponto da imagem. A derivada de uma função f de uma variável mostra, para cada ponto de f, o quanto há de variação em seu entorno. Quando se trata de funções de duas ou mais dimensões, esta variação pode ser medida por derivadas parciais em cada direção. O vetor $\nabla f_p = \left\langle \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2} \right\rangle$ é um vetor de derivadas parciais da função f(x,y) para o

ponto p=(x_1,y_1). Ele pode ser estendido para várias dimensões, e é chamado de **gradiente** (ROGAWSKI; ADAMS, 2018). O gradiente de uma função, portanto, é uma função vetorial que mostra as dissimilaridades em cada ponto da função.

O filtro de Sobel apresentado na seção anterior calcula uma aproximação da norma do gradiente em cada ponto da imagem, mas cada rotação do filtro de Sobel aproxima a derivada parcial em uma direção. O filtro $UD = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$ aproxima a derivada parcial na direção y (vertical), e o filtro $LR = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$ aproxima a derivada parcial na direção x (horizontal).

A somatória dos valores absolutos dos resultados da aplicação de cada rotação do filtro de Sobel é a aproximação da norma do gradiente da imagem em ambas as direções. Para um pixel cujos vizinhos possuem níveis de cinza próximos (região homogênea), o filtro de Sobel resulta em valores próximos de zero, e para um pixel cujos vizinhos possuem níveis de cinza diferentes (região heterogênea), o filtro de Sobel resulta em valores distantes de zero. As regiões heterogêneas da imagem são as bordas dos objetos. Por isso, o filtro de Sobel é de detecção de bordas. O filtro de Sobel combinado (somatória do resultado das rotações do filtro de Sobel) é o **filtro de detecção de bordas de Sobel** (SOLOMON, 2013). A Figura 4.20 mostra a aplicação dos filtros UD (*Up-Dow*, ou seja, vertical (b)) e LR (*Left-Right*, ou seja, horizontal (c)), isoladamente, assim como a somatória do resultado de ambos (d). Na Figura 4.20(d), é possível perceber o realce das bordas nas duas direções.

Figura 4.20 | Filtro de detecção de bordas de Sobel: (a) imagem original; (b) aplicação do filtro LR; (c) aplicação do filtro UD; e (d) combinação dos filtros de Sobel LR e UD



(a) (b) (c) (d)

Fonte: elaborada pelo autor.

Existem outros filtros similares ao de Sobel, como o filtro de Roberts, dado por $\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$ e suas rotações, e o de Prewitt, dado por $\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$ e suas rotações.

Todos eles realçam as bordas, mas o delineamento exato do contorno dos objetos exige um pós-processamento.

Reflita

O pós-processamento da imagem obtida por filtros de realce de bordas pode ser realizado pela aplicação de limiares para a delineação dos contornos dos objetos. Para imagens simples, como aos quadrados da Figura 4.20, o delineamento poderia ser obtido por um limiar, mas, e para imagens mais complexas, capturadas do mundo real? Seria trivial obter um bom limiar para a segmentação, ou o limiar poderia não resultar em contornos fechados para os objetos?

Segmentação por similaridade

A segmentação por similaridade consiste na agregação, em um mesmo objeto, dos pixels com características similares. Nesta abordagem, o objetivo é marcar todos os pixels que compõem o objeto, e não o seu contorno. Não há, portanto, a necessidade de pós-processamento.

A técnica mais simples e rápida de segmentação de imagens é a aplicação de um limiar (*threshold*). A Figura 4.21 ilustra a segmentação por limiar. É possível perceber que um limiar em 170 segmenta perfeitamente o quadrado maior da imagem sintética da Figura 4.21(a). Na Figura 4.21(c) o rosto do garoto, no entanto, possui níveis de cinza próximos aos dos reflexos de luz na parede ao fundo, e tudo foi segmentado como um mesmo objeto.

Figura 4.21 | Segmentação por limiar

(a) imagem sintética (b) limiar 170 de (a) (c) imagem real (d) limiar 170 de (c)

Fonte: elaborada pelo autor.

Mesmo com algoritmos de detecção automática do valor do limiar baseados no histograma, o problema observado na Figura 4.21(d) pode persistir. Uma alternativa é a segmentação de objetos por similaridade usando o **crescimento de regiões** (GONZALEZ; WOODS, 2011).

Exemplificando

Verifique a aplicação de diversos limiares diferentes sobre a imagem da Figura 4.21(c) utilizando um aplicativo de edição de imagens, como o Gimp (GIMP, 2019). Você pode testar tanto limiares fixos quanto limiares automáticos disponibilizados pela ferramenta e perceberá

que não é trivial encontrar um valor de limar que segmente a face do garoto.

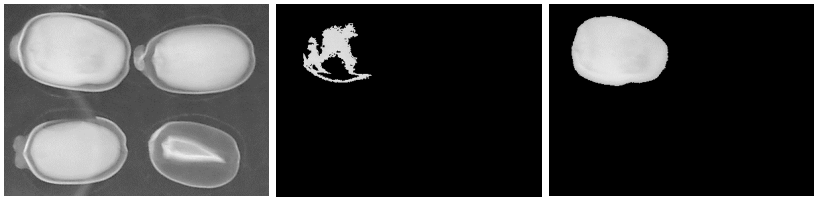
O crescimento de regiões consiste em marcar um ou mais pontos do objeto de interesse e, a partir desses pontos de referência, agregar ao objeto os pontos similares com base nas vizinhanças dos pixels. Considere uma ferramenta interativa de edição de imagens. Você deseja selecionar um dos quadrados da Figura 4.21(a). Você clica em um ponto do quadrado de interesse e tem todo o quadrado selecionado. O código a seguir mostra um algoritmo que faz isso com o uso de uma estrutura de dados de pilha. A função "vizinhos" retorna à lista de coordenadas dos vizinhos do ponto (x,y), respeitando os limites da imagem f.

```
def regioes(f,y,x,s=0):
    regiao = np.zeros(f.shape, dtype=np.uint8)
    regiao[y,x] = 1;     viz = vizinhos(y,x,f.shape)
    while len(viz) > 0:
        (yv,xv) = viz.pop()
        if (regiao[yv,xv] == 0):
            if ((f[yv,xv]>=f[y,x]-s)and(f[yv,xv]<=
             f[y,x]+s)): # é similar    regiao[y
              v,xv] = 1;      viz += vizinhos(y
              v,xv,f.shape)
    return regiao
```

Se o nível de cinza do pixel de referência p=(x,y) é g, o código acima considera como similar ao pixel p qualquer pixel com nível de cinza no intervalo [g-s,g+s]. Se s=0, somente pixels iguais a p são agregados ao objeto. A Figura 4.22 mostra o resultado do crescimento de regiões para uma imagem sintética (a) a (c) e para uma imagem real (d) a (f). A Figura 4.22(d) mostra que poucos pixels da semente são muito parecidos (s=5) com o pixel de referência, mas, se o critério de similaridade é mais amplo (s=40), a semente pode ser melhor detectada.

Figura 4.22 | Crescimento de regiões para diferentes valores de s

(a) (b) s=5 (c) s=40



(d) (e) s=5 (f) s=40

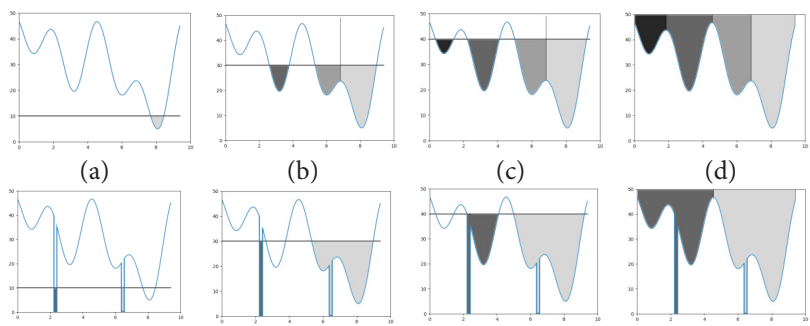
Fonte: elaborada pelo autor.

Os critérios de similaridade entre pixels podem ser diversos, e a literatura é vasta em algoritmos de crescimento de regiões. A seguir, apresentamos um desses algoritmos: o watershed.

Watershed

Watershed é uma técnica de segmentação de imagens baseada nos princípios do crescimento de regiões. Inicialmente, foi proposta por Beucher e Lantuéjoul (1979) como uma analogia ao conceito de bacias hidrográficas (*watershed*, em inglês). Sua definição inicial foi algorítmica, e diversos algoritmos foram apresentados na literatura para o cálculo cada vez mais eficiente do watershed. Posteriormente, o algoritmo foi matematicamente definido como um problema de matemática discreta, como proposto por Falcão, Stolfi e Lotufo (2004). Faz parte de uma linha de estudos em processamento de imagens chamada morfologia matemática (DOUGHERTY; LOTUFO, 2003). A ideia básica do watershed é ilustrada na Figura 4.23.

Figura 4.23 | Ilustração do watershed pela analogia da inundação. De (a) a (d): inundação a partir dos mínimos locais; de (e) a (h): inundação a partir de marcadores



(a) (b) (c) (d)

(e) (f) (g) (h)

Fonte: elaborada pelo autor.

A analogia da Figura 4.23 permite uma explicação rápida do watershed. Considere que a função é uma superfície com furos apenas nos pontos de mínimos locais. Você submergirá aos poucos essa superfície na água. À medida que você desce a superfície na água, esta começa a entrar pelos furos. A cada furo você atribui um rótulo. As águas que entram por dois furos diferentes não podem se misturar. Em (a), a água atingiu o nível 10, inundando um vale apenas; em (b), a água atingiu o nível 30 e foi construída uma barreira para impedir a mistura das águas dos dois vales da direita; em (c), a água atingiu o nível 40 e já está em todos os vales; finalmente, em (d), a água atingiu o nível máximo e foram construídas mais duas barreiras.

O processo de inundação é um processo de crescimento de regiões. A diferença é que, no crescimento de regiões apresentado anteriormente, dois pixels vizinhos são colocados na mesma região por um critério de similaridade dos seus níveis de cinza, enquanto no watershed os pixels são colocados na mesma região se pertencem ao mesmo vale ou bacia hidrográfica.

Assimile

O watershed é um algoritmo de crescimento de regiões que simula o conceito de bacias hidrográficas. As regiões no watershed crescem agregando pixels com base nos níveis de cinza, considerando a imagem como um relevo topográfico em processo de inundação.

Na definição **clássica** do watershed, cada mínimo local se torna uma região, o que pode não ser muito útil na prática, visto que as imagens reais possuem muitos mínimos locais de baixa relevância. Para guiar a segmentação, podemos utilizar o **watershed por marcadores**. As Figuras 4.23 (e) a (h) mostram a mesma sequência de inundação das figuras (a) a (d), mas para o watershed por marcadores, usando dois marcadores. Para cada marcador é criado artificialmente um vale profundo, e a superfície só tem furos nos marcadores. Em (f), a água já atingiu um nível acima do mínimo de um vale, mas o vale não foi inundado porque não tem o furo. O mesmo acontece em (g). Em (h), temos o resultado final com apenas duas regiões.

Pesquise mais

A animação gráfica 3D sugerida a seguir, trata do algoritmo watershed de segmentação de imagens:

MARCO ANTONIO G. CARVALHO. **Watershed – visualização 3D**. 22 out. 2012.

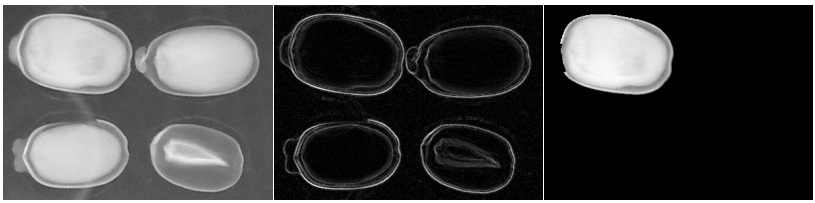
A implementação do watershed segue a lógica de crescimento de regiões. Para cada marcador, atribuiu-se um rótulo, e desses marcadores começa o crescimento. Como a inundação começa dos níveis de cinza mais baixos em direção aos mais altos, é necessário usar uma estrutura de dados de *heap* (no caso, uma árvore binária balanceada para acesso rápido aos dados) para implementar uma **fila de prioridades**. O pixel que crescerá será sempre o pixel de menor nível de cinza. O nível de cinza mais baixo é a prioridade. A biblioteca padrão do Python oferece uma implementação do *heap* no pacote *heapq*. Com isto, podemos implementar o watershed por marcadores como se segue:

```
import heapq as hq
def watershed(f,marcadores):
    # cria pq e saida
    h = [];    w = np.zeros(f.shape)

    # insere marcadores na pq, com prioridade 0
    for (y,x) in np.argwhere(marcadores>0):
        w[y,x] = marcadores[y,x]
        hq.heappush(h,(0,[y,x]))

    # iterações
    while len(h) > 0:
        (_,(y,x)) = hq.heappop(h)
        insereVizinhos(f,w,h,y,x)

    return w
```

A função *insereVizinhos* atribui a cada pixel vizinho o mesmo rótulo do pixel presente e insere os vizinhos na fila de prioridade, caso não tenham sido processados ainda. Cada pixel é inserido na fila com prioridade igual ao seu nível de cinza, que representa sua altitude na superfície. Na atividade prática desta seção, você terá em mãos o código fonte completo do watershed, incluindo a função *insereVizinhos*.

A Figura 4.24 mostra o resultado do watershed aplicado sobre a mesma imagem da Figura 4.22(d), utilizando um marcador no mesmo pixel de referência do algoritmo de crescimento de regiões que gerou a Figura 4.22(f), e um marcador na borda da imagem, representando o fundo. O watershed é aplicado sobre a imagem de gradiente da Figura 4.24(b), obtida pelo filtro de detecção de bordas de Sobel. Compare o resultado da Figura 4.24(c) com a Figura 4.22(f).

Figura 4.24 | Segmentação por watershed: (a) imagem original; (b) gradiente; e (c) resultado



(a) (b) (c)

Fonte: elaborada pelo autor.

Conhecendo o algoritmo de watershed, você já será capaz de resolver muitos problemas de segmentação de imagens. Mas, como já afirmado anteriormente, cada problema exige uma solução específica. A literatura é vasta em algoritmos de segmentação de imagens e, se você deseja trabalhar com processamento de imagens, poderá se aprofundar estudando outras técnicas de segmentação. Os principais conceitos foram passados, e as portas estão abertas. Bom trabalho!

Sem medo de errar

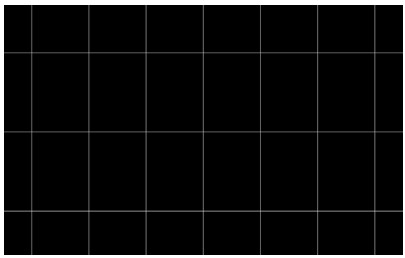
O seu problema é a segmentação de imagens de sementes de raio X de café. Você já demonstrou o conhecimento dos filtros no domínio espacial e da frequência e já sabe utilizá-los para preparar a imagem para ser submetida a um algoritmo de segmentação. Agora, você deve aplicar um algoritmo de segmentação para finalizar o produto do seu trabalho. Demonstre que é possível segmentar as sementes com o algoritmo de watershed. Você deve entregar a imagem segmentada e o código fonte.

A solução para este problema começa pelo reconhecimento de que as sementes de café estão bem alinhadas em uma grade. Primeiramente, vamos gerar uma grade contendo um pixel sobre cada semente. O código a seguir gera as coordenadas x e y das linhas e colunas, respectivamente, de uma grade com espaçamento constante entre linhas e colunas.

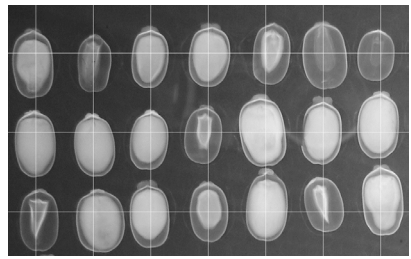
```
def grade():
    # grade
    yy = []
    for i in range(120,600,200):
        yy.append(i)
    xx = []
    for i in range(70,1010,145):
        xx.append(i)
    return (yy,xx)
```

A Figura 4.25 mostra esta grade desenhada.

Figura 4.25 | Grade sobre a qual foram posicionadas as sementes



Grade



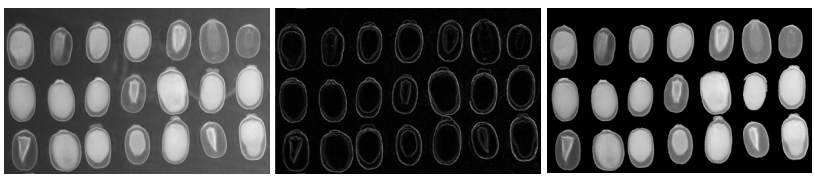
Grade sobreposta à imagem de sementes

Fonte: elaborada pelo autor.

Cada cruzamento das linhas e colunas da grade deve, então, ser um marcador de rótulo diferente. A borda da imagem será o marcador de fundo. Desta forma, teremos 22 marcadores, sendo um de fundo e 21 de sementes.

Aplica-se o filtro de detecção de bordas de Sobel para construir uma imagem de gradiente, em que os valores mais altos são as fronteiras entre objetos. Finalmente, aplica-se o watershed por marcadores para obter o resultado apresentado na Figura 4.26.

Figura 4.26 | Segmentação das sementes: (a) original; (b) gradiente; (c) segmentação.



(a) (b) (c)

Fonte: elaborada pelo autor.

A solução do problema de segmentação de sementes aplica todo o conhecimento de toda a unidade. Desta forma, você demonstra saber reconhecer e aplicar a segmentação de imagens usando watershed e filtros no domínio espacial e da frequência.

Avançando na prática

Preenchimento com balde de tinta

Descrição da situação-problema

Você está desenvolvendo um aplicativo para dispositivo móvel para uma empresa fabricante de tintas. O objetivo do aplicativo é que você possa tirar uma fotografia do ambiente que deseja pintar e testar diferentes cores, para fazer a melhor escolha. Você deve implementar a ferramenta de "preenchimento com balde de tinta". Com ela, o usuário tocará no ponto da parede que quer testar a cor, e a ferramenta atribuirá a cor desejada a toda a parede. Apresente a implementação dessa ferramenta.

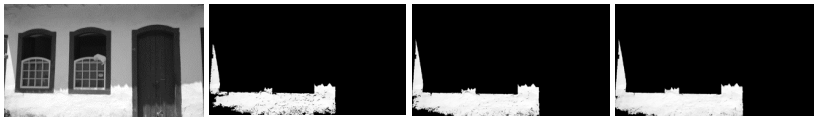
Resolução da situação-problema

A ferramenta de preenchimento com balde de tinta é interativa de segmentação de imagens. Utiliza uma técnica de agregação de pixels a um pixel de referência com base em um critério de similaridade: a cor. O algoritmo de crescimento de regiões apresentado nesta seção é o algoritmo implementado pela ferramenta.

Nos editores de imagem, é comum que esta ferramenta cresça a região a partir do pixel de referência p apenas sobre os pixels iguais a p. Isso funciona bem para imagens sintéticas, porém, para imagens reais, a variação de reflexões sobre uma superfície de mesma cor é grande, fazendo com que os valores dos pixels de uma área considerada homogênea por um ser humano não sejam idênticos.

No caso do aplicativo que você está desenvolvendo, as imagens são reais, capturadas pela câmera do dispositivo. Neste caso, você deve flexibilizar o critério de similaridade, como foi feito para a Figura 4.22(c). Isso deverá ser suficiente. A Figura 4.27 mostra a aplicação do crescimento de regiões para a foto de uma casa.

Figura 4.27 | Ferramenta balde de tinta



original s=15 s=30 s=50

Fonte: elaborada pelo autor.

O pixel de referência escolhido foi um pixel na parede branca logo abaixo da janela em que se encontra o gato. A parte sombreada da parede não foi agregada à região em nenhum dos casos, mas s=50 é o que dá o melhor resultado.

Faça valer a pena

1. Técnicas de segmentação de imagens podem ser subdivididas em dois grupos: baseadas em gradiente e baseadas em regiões. As técnicas baseadas em gradiente detectam ou realçam os pontos de dissimilaridade ou descontinuidade da imagem, que são as bordas dos objetos; e as técnicas baseadas em regiões detectam o conjunto dos pontos que compõem o interior do objeto, agregando pontos por similaridade.

Sobre as técnicas de segmentação de imagens, assinale a alternativa correta.

a) O filtro gaussiano é uma técnica de segmentação de imagens baseada em regiões, pois suaviza os contornos dos objetos.
b) O crescimento de regiões é uma técnica de segmentação baseada em gradiente que agrega os pixels de níveis de cinza próximos.
c) O filtro de Sobel combinado é uma técnica de segmentação baseada em regiões, pois calcula o gradiente da imagem.
d) O watershed é uma técnica baseada em regiões, mesmo que sua aplicação seja feita sobre o gradiente da imagem.
e) O limiar é uma técnica baseada em gradiente, e é o mais rápido algoritmo de detecção de bordas, mas pouco eficaz.

2. Considere a figura:

1 2 3 4

Fonte: elaborada pelo autor.

As imagens 2 a 4 foram obtidas da imagem 1 pelo algoritmo de crescimento de regiões, com diferentes graus de flexibilidade no critério de similaridade entre os pixels. Considere p o ponto de referência onde o crescimento de regiões se inicia. O ponto p escolhido foi um ponto sobre a testa do bebê.

Sobre o algoritmo de crescimento de regiões aplicado na figura, assinale a alternativa correta.

a) Na imagem 2, foram considerados similares ao ponto p somente os pontos iguais a p, como na ferramenta de 'balde de tinta'.
b) Nas três imagens, foram considerados similares ao ponto p os pontos de valor f(p)±s, sendo que s aumenta da imagem 2 para a 4.
c) Nas três imagens, foram considerados similares ao ponto p os pontos iguais a p, porém no gradiente da imagem 1.
d) Na imagem 4, s foi muito grande, e a região chegou à almofada. Isso pode ser evitado com um pré-processamento por um filtro gaussiano.
e) Nas três imagens, foram considerados similares ao ponto p os pontos de valor f(p)±s, porém no gradiente da imagem 1.

3. Considere a figura:

1 2 3

Fonte: elaborada pelo autor.

A imagem 1 é a original. A imagem 2 é o gradiente de 1. A imagem 3 é o resultado da segmentação de 1 usando o algoritmo de watershed por marcadores. As alternativas a seguir mostram diferentes opções de marcadores que podem ter sido utilizados para gerar o resultado da imagem 3. Nas alternativas, os marcadores aparecem em vermelho, sobrepostos à imagem original, mas cada quadrado vermelho é um marcador diferente.

Assinale a alternativa que contém os marcadores que foram utilizados na segmentação.

a)

b)

c)

d)

e)

Referências

BEUCHER S.; LANTUEJOUL, C. **Use of watersheds in contour detection**. International workshop on image processing, real-time edge and motion detection. Rennes, França: [s.n.], 1979.

BRIGHAM, O. E. **The Fast Fourier Transform and its applications**. São Paulo: Pearson, 1988.

BUSSAB, W. O.; MORETTIN, P. A. **Estatística básica**. 9. ed. São Paulo: Saraiva, 2017.

DANIELSON, G. C.; LANCZOS, C. Some improvements in practical Fourier analysis and their application to x-ray scattering from liquids. **Journal of the Franklin Institute**, v. 233, n. 4, p. 365-380, 1942.

DAVIES, E. **Computer & machine vision:** theory, algorithms, practicalities. 4. ed. Rio de Janeiro: Elsevier, 2012.

DOUGHERTY, E. R.; LOTUFO, R. A. **Hands-on Morphological Image Processing**. Washington: SPIE Publications, 2003.

FALCAO, A. X.; STOLFI, J.; LOTUFO, R. A. The image foresting transform: theory, algorithms, and applications. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 26, n. 1, p. 19-29, 2004.

GARCIA-GARCIA, A. et al. A survey on deep learning techniques for image and video semantic segmentation. **Applied Soft Computing**, v. 70, p. 41-65, 2018.

GIMP. **Gimp** – GNU Image Manipulation Program. Disponível em: http://gimp.org. Acesso em: 5 jan. 2019.

GONZALEZ, R. C.; WOODS, R. E. **Processamento de Imagens Digitais**, 3. ed. São Paulo: Pearson, 2011.

KREYSZIG, E. **Advanced engineering mathematics**, 10. ed. Nova Jersey: Wiley, 2011.

PENG, B.; ZHANG, L.; ZHANG, D. A survey of graph theoretical approaches to image segmentation. **Pattern Recognition**, v. 46, n. 3, p. 1020-1038, 2013.

PONTI, M. A.; COSTA, G. B. P. **Como funciona *Deep Learning***. 2017. Disponível em: http://conteudo.icmc.usp.br/pessoas/moacir/papers/Ponti_Costa_Como-funciona-o-Deep-Learning_2017.pdf. Acesso em: 23 fev. 2019.

ROGAWSKI, J.; ADAMS, C. **Cálculo:** Volume 2. 3. ed. Porto Alegre: Bookman, 2018.

SHAPIRO, L. G.; STOCKMAN, G. C. **Computer Vision**. Nova Jersey: Prentice-Hall, 2001.

SOLOMON, C. **Fundamentos de processamento digital de imagens:** uma abordagem prática com exemplos em Matlab. São Paulo: LTC, 2013.

SWOKOWSKI, E. W. **Cálculo com geometria analítica**. 2. ed. São Paulo: Makron Books, 1995. v2.

ISBN 978-85-522-1367-3