PlantGoshi
Projeto Integrador III - Sistema Autônomo

Anderson J. Silva, Felipe R. de Luca, Nelson J. Dressler

[1]Bacharelado em Ciência da Computação – Centro Universitário Senac - Santo Amaro
São Paulo - SP - Brasil
2015

Resumo. *O projeto foi desenvolvido para a disciplina Projeto Integrador III: Sistema Autônomo, com o objetivo de aplicar técnicas e implementar algoritmos de visão computacional em um jogo de tema livre. Para tal, criamos um jogo digital em 2D, desenvolvido em linguagem C, onde o jogador deve cuidar de uma árvore em seu processo de crescimento, com o objetivo principal de colher os melhores frutos. Para isso, o jogador terá como ferramenta de interação uma varinha mágica, que permitirá aplicar poderes que interajam com os elementos dentro do jogo, contribuindo com o crescimento da árvore e impedindo que pragas ataquem os frutos. A interação da varinha com o jogo será por intermédio do reconhecimento dela nas imagens capturadas pela câmera de video instalada no computador, processadas por algoritmos baseados em levantamento bibliográfico.*

1. Introdução

O Projeto consiste num jogo de simulação de criação de plantas e do seu cuidado contra a presença de invasores e/ou criaturas nocivas à sua sobrevivência no seu habitat natural, a terra. O jogador terá poderes por intermédio de uma varinha mágica e, ao longo do jogo, deverá utilizá- los para alimentar e aprimorar a planta ou a árvore e também combater as pragas aparentes através de interação com diferentes cores emitidas pelo LED.

Os frutos nascem verdes em pontos randômicos e demoram um tempo x para seguir para a segunda etapa. Se o fruto no estado verde nõ for ”regado” (ou, poder de mÃºsica) depois de um tempo determinado, ele vai ficar maduro com praga ou ficará estragado. Isso força o usuário a usar o poder de ”regar” (ou, poder de música) no fruto.

Quando o fruto fica maduro, depois de um tempo x ele cria praga, forçando o usuário a usar o poder de ”remover” para eliminar a praga do fruto maduro.

Com o fruto maduro e sem praga o usuário poderá usar o poder de ”colher” para colher o fruto e somar pontos. Se o usuário colher fruto verde, maduro com praga ou estragado ele perde pontos. Caso deixe o fruto estragar também pontos são descontados. Depois que uma quantidade x de frutos estragar o jogo termina.

2. Estrutura principal do jogo

2.1. Layout

[SCREENSHOT DA TELA]

2.2. Etapas

A partida do jogo é dividida em três principais etapas:

1. **Nascimento e Crescimento da Árvore:** o jogador deverá estar atento a regar a árvore sempre que necessário e, ao mesmo tempo, combater ervas daninhas que irão crescer ao pé dela.
2. **Amadurecimento dos Frutos:** os frutos irão crescer mais rápido se o jogador utilizar notas musicais. Também irão crescer pragas nesses frutos, que podem ser combatidas com o poder de remoção de pragas.
3. **Colhimento dos Frutos:** é o momento no qual os frutos cresceram e amadureceram o suficiente para serem colhidos, contabilizando pontos para o jogador.

3. Visão Computacional

Compreendendo a parte de visão computacional, foi realizado um levantamento bibliográfico referente ao processamento digital de imagens, reconhecimento de padrões em imagens, operações aritméticas e um estudo aprofundado sobre os modelos de cores, sua natureza e suas características principais.

3.1. Algoritmos utilizados

3.1.1. HSV e RGB

Cada pixel de uma imagem extraída da câmera do computador no modelo de cores RGB (Red, Green, Blue) é convertido em HSV / HSB (Hue, Saturation, Value / Brightness), permitindo descobrir o grau da cor pura (Matiz), as faixas representada por cada cor, a porcentagem de saturação da cor (Pureza) e a porcentagem de brilho (Valor).

O H é a matiz e é medida em graus compreendendo valores de 0°a 359°. A faixa de cada uma das seis cores principais (primárias e secundárias) é definida numa margem de 60 graus. As faixas são classificadas da seguinte maneira: Vermelho (0°a 59°), Amarelo (60°a 119°), Verde (120°a 179°), Ciano (180°a 239°), Azul (240°a 299°) e Magenta (300°a 359°). O S é a saturação e é medida em porcentagem nos valores de 0 a 100%. Finalmente, o V é o brilho e medido também em porcentagem, (0 a 100%). Como observação importante, é possÃvel notar que as cores branco e preto são definidas de acordo com o valor de V: quando se aproxima de 0, emite a cor preta, e quando se aproxima de 100, branca.

Por exemplo, para reconhecer a luz do LED, é possível por intermédio de uma porcentagem alta do V (Valor de brilho) e sua cor pela faixa de H (Matiz).

Para manipular essa representação, foi necessário a criação de uma estrutura chamada Pixel que deve armazenar todos os dados de cores de apenas um pixel e a implementação de dois algoritmos que compreendem as conversões nos dois sentidos: RGB para HSV e HSV para RGB.

A fim de padronizar, foram implementadas também algumas funções de formatação: conversão de um valor decimal para porcentagem (S e V), conversão de um grau para um valor de 0 a 5 (H) e conversão de canais de cores RGB em um valor decimal entre 0 e 1.

Além disso, foram desenvolvidas funções complementares de máximo e mínimo dentre três valores (R, G e B).

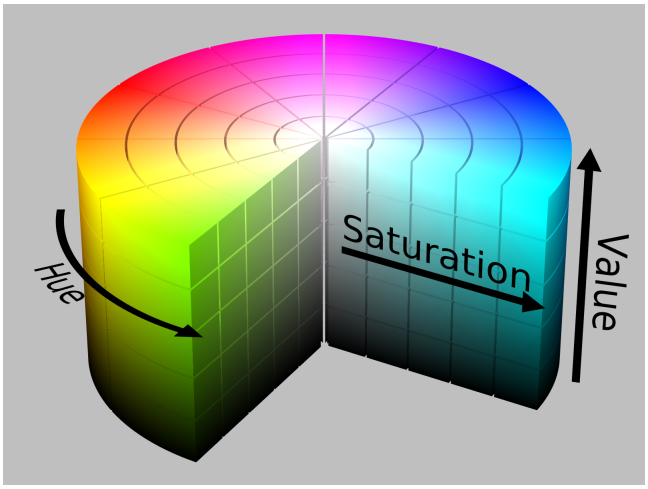

Figure 1. Representação tridimensional do espaço de cor HSV. Créditos: SharkD. http://en.wikipedia.org/wiki/HSL_and_HSV

3.1.2. Escala de cinza

Os valores RGB de um pixel podem ser convertidos em escala de cinza por uma série de métodos. Com esse fim, foi escolhido o método de escalas fixas correspondendo a cada um dos canais RGB. É efetuada a seguinte operação sobre cada pixel:

```
R x 0,3 + G x 0,51 + B x 0,18
```

Dessa maneira, cada pixel assume apenas um valor e, ao mesmo tempo, é mantida uma escala para cada valor RGB, levando em consideração a sua maior tonalidade.

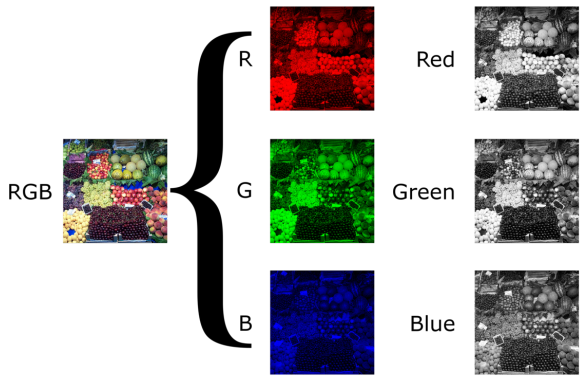

Figure 2. Conversão de canais de cores para escala de cinza. Créditos: Nevit Dilmen. http://en.wikipedia.org/wiki/Grayscale

3.1.3. Redução de cores

A redução de cores é uma técnica que tem por objetivo facilitar o reconhecimento de uma região específica, diminuindo as cores visíveis e tornando uma região mais destacada e isolada que as demais. Para tal, é usada a seguinte operação sobre cada canal RGB:

```
Canal = Canal x Fator + Fator / 2
```

sendo o fator um valor inteiro que define a proporção da quantidade cores que se quer diminuir na imagem. Exemplos de Fatores: 2, 16, 64, 128. Quanto maior o fator, menor a quantidade de cores visíveis.

Figure 3. Redução da quantidade de cores de acordo com valor do parâmetro K. Fonte: http://opencvpython.blogspot.com.br/2013/01/k-means-clustering-3-working-with-opencv.html

3.1.4. Controle de massas

Visando uma definição exata e correta do ponto central que deve ser devolvido pela função da visão computacional, foi necessário a implementação de um mecanismo para ter ideia caso outras massas brancas além da principal (o LED) está sendo retornada e, com isso, comprometendo a jogabilidade do jogo.

Essa função analisa a vizinhança de cada um dos pixels da imagem contabilizando as massas brancas presentes em tela.

Para isso, é criada uma estrutura bidimensional chamada Matriz de Vértices, onde cada vértice armazena informações relevantes sobre cada pixel: valor definindo se o pixel estÃ¡ aceso (1 - branco) ou apagado (0 - preto) e o numero de massa definido por uma variável contadora.

Ao final, o algoritmo retorna um número inteiro compreendendo a quantidade de massas brancas encontradas.

3.1.5. Erosão e dilatação

A Erosão e a Dilatação são técnicas de filtragem com o objetivo de reduzir ruídos presentes na imagem, que possam dificultar a detecção de pontos principais na tela.

Os ruídos podem ser causados por uma série de fatores, dentre eles limitações da câmera presente no computador e presença de pontos isolados na tela que tenham características similares aos procurados.

A Erosão trata de eliminar os pontos isolados na imagem, os quais não possuem outros em destaque em sua vizinhança.

A Dilatação trata de aumentar os pontos isolados na imagem, consistindo no processo contrário da Erosão.

Com uma sequência razoável de operações de Erosão e Dilatação, a quantidade de ruídos é descartada, ao mesmo passo que a região que deve ser destacada é mantida em seu tamanho original.

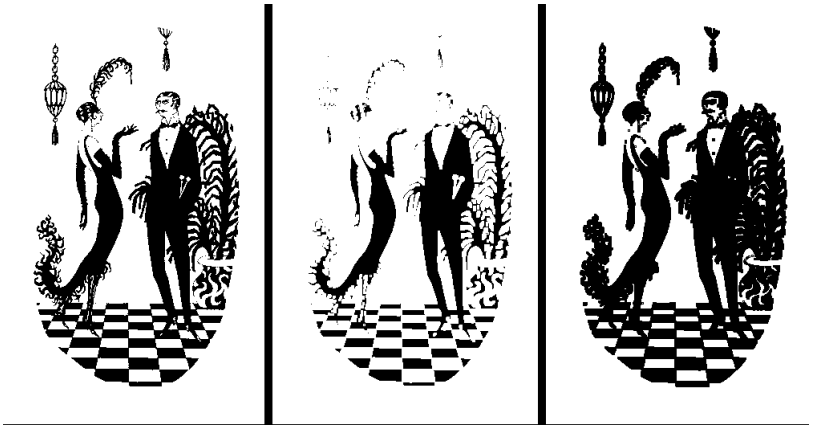

Figure 4. Esquerda: Imagem original. Meio: erosão. Direita: dilatação. Fonte: http://visiblevisible.org/teaching/setpixel/students/katherine/images/erodeDilate.png

3.2. Aplicação dos algoritmos

Utilizando os algoritmos descritos em **3.1**, foi criada uma função de visão computacional que aplica cada um desses algoritmos para retirar informações sobre os pixels da imagem.

Primeiro, o canal RGB é convertido em HSV antes e após a aplicação do efeito de reduç ao de cor para cruzar as informações de luminosidade com a cor específica do LED a ser detectado pelo algoritmo, respectivamente.

Após isso feito, é armazenado numa Matriz Binária a região do LED, definindo o valor 1 para pixel com a cor e luminosidade detectada e 0 para não detectada.

Essa matriz passa por uma sequência de Erosões e Dilatações até que a região principal seja destacada e os ruídos eliminados. Essas operações ocorrem por intermédio de uma matriz chamada de Máscara.

Posteriormente, na matriz binária é verificada a quantidade de massas brancas presentes e, caso haja mais de uma, o LED é simplesmente ignorado.

Caso haja apenas uma massa, é calculada a média das posições x e y onde os pixels estão acesos e é devolvida uma coordenada (x, y) central. Para isso, é foi criada uma estrutura chamada Visão Computacional que armazena o par ordenado da região central do LED, a cor detectada pelo mesmo e faixa de luminosidade que é testada no momento da calibração da câmera pelo algoritmo.

4. Simulação da árvore

Adotamos a estrutura de dados de árvore ternária para construir a árvore do jogo. Essa estrutura é basicamente constituida em pontos de crescimento que se dividem no máximo

em três partes cada um. Essas partes dão origem aos galhos que crescem até determinado tamanho, decidido aleatóriamente. As pontas de cada galho são novos pontos de crescimento, que se dividem de novo e dão origem a novos galhos

Para que simulação tenha um aspecto e comportamento mais próximo de uma árvore, elaboramos uma solução que inclui a matéria estudada na disciplina de Estrutura de Dados e elaboramos algoritmos para simular o crescimento da árvore, dos galhos e dos frutos. Basicamente a árvore cresce de acordo com um valor de energia de crescimento fornecida a ela logo no início do jogo. Uma parte dessa energia é consumida pelo tronco e o restante é distribuido de maneira aleatória para os próximos galhos que irão nascer, de acordo com o seguinte critério:

```
SE galho.energiaConsumida == ((galho.energiaLimite * 50) / 100) E
galho.temFilhos == FALSO ENTÃO galho.criarFilhos = SIM
```

Isso se repete até que não haja mais energia suficiente para repassar ao galho seguinte. O jogador tem a oportunidade de fornecer mais energia ao longo da partida, o que irá proporcionar uma árvore mais desenvolvida e com mais frutos para serem colhidos.

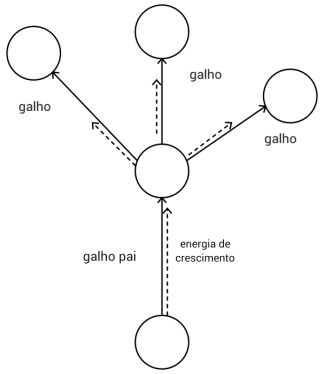

Figure 5. Crescimento dos galhos e energia transferida de pai para filho.

4.0.1. Crescimento

Para que os galhos crescam controladamente mas mantendo certa diferenciação em tamanho e direção entre um e outro, estabelecemos um critério que chamamos de energia de crescimento e energia limite. A energia limite determina o quanto cada galho da árvore vai crescer e é estabelecido através de uma porcentagem, calculada a partir do total de energia de crescimento fornecida para o galho, o nível de altura desse galho e a energia limite da árvore:

```
galho.energiaLimite = (energiaRecebida * (arvore.energiaLimite -
(galho.profundidade * 2)) / 100)
```

O valor da energia limite pode ser alterado de acordo com a interação do usuário ao longo do jogo, o que pode proporcionar uma árvore mais mais ou menos desenvolvida.

Enquanto os galhos crescem eles podem dar origem a novos galhos. Isso é determinado após ser consumida uma determinada quantidade de energia de crescimento fornecida ao galho, o que permite que a simulação de crescimento da árvore fique mais natural. A única exceção é o tronco da árvore, que tem o crescimento mais controlado:

```
    SE galho.energiaConsumida < galho.energiaLimite E
galho.energiaRecebida > 0 ENTÃO galho.crescer
```

4.0.2. Pontos de crescimento de frutos e folhas

Os pontos de crescimento de frutos e folhas são determinados de acordo com o mínimo de energia que o galho tem para crescer. Caso esse valor seja igual ou abaixo de determinado critério, o algoritmo de simulação assume que não irão surgir novos galhos a partir do galho atual, ainda durante a fase de crescimento. Com isso determinado, o galho passa a produzir folhas e frutos ao invés de novas ramificação de galhos.

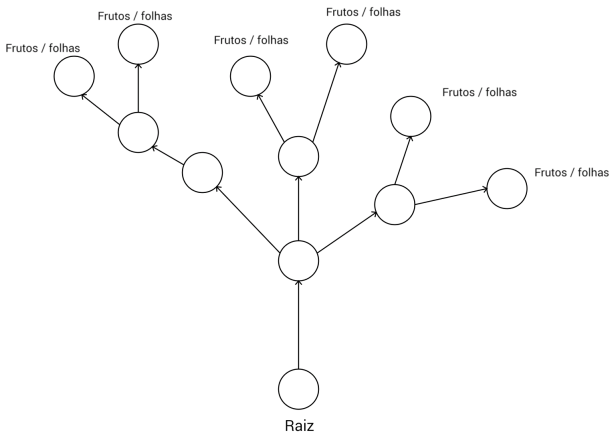

Figure 6. Estrutura da árvore e pontos de crescimento de frutos e folhas

4.1. Primeiros experimentos

Para o reconhecimento de uma cor emitida por um LED do Arduino, fizemos uma otimização física com um pedaço de papel que reduzisse o brilho da luz, porém ainda não é suficiente para o reconhecimento total.

Como experimentos necessários, temos o reconhecimento da luminosidade para verificar a proximidade da luz com a câmera e uma otimização afim de isolar a cor do LED, ignorando as outras cores similares e aparentes na câmera.

5. Bibliotecas

5.1. OpenCV 2.4.11

OpenCV (Open Source Computer Vision Library) é uma biblioteca de código aberto de visão computacional e aprendizado de máquina, desenvolvida para dar suporte a aplicativos que requerem processamento de imagens.

5.2. Allegro 5.0

O Allegro é uma biblioteca multiplataforma de programação voltada para desenvolvimento de jogos. Ela oferece suporte para programação baixo nível em C e C++, ou seja, fornece ferramentas para que o usuário desenvolva sua própria programação de jogo.

5.3. Arduino-Serial

Biblioteca de código aberto em linguagem de programação C, que oferece suporte a comunicação via porta serial do computador com o Arduino.

5.4. Linguagens de programação

O projeto inteiro foi desenvolvido em linguagem C padrão 99.

6. Equipamentos

- Câmera de captura de vídeo
- Placa controladora Arduino Uno
- LED RGB
- Computador (desktop ou notebook)

7. Bibliografia

References

Cavalcanti, J. Disciplina de computação gráfica. http://www.univasf.edu.br/~jorge.cavalcanti/comput_graf06_Cores.pdf.

Conci, A., Azevedo, E., and Leta, F. R. Computação gráfica. http://computacaografica.ic.uff.br/transparenciasvol2cap4.pdf.

Cook, J. D. (2009). Three algorithms for converting color to grayscale. http://www.johndcook.com/blog/2009/08/24/algorithms-convert-color-grayscale/.

E, A., A, C., and R, L. F. (2009). *Computação Gráfica: teoria e prática*, volume 2. Campus.

Itseez. Opencv. http://opencv.org/.

Kurt, T. E. (2006). Arduino-serial. http://todbot.com/blog/2006/12/06/arduino-serial-c-code-to-talk-to-arduino.

Marengoni, M. and Stringhini, D. Tutorial: Introdução à visão computacional usando opencv. http://seer.ufrgs.br/rita/article/viewFile/rita_v16_n1_p125/7289.

Parker, J. R. (2011). *Algorithms for Image Processing and Computer Vision*. Wiley Publishing, Inc., 10475 Crosspoint Boulevard. Indianapolis, IN 46256, 2nd edition.

RapidTables. Rgb to hsv color conversion. http://www.rapidtables.com/convert/color/rgb-to-hsv.htm.

Team, A. Allegro 5. http://alleg.sourceforge.net/readme.html.

Wright, S. (2010). *Digital Compositing for Film and Video*. Focal Press, 3rd edition edition.