

1. Introdução

GPUs (*Graphics Processing Unit*) são um tipo de *hardware* acelerador capaz de realizar simultaneamente um número massivo de operações de ponto flutuante. Sua finalidade original era o processamento de aplicações gráficas, já que envolvem a execução das mesmas instruções (cálculos matemáticos intensos) sobre um grande conjunto de dados (milhões de *pixels* de uma imagem).

Atualmente, GPUs também são utilizadas para processamento de propósito geral (GPGPU), com destaque para o treinamento de modelos de *machine learning*. Neste EP, vamos “voltar às origens” e utilizar CUDA para aplicar um efeito simples em uma imagem: mudança de matiz (*hue*).



Figura 1: Comparação entre representações simplificadas das arquiteturas de uma CPU e uma GPU [1].

No sistema RGB, as cores são descritas como combinação das componentes primárias R, G e B (*red*, *green* e *blue*). Já no sistema HSV (*hue*, *saturation* e *value*), as cores são descritas por outras três componentes associadas à forma como nós as percebemos:

- **matiz (*hue*)**: representa a tonalidade dominante da cor e é medido em graus de um círculo de cores (de 0 a 360 graus). Os valores de matiz correspondem a diferentes cores, como vermelho, amarelo, verde, azul, etc.

- **saturação (saturation)**: indica a intensidade ou pureza da cor. Valores mais baixos de saturação resultam em cores mais desbotadas ou acinzentadas, enquanto valores mais altos resultam em cores mais vivas e intensas.
- **valor (value)**: representa o brilho ou a luminosidade da cor. Valores mais altos de valor resultam em cores mais claras e brilhantes, enquanto valores mais baixos resultam em cores mais escuras.

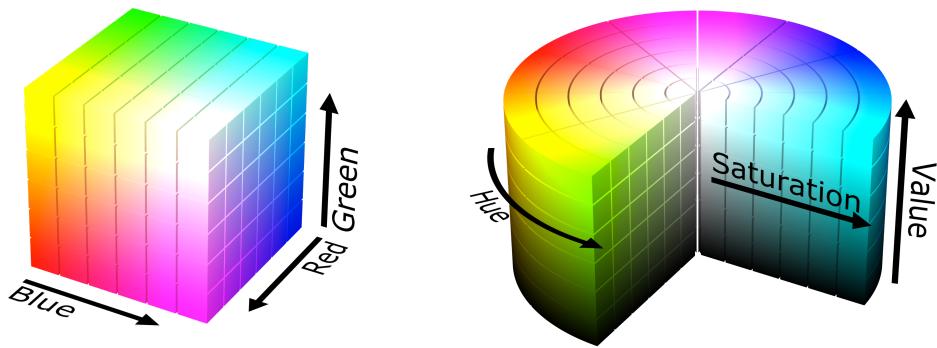


Figura 2: Comparação entre dois modelos de cores: RGB (esquerda) e HSV¹.

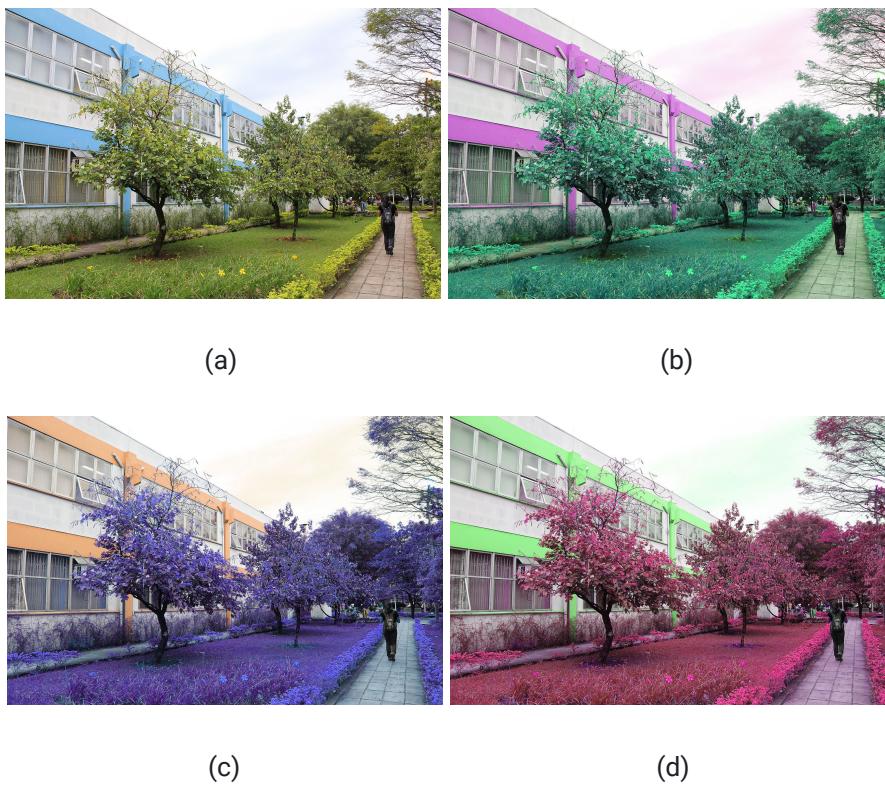


Figura 3: Exemplos de mudança de matiz: imagem original² (a), mudança de 0.25 (b), mudança de 0.5 (c) e mudança de 0.75 (d).

¹ [File:RGB color solid cube.png - Wikimedia Commons](#)

[File:HSV color solid cylinder.png - Wikimedia Commons](#)

² Imagem original: [Marcos Santos/USP Imagens](#)

A Figura 2 mostra que o sistema HSV pode ser considerado, de certa forma, um sistema de coordenadas polares em relação ao sistema RGB, de coordenadas cartesianas, de modo que uma mudança no matiz corresponde a uma rotação no sistema HSV. Assim, ao alterar o matiz de cada *pixel* da imagem, mudam-se as cores sem alterar a saturação e o brilho, como ilustrado pela Figura 3.

Uma forma de alterar o matiz é: converter a imagem de RGB para HSV, modificar o valor do matiz (*hue*), e então converter de volta para RGB. Se r , g e b são as componentes RGB de um *pixel*, então essas operações podem ser descritas pela multiplicação de uma matriz³ pelo vetor coluna (r , g , b), para cada *pixel*:

$$\begin{pmatrix} r' \\ g' \\ b' \end{pmatrix} = \begin{pmatrix} A & B & C \\ C & A & B \\ B & C & A \end{pmatrix} \cdot \begin{pmatrix} r \\ g \\ b \end{pmatrix}$$

$$A = \cos \alpha + \frac{1}{3} \cdot (1 - \cos \alpha)$$

$$B = \frac{1}{3} \cdot (1 - \cos \alpha) - \frac{1}{\sqrt{3}} \cdot \sin \alpha$$

$$C = \frac{1}{3} \cdot (1 - \cos \alpha) + \frac{1}{\sqrt{3}} \cdot \sin \alpha$$

Onde α é o ângulo de desvio do matiz. Como essa mesma operação é realizada para cada um dos milhões de *pixels* de uma imagem (multiplicação por uma matriz constante), trata-se de um problema adequado para ser executado na GPU.

2. Código fornecido

O código fornecido para o trabalho está disponível no GitHub⁴. Há apenas um arquivo fonte (`hue_modify.cu`), além do `Makefile`. O código gera um executável `hue_modify` que lê uma imagem no formato `.png`, altera o seu matiz e gera uma imagem de saída no formato `.png` contendo o resultado. Foi fornecida uma função sequencial `modify_hue_seq`, que deverá ser paralelizada usando CUDA, completando o código nas funções `modify_hue` e `modify_hue_kernel`.

Para alternar entre a execução entre as versões sequencial e paralela, altere manualmente a função a ser chamada na `main` (`modify_hue_seq` ou `modify_hue`).

Para compilar e executar, é necessário ter o CUDA Toolkit⁵ instalado (que inclui o compilador `nvcc`), assim como acesso a uma GPU. Caso nenhum membro do grupo

³ [Shift hue of an RGB Color - Stack Overflow](#)

⁴ <https://github.com/vitorterra/MAC0219-5742-EP3-2023>

⁵ [CUDA Toolkit 12.2 Downloads | NVIDIA Developer](#)

tenha acesso a uma GPU localmente, é possível executar código em GPU na nuvem usando os créditos gratuitos do Google Colab⁶ (que foi o que aconteceu no desenvolvimento do próprio EP). Em caso de dúvidas sobre como utilizar o Google Colab, entre em contato com o monitor.

Para compilar o projeto, use o comando:

```
$ make
```

Para executar o projeto, use o comando:

```
$ ./hue_modify <input_file> <output_file> <hue_diff>
```

onde `input_file` é o nome da imagem de entrada (no formato `.png`), `output_file` é o nome da imagem de saída (também no formato `.png`) e `hue_diff` é o desvio no matiz da imagem, sendo um valor entre 0.0 e 1.0 (ângulo normalizado entre 0 e 360 graus).

Para debugar, a ferramenta `compute-sanitizer`⁷ pode ser útil, além das verificações de erro (`checkErrors`) presentes no código.

3. Tarefas e Entrega

Você e seu grupo deverão paralelizar o código para mudança de matiz utilizando CUDA, implementando as funções `modify_hue` e `modify_hue_kernel` presentes no código. A função `modify_hue` possui alguns comentários que indicam quais funções CUDA devem ser chamadas - vocês devem substituir as reticências pelos parâmetros corretos. No entanto, os comentários são apenas uma sugestão - fiquem à vontade para implementar as funções como desejarem, contanto que o resultado esteja correto.

Vocês deverão entregar no e-Disciplinas o arquivo `hue_modify.cu`, com os nomes dos integrantes no cabeçalho, contendo a implementação das funções `modify_hue` e `modify_hue_kernel`. Desta vez **não** é necessário entregar relatório, nem analisar a variação dos tempos de execução.

A nota do EP3 vai de **0.0** a **10.0**, e a avaliação será feita da seguinte maneira:

- Implementação da solução em CUDA: vale **10.0**, divididos da seguinte maneira:
 - Código compila sem erros e `warnings`: **4.0**
 - Código executa sem erros e produz o resultado correto: **5.0**
 - Boas práticas de programação e clareza do código: **1.0**

⁶ [TensorFlow with GPU - Colaboratory](#)

⁷ [Compute Sanitizer User Manual](#)

Em caso de dúvidas, use o fórum de discussão do e-Disciplinas ou entre em contato diretamente com o monitor (vitorterra@ime.usp.br) ou o professor (gold@ime.usp.br).

4. Referências

Este EP foi baseado em um dos trabalhos do curso *CS344 - Parallel Programming* do Udacity. O código também foi baseado nas amostras de CUDA disponibilizadas pelo Pedro Bruel [2].

[1] - NVIDIA Corporation. CUDA C Programming Guide. 9.0. ed. [S.I.], 2017.

[2] - https://github.com/phrb/PPD/blob/main/lectures/tex/cuda/code_samples/vecAdd