UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

ANNA TAVARES FRATUS

GERAÇÃO DE CONTORNOS ESTILIZADOS PARA CENAS 3D COM PRESERVAÇÃO DE ILUMINAÇÃO, COR E LIMITE DE OBJETOS

PONTA GROSSA

2022

ANNA TAVARES FRATUS

GERAÇÃO DE CONTORNOS ESTILIZADOS PARA CENAS 3D COM PRESERVAÇÃO DE ILUMINAÇÃO, COR E LIMITE DE OBJETOS

Generation of Stylized Outlines for 3D Scenes Preserving Lighting, Color and Boundary of Objects

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Ciência da Computação do Curso de Bacharelado em Ciência da Computação da Universidade Tecnológica Federal do Paraná.

Orientador: Drª. Itamar Iliuk

Coorientador: Drª. Mauren Louise Sguario Coelho De Andrade

PONTA GROSSA

2022

ANNA TAVARES FRATUS

GERAÇÃO DE CONTORNOS ESTILIZADOS PARA CENAS 3D COM PRESERVAÇÃO DE ILUMINAÇÃO, COR E LIMITE DE OBJETOS

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Ciência da Computação do Curso de Bacharelado em Ciência da Computação da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 08 de novembro de 2022

Itamar Iliuk
Doutora
Universidade Tecnológica Federal do Paraná

Helyane Bronoski Borges
Doutora
Universidade Tecnológica Federal do Paraná

Simone Nasser Matos
Doutora
Universidade Tecnológica Federal do Paraná

PONTA GROSSA

2022

Dedico este trabalho aos meus amigos da Miralumo, minhas orientadoras, e minhas professoras, por todo o apoio que ofereceram.

RESUMO

Nem sempre o fotorrealismo é a melhor forma de se transmitir um sentimento, uma sensação ou de se contar uma história específica. No contexto de filmes de animação 3D a estilização é utilizada como ferramenta tanto narrativa, quanto visual, porém muitas dessas soluções não são disponibilizadas para uso. Este trabalho propõe um método de geração de contornos estilizados, sensíveis a dados do 3D, como iluminação, cor e limite de objetos. O método promove controle artístico sobre o visual dos contornos, além de ser uma solução replicável e aberta.

Palavras-chave: renderizacao de contornos estilizados; non-photorealistic rendering (npr); opencv; detecção de bordas.

ABSTRACT

Photorealism is not always the best way to convey a feeling, a sensation or to tell a specific story. In the context of 3D animation movies, stylization is used as both a narrative and a visual tool, but many of these solutions are not open. This work proposes a method for generating stylized contours, sensitive to 3D data, such as lighting, color and object boundaries. The method promotes artistic control over the appearance of contours, in addition to being an open and replicable solution.

Keywords: stylized outline rendering; non-photorealistic rendering (npr); opencv; edge detection.

LISTA DE FIGURAS

Figura 1 – Arte conceitual e exemplo de uma renderização de contorno estilizada . . . 13
Figura 2 – Redução de ruído Gaussiano - a primeira imagem é uma foto única do par de galáxias NGC 3314, com ruído e, em sequência, o resultado da média de 5, 10, 20, 50 e 100 imagens diferentes 16
Figura 3 – *Pixels* cobertos por um triangulo 19
Figura 4 – O processo de *ray tracing* 19
Figura 5 – Desenho de contorno utilizando processamento de imagem - (a) Mapa de profundidade - (b) Bordas do mapa de profundidade - (c) Mapa de normais - (d) Bordas do mapa de normais - (e) Imagens de bordas combinadas - (f) Um caso difícil: pedaço de papel dobrado - (g) Bordas de profundidade 20
Figura 6 – Aplicação do detector de bordas *Canny*. (a) Imagem original. (b) Imagem suavizada. (c) Resultado da aplicação do operador *Sobel*. (d) Resultado da supressão. (e) Resultado do limiar de histeria. (f) Resultado da limiarização utilizando o menor limiar. (g) Resultado de utilizar o maior limiar 22
Figura 7 – Silhueta de superfícies suaves 23
Figura 8 – Fluxo de execução do método Geração de Contornos Estilizados (GCE) . . 25
Figura 9 – Representação de uma sequência 26
Figura 10 – Passo da imagem composta (cor) 26
Figura 11 – Passo de normais 27
Figura 12 – Passo de máscara de objetos (id por objeto) 27
Figura 13 – Passo de profundidade 28
Figura 14 – Passo de alfa 28
Figura 15 – Divisão em regiões para computação paralela 29
Figura 16 – Passo de curvatura de superfície calculado a partir do passo de normal . . . 30
Figura 17 – Detecção de bordas em cada passo utilizando o filtro *Canny* (CANNY, 1986) 33
Figura 18 – Combinação de diferentes passos para atingir resultados visuais específicos 33
Figura 19 – Aplicação de espessura 36
Figura 20 – Suavização de serrilhamento do contorno 36
Figura 21 – Aplicação de cor sólida no contorno 38
Figura 22 – Resultado da recuperação de cor dos objetos 38

Figura 23 – Tonalização dos contornos para melhor integração 40
Figura 24 – Composição do contorno com a renderização original 41
Figura 25 – Interface gráfica implementada para interação com o método GCE 42
Figura 26 – Seleção de diferentes passos para visualização 43
Figura 27 – Opções de detecção de borda 43
Figura 28 – Opções de configuração de linha 44
Figura 29 – Alguns *frames* da animação produzida para testes 45
Figura 30 – Passos gerados pela geração de contornos 45
Figura 31 – Personagem 2 46
Figura 32 – Método GCE aplicado em duas renderizações de coqueiros 47
Figura 33 – Variação de parâmetros de detecção e de borda 48

LISTAGEM DE CÓDIGOS FONTE

Listagem 1 – Função calculate_facing 31
Listagem 2 – Função noise_removal_npfloat32 34
Listagem 3 – Função apply_line_width 35
Listagem 4 – Função anti_aliasing 37
Listagem 5 – Método get_background_color 39

LISTA DE ABREVIATURAS E SIGLAS

Siglas

GCE	Geração de Contornos Estilizados
NPR	Renderização Não Fotorrealista, do inglês *Non-Photorealistic Rendering*
OpenCV	*Open Source Computer Vision Library*
PDI	Processamento Digital de Imagens

SUMÁRIO

1 INTRODUÇÃO 13

1.1 Considerações iniciais 13

1.2 Objetivos 14

1.2.1 Objetivo geral 14

1.2.2 Objetivos específicos 14

1.3 Justificativa 14

2 REFERENCIAL TEÓRICO 16

2.1 Imagens Digitais 16

2.2 Processamento digital de imagens 16

2.2.1 Filtros de convolução 17

2.2.2 Desfoque gaussiano (*gaussian blur*) 17

2.2.3 *Unsharp mask* 17

2.2.4 Dilatação 18

2.2.5 Erosão 18

2.3 Renderização 18

2.3.1 Rasterização 18

2.3.2 *Ray Tracing* 19

2.4 *Non-Photorealistic Rendering* (NPR) 20

2.5 Geração de contornos 20

2.5.1 Técnicas de espaço de imagem 20

2.5.2 Detecção de bordas *Canny* 21

2.5.3 Técnicas de espaço de objeto 23

3 MATERIAIS E METODOLOGIA 24

3.1 Materiais 24

3.2 Metodologia 25

3.2.1 Animação renderizada do 3D 26

3.2.2 Carregamento dos passos de cada frame 26

3.2.3 Divisão em regiões para computação paralela 28

3.2.4 Cálculo de curvatura de superfície usando normais 29

3.2.5 Pré-desfoque dos passos para redução de ruídos 32

3.2.6 Detecção de borda utilizando (filtro *Canny*) 32
3.2.7 Remoção de pequenas regiões conectadas para redução de ruídos 33
3.2.8 Aplicação de espessura da linha utilizando filtros *Dilate* e *Erode* 35
3.2.9 Suavização serrilhamento (Gaussian Blur + Unsharp) 36
3.2.10 Cor sólida 37
3.2.11 Cor da linha respeitando iluminação e sobreposição de objetos 38
3.2.12 Alteração de matiz, saturação e valor da borda (tonalização) 40
3.2.13 Composição da borda com a imagem 40
3.2.14 Junção das regiões processadas em paralelo em uma única imagem 41
3.2.15 Saída de dados 41
4 RESULTADOS 42
4.1 Ferramenta para interação com o método GCE 42
4.2 Testes realizados 44
4.2.1 Animação 44
4.2.2 Testes estáticos 46
4.3 Discussões 48
5 CONCLUSÃO 50
REFERÊNCIAS 51

1 INTRODUÇÃO

Neste capítulo são descritas as considerações iniciais, os objetivos e a justificativa do trabalho apresentado.

1.1 Considerações iniciais

Renderização Não Fotorrealista, do inglês *Non-Photorealistic Rendering* (NPR) é o nome dado para técnicas de geração de imagem que focam na estilização como alternativa à renderização realista (GOOCH; GOOCH, 2001). O fotorrealismo é utilizado em diversas áreas, como efeitos visuais em filmes *live action*, visualização de arquitetura, visualizações científicas, etc. Porém nem sempre simular a realidade fielmente é a melhor maneira de se transmitir um sentimento ao espectador, uma sensação, ou ainda, de se contar uma história.

Estúdios de longa metragem de animação 3D aplicam visuais mais estilizados em suas produções tanto como ferramenta de história, como forma de exploração visual. Esse uso pode ser observada em filmes como *Spider-Man: Into the Spider-Verse* (DIMIAN *et al.*, 2019) e *The Mitchells vs. The Machines* (ARGULA *et al.*, 2021) da *Sony Imageworks*. Porém, as técnicas utilizadas para atingir esses resultados visuais estilizados não são compartilhadas amplamente pelos estúdios como ferramentas ou soluções.

Figura 1 – Arte conceitual e exemplo de uma renderização de contorno estilizada

Fonte: Adaptado de Argula *et al.* (2021).

Uma das técnicas desenvolvidas para o filme *The Mitchells vs. The Machines*, para atingir o visual pretendido para o filme, é a geração de contornos utilizando dados do 3D, como

iluminação, cor, limite de objetos e distância da câmera (LAVENDER; JUDSON; OBRETENOV, 2021), Figura 1.

Este trabalho propõe um método de geração de contornos, sensíveis a dados do 3D como iluminação, cor, textura, limite de objetos. A implementação do método foi testada e validada dentro da produtora de animação Miralumo, que ofereceu suporte fornecendo animações 3D para testes, infraestrutura de hardware, orientação artística e apoio técnico.

1.2 Objetivos

Nas subseções seguintes são apresentados o objetivo geral do trabalho e os objetivos específicos.

1.2.1 Objetivo geral

O objetivo geral deste trabalho é o desenvolvimento de um método e uma ferramenta de geração de contornos estilizados para renderizações 3D que une técnicas de Processamento Digital de Imagens (PDI) com dados do processo de renderização 3D.

1.2.2 Objetivos específicos

Os objetivos específicos elencados para o desenvolvimento deste trabalho foram os seguintes:

- Realizar uma revisão da literatura sobre trabalhos correlatos e conceitos envolvidos;
- Definir técnicas e abordagens para desenvolver o método que gere o resultado visual pretendido;
- Desenvolvimento do método de geração de contornos estilizados;
- Implementação de uma ferramenta para interação com o método;
- Validação dos resultados obtidos.

1.3 Justificativa

O fotorrealismo, apesar de possuir diversas aplicações, nem sempre é a melhor forma de transmitir um sentimento, uma sensação ou de se contar uma história específica.

Estúdios de animação 3D fazem uso de técnicas de estilização tanto como ferramenta narrativa como forma de inovação visual no cinema. Porém, muitas das técnicas de NPR, utilizadas pelos estúdios citados, não são compartilhadas com a indústria em ferramentas e soluções.

A proposta deste trabalho é a criação de um método para geração de contornos estilizados sensíveis às informações do 3D, como iluminação, cor, textura, limite de objetos, que seja replicável e aberta. O projeto contribuições técnicas e atenção à uma área de pesquisa pouco explorada no Brasil, que é a computação gráfica aplicada no contexto de filmes de animação 3D.

2 REFERENCIAL TEÓRICO

Neste capítulo são apresentados os principais assuntos que fundamentam este projeto de pesquisa. Cada sessão apresenta conceitos relevantes para a compreensão da metodologia proposta de geração de contornos apresentada neste trabalho.

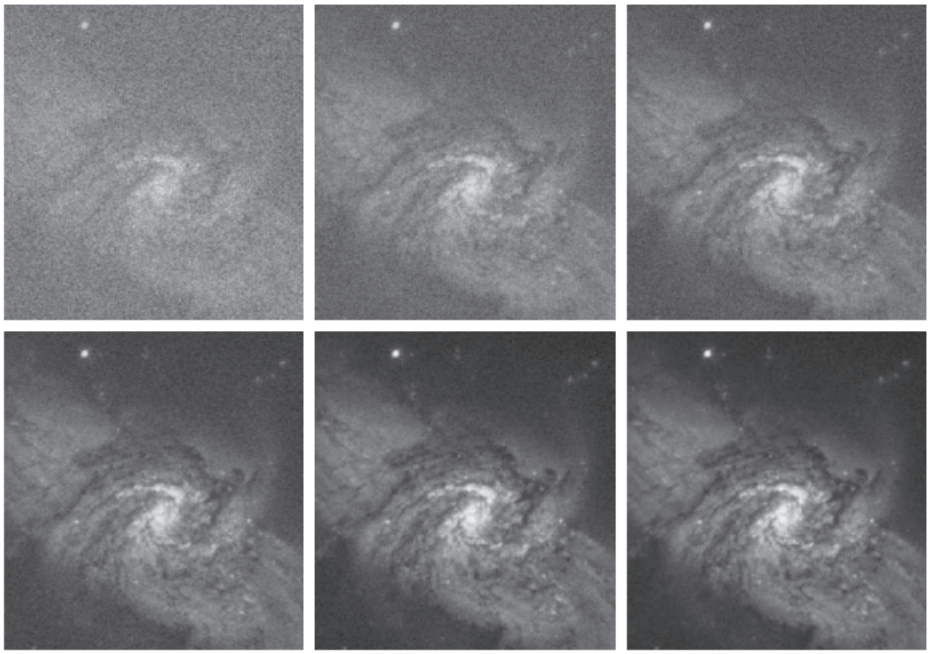
2.1 Imagens Digitais

Segundo Gonzalez e Woods (2018) uma imagem pode ser definida como uma função bidimensional $f(x, y)$, onde x e y são coordenadas espaciais e a amplitude de f para qualquer ponto (x, y) representa a intensidade da imagem no ponto. A imagem é uma imagem digital quando x, y e os valores de f são todos finitos e discretos. Para cada um desses elementos finitos, com posição e valores particulares, é dado o nome de *pixel*.

2.2 Processamento digital de imagens

Segundo Gonzalez e Woods (2018), o processamento digital de imagens engloba processos cujas entradas e saídas são imagens e, em adição, processos que extraem atributos de imagens e até, inclusive, realizam o reconhecimento de objetos individuais.

Figura 2 – Redução de ruído Gaussiano - a primeira imagem é uma foto única do par de galáxias NGC 3314, com ruído e, em sequência, o resultado da média de 5, 10, 20, 50 e 100 imagens diferentes



Fonte: Adaptado de Gonzalez e Woods (2018).

Técnicas de PDI incluem pré-processamento de imagem, como redução de ruído (Figura 2), ajuste de contraste, e aumento de nitidez; aplicação de filtros, como desfoque e detec-

ção de bordas; segmentação de áreas ou objetos; descrição de objetos para posterior computação; e classificação (reconhecimento) de objetos individuais.

As principais técnicas de processamento de imagens utilizadas pela metodologia proposta neste trabalho são descritas a seguir.

2.2.1 Filtros de convolução

Filtros de convolução (DAVIES, 2012) são descritos por um *kernel*, essencialmente uma matriz com valores. O processo de convolução é operado em cada *pixel* da imagem, onde os valores dos *pixels* vizinhos são somados, sendo multiplicados pelo valor de sua posição no *kernel*, e se necessário, a soma final é dividida por um fator de normalização.

2.2.2 Desfoque gaussiano (*gaussian blur*)

O desfoque gaussiano é um filtro de convolução com objetivo de desfoque, cujo *kernel* é gerado por uma função gaussiana (DAVIES, 2012). Assim, os *pixels* mais próximos ao centro possuem maior influência na produção da média, enquanto os *pixels* da borda possuem menor influência.

Um *kernel* 3x3 aproximado de desfoque gaussiano é o seguinte:

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

2.2.3 *Unsharp mask*

Unsharp mask é uma das técnicas mais utilizadas para o aumento de nitidez em imagens (DAVIES, 2012). A técnica trabalha utilizando uma versão ligeiramente desfocada da imagem original que é então subtraída da imagem original para detectar a presença de bordas, criando assim uma máscara. A máscara é utilizada então para aumentar o contraste nas áreas de borda, gerando uma imagem mais nítida. Um limiar pode ser utilizado como controle de quando se aplicar ou não o contraste.

A fórmula básica para a geração da imagem com nitidez é dada pela seguinte operação, onde n é a imagem com nitidez aumentada, o a imagem original, d a imagem desfocada e v é um valor que controla a intensidade da máscara:

$$n = o + (o - d) \times v$$

2.2.4 Dilatação

O processo de dilatação é um processo morfológico, que aumenta a área de características (DAVIES, 2012). É geralmente utilizado para preenchimento de pequenos "buracos"dentro de máscaras.

Funciona de forma parecida com a convolução: para cada pixel, é avaliado se na sua vizinhança, delimitada por um *kernel*, existe um *pixel* válido que coincide com um *pixel* válido do *kernel*. Se esta situação ocorrer, é atribuído um valor para o *pixel* atual, caso contrário, o *pixel* não é alterado.

2.2.5 Erosão

A operação de erosão é um processo morfológico com o objetivo contrário da dilatação, ou seja, ela diminui a área de características (DAVIES, 2012). É geralmente utilizada para remover ruídos na forma de pontos menores do que o *kernel* utilizado.

Para cada pixel, é avaliado se na sua vizinhança, delimitada por um *kernel*, todos os *pixels* válidos do *kernel* coincidem com pixels válidos da imagem. Se esta situação ocorrer, o *pixel* não é alterado, caso contrário, é atribuído zero para o *pixel*.

2.3 Renderização

Em computação gráfica, renderização é o processo pelo qual uma cena virtual é convertida em uma imagem (GOOCH; GOOCH, 2001). Dentre as principais formas de renderização podemos citar a rasterização e o *ray tracing*, descritos nas subseções a seguir.

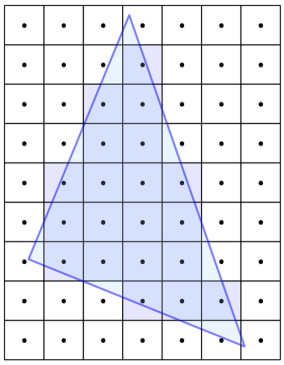
2.3.1 Rasterização

A rasterização consiste no processo de converter uma geometria contínua em sua representação discreta, formada por *pixels*, a partir da visão de uma câmera virtual (HUGHES *et al.*, 2013).

O processo de rasterização cria uma espécie de impressão da geometria no plano de visualização (Figura 3) e, a princípio, não carrega informações de textura, iluminação, ou de qualquer efeito ótico. Essas características secundárias são descritas geralmente por *shaders*, que são passos posteriores de processamento, que consideram, por exemplo, posição de luzes e objetos para simular efeitos.

É a técnica geralmente utilizada em aplicações de tempo real devido seu menor custo computacional e presença de hardware dedicado nas placas de processamento gráfico.

Figura 3 – *Pixels* cobertos por um triangulo

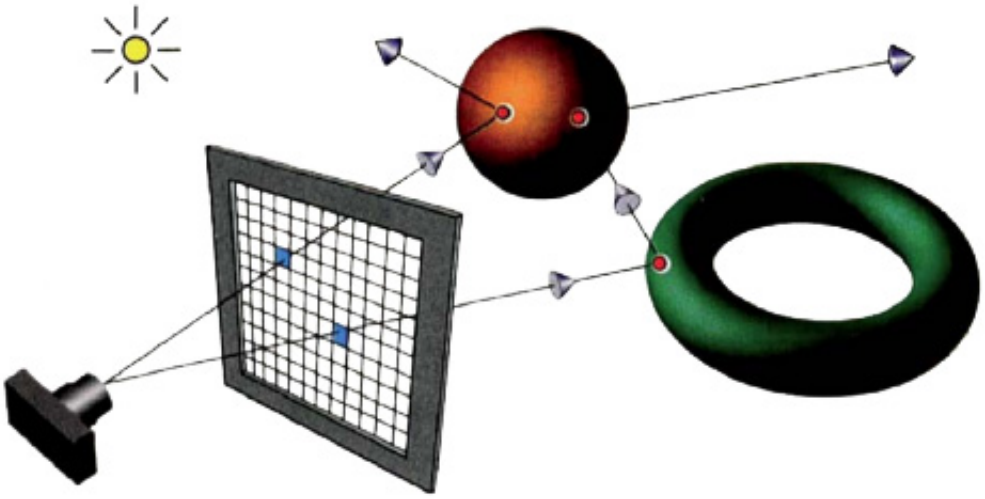


Fonte: Kraus (2011).

2.3.2 *Ray Tracing*

Ray Tracing é uma técnica de computação gráfica que cria imagens lançando raios (SUFFERN, 2007). Os raios são lançados a partir de uma câmera, passam através de *pixels* e são testados se colidem com objetos. Quando o raio atinge um objeto o renderizador verifica quanta luz é transmitida ao longo do raio e se ele sofreu reflexão, alterações do material do objeto, entre outras características, para determinar a cor do pixel, Figura 4.

Figura 4 – O processo de *ray tracing*



Fonte: Adaptado de Suffern (2007).

A ideia central do *ray tracing* é a simulação física da luz, e por esse motivo seu uso é muito relacionado com a geração de imagens fotorrealistas. É uma técnica capaz de simular diversos efeitos óticos com realismo, como sombras suaves, oclusão, reflexão, refração, profundidade de campo, *scattering*, *caustics* etc. A principal desvantagem do *ray tracing* em relação à rasterização é o custo de performance muito mais alto.

2.4 *Non-Photorealistic Rendering* (NPR)

A renderização não fotorrealista, do inglês *non-photorealistic rendering* (NPR), se refere às técnicas de geração de imagens digitais que não têm como foco o realismo (GOOCH; GOOCH, 2001).

O fotorrealismo é utilizado em diversas áreas, como efeitos visuais em filmes *live action*, visualização de arquitetura, visualizações científicas, etc. Porém nem sempre simular a realidade fielmente é a melhor maneira de se transmitir um sentimento, uma sensação, ao espectador, ou ainda, de se contar uma história.

Entre as principais técnicas de NPR podemos citar a extração de contornos e silhuetas, renderização com estilo de caneta e tinta, renderização com estilo de pintura e renderização para ilustrações técnicas (GOOCH; GOOCH, 2001).

2.5 Geração de contornos

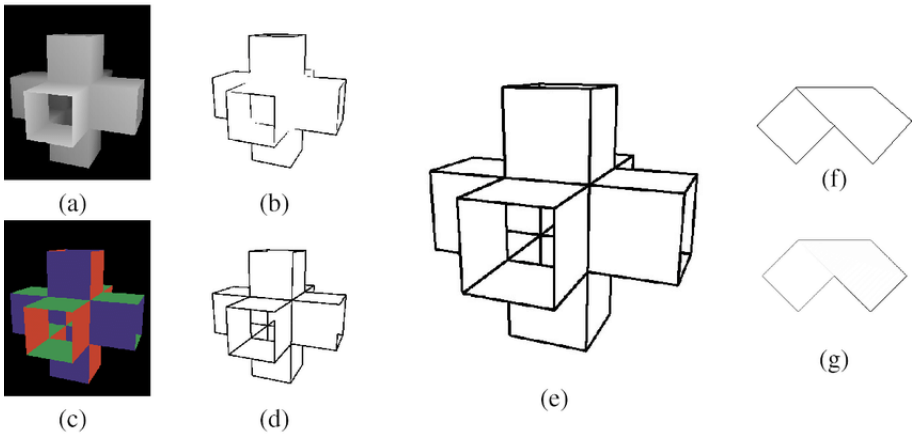
Uma das principais técnicas de NPR é a extração de contornos e silhuetas. Segundo (GOOCH; GOOCH, 2001) silhuetas são ótimas ferramentas para se representar formas, e seres humanos são ótimos em detectar formas a partir de linhas.

As técnicas para se gerar contornos podem ser divididas entre *técnicas de espaço de imagem* e *técnicas de espaço de objeto*. Estas técnicas são descritas nas subseções seguintes.

2.5.1 Técnicas de espaço de imagem

As técnicas que trabalham com o espaço de imagem pertencem a área de Processamento de Imagens, e possuem como entrada uma imagem já renderizada da cena.

Figura 5 – Desenho de contorno utilizando processamento de imagem - (a) Mapa de profundidade - (b) Bordas do mapa de profundidade - (c) Mapa de normais - (d) Bordas do mapa de normais - (e) Imagens de bordas combinadas - (f) Um caso difícil: pedaço de papel dobrado - (g) Bordas de profundidade



Fonte: Adaptado de Gooch e Gooch (2001).

Os contornos são extraídos através de filtros e algoritmos de detecção de bordas, como o citado na subseção 2.5.2. Um dos principais problemas dessas técnicas é que as bordas não costumam representar a silhueta do objeto, superfícies com muita informação de textura são problemáticas e bordas não são detectadas quando objetos de mesma cor se sobrepõem (GOOCH; GOOCH, 2001). A Figura 5 representa algumas técnicas de geração de contornos em espaço de imagem.

2.5.2 Detecção de bordas *Canny*

A detecção de bordas *Canny* (CANNY, 1986) é um algoritmo de detecção de bordas multiestágio em espaço de imagem consolidado, que detecta uma grande variedade de bordas e possui controle de limiares para supressão de falsos positivos.

Os estágios são os seguintes:

1. Redução de ruído: é utilizado um filtro de suavização gaussiano 5x5, pois a detecção de bordas é suscetível a ruídos.

2. Gradiente de intensidade da imagem: A imagem suavizada é então filtrada com um *kernel Sobel* na direção horizontal e vertical para obter a primeira derivada na direção horizontal e na direção vertical. A partir dessas duas imagens, é encontrado o gradiente e a direção da borda (ângulo).

3. Supressão de não máximos: depois de obter a magnitude e a direção do gradiente, é feita uma varredura completa da imagem para remover quaisquer *pixels* indesejados que possam não constituir uma borda. Para isso cada pixel é verificado se é um máximo local em sua vizinhança na direção do gradiente.

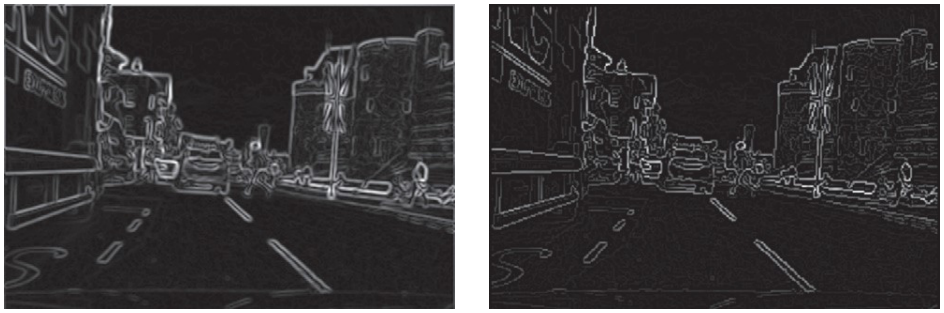
4. Limiar de histerese: neste estágio é decidido quais *pixels* são realmente bordas. Para isso, são utilizados dois valores de limiar, um mínimo e máximo. Os *pixels* com gradiente de intensidade maior que o máximo são certamente consideradas bordas e aqueles abaixo do mínimo são diretamente descartados. Os *pixels* que estão entre os dois limiares são selecionados como bordas ou não com base em sua conectividade. Se eles estiverem conectados a "bordas seguras", eles são considerados parte das bordas. Caso contrário, eles também são descartados.

Na Figura 6 é mostrado um exemplo passo a passo da execução do algoritmo.

Figura 6 – Aplicação do detector de bordas *Canny*. (a) Imagem original. (b) Imagem suavizada. (c) Resultado da aplicação do operador *Sobel*. (d) Resultado da supressão. (e) Resultado do limiar de histeria. (f) Resultado da limiarização utilizando o menor limiar. (g) Resultado de utilizar o maior limiar

(a) **(b)**

(c) **(d)**



(e) **(f)**

(g)

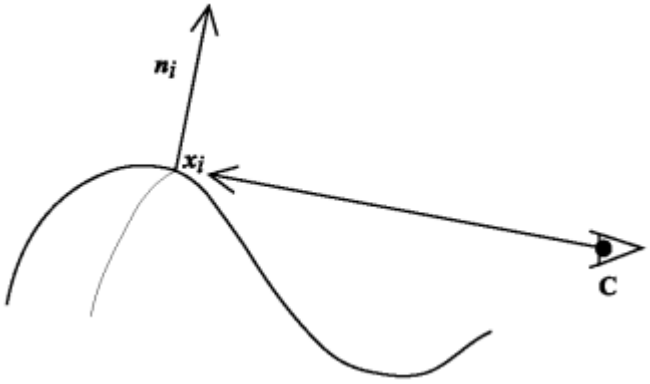
Fonte: Adaptado de Davies (2012).

2.5.3 Técnicas de espaço de objeto

As Técnicas de espaço de objeto trabalham diretamente com os modelos em 3D e estão ligadas ao processo de renderização da cena, podendo produzir curvas com mais precisão.

Segundo Gooch e Gooch (2001), uma das abordagens para se detectar silhuetas no espaço de objeto para malhas facetadas é detectar quais são as linhas que conectam polígonos voltados para trás, em relação à visualização, com polígonos voltados para frente.

Figura 7 – Silhueta de superfícies suaves



Fonte: Adaptado de Gooch e Gooch (2001).

Já para malhas suaves, geradas por *Non-Uniform Rational Basis Spline* (NURBS) ou subdivisão de superfície, a silhueta pode ser definida como os pontos de superfície x_i cuja normal n_i é perpendicular ao vetor de visualização de origem C (Figura 7):

$$n_i \times (x_i - C) = 0$$

3 MATERIAIS E METODOLOGIA

Neste capítulo são descritos os materiais utilizados e a metodologia criada para alcançar os resultados descritos no Capítulo 4.

3.1 Materiais

A solução foi implementada utilizando a linguagem de programação *Python* (PYTHON SOFTWARE FOUNDATION, 2022) em conjunto com a biblioteca de processamento de imagens *Open Source Computer Vision Library* (OpenCV) (OPEN SOURCE VISION FOUNDATION, 2022). A decisão foi motivada por facilitar tanto o ciclo de testes da metodologia quanto a replicabilidade da solução por outras pesquisas.

A linguagem *Python* (PYTHON SOFTWARE FOUNDATION, 2022) fornece sintaxe simplificada, facilidade de implementação e maior fluxo de testes quando comparada com linguagens de mais baixo nível, porém ao custo de tempo de execução mais lento. Nesta solução a linguagem funciona principalmente como gerenciadora do fluxo de dados entre as funções de bibliotecas pré-compiladas em C/C++, o que aumenta o desempenho de execução. Excetuando-se em 3 procedimentos onde as bibliotecas não ofereciam ferramentas suficientes e foi necessário a implementação de algoritmos de processamento de imagem em *Python* (PYTHON SOFTWARE FOUNDATION, 2022), ineficientes em tempo de execução quando comparados com sua possível implementação em C/C++, porém com impacto atenuado pela implementação de execução concorrente com a biblioteca *Multiprocessing* integrada ao *Python* (PYTHON SOFTWARE FOUNDATION, 2022).

A biblioteca de processamento de imagens *OpenCV* (OPEN SOURCE VISION FOUNDATION, 2022), aliada com a biblioteca de manipulação matricial *NumPy* (NUMFOCUS FOUNDATION, 2022), fornecem uma grande gama de ferramentas para manipulação de dados, aplicação de funções matemáticas sobre matrizes multidimensionais e algoritmos de processamento de imagem pré-compilados em C e C++. Por serem bibliotecas pré-compiladas, seu uso, onde é possível, contorna o problema de desempenho gerado pelo interpretador de *Python* (PYTHON SOFTWARE FOUNDATION, 2022).

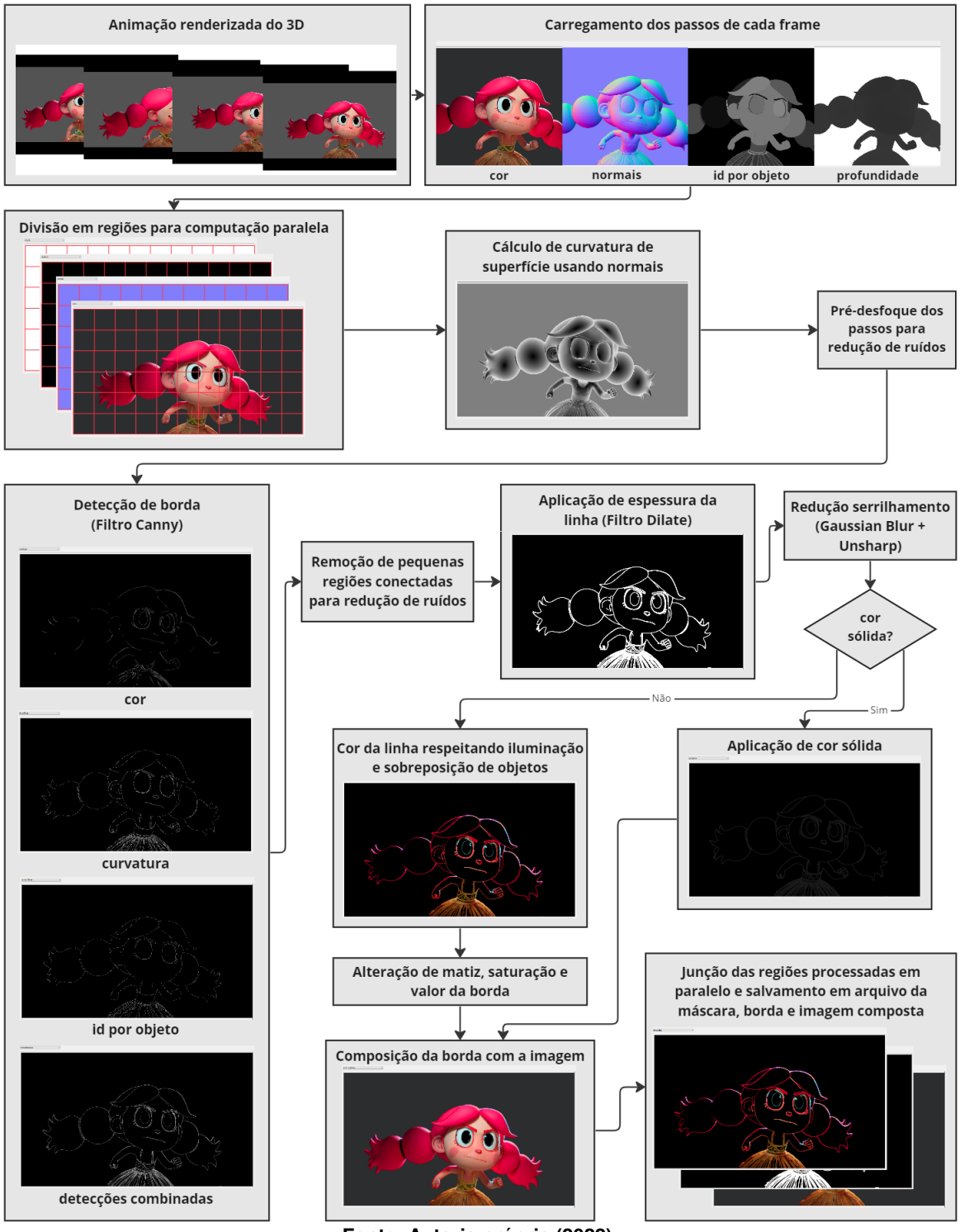
Outra biblioteca relevante para o ciclo de testes e uso da solução foi a biblioteca de interface gráfica *PySide2* (QT GROUP, 2022), utilizada para a visualização e interação com a solução. Porém seu uso não é descrito neste trabalho por não ser o foco da pesquisa.

O renderizador utilizado para a renderização da animação utilizada como teste foi o *VRay* for *Maya* (CHAOS SOFTWARE EOOD, 2022), porém a solução é agnóstica de renderizador e é facilmente adaptada para qualquer renderizador com saída em passos.

3.2 Metodologia

Para a obtenção do resultado pretendido foi desenvolvido o Método Geração de Contornos Estilizados (GCE) descrito nesta seção. Um resumo da execução do método é apresentado no fluxograma na Figura 8, e cada passo na imagem é descrito em sequência.

Figura 8 – Fluxo de execução do método GCE

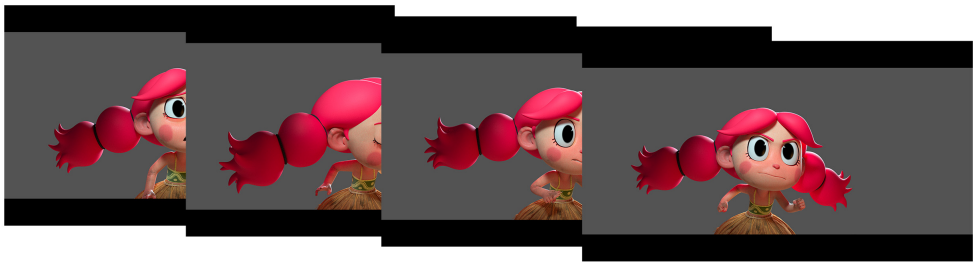


Fonte: Autoria própria (2022).

3.2.1 Animação renderizada do 3D

O método GCE é pensado para aplicação em sequências de imagens (animações), porém também pode ser utilizada para geração de uma única imagem (*frame*).

Figura 9 – Representação de uma sequência



Fonte: Autoria própria (2022).

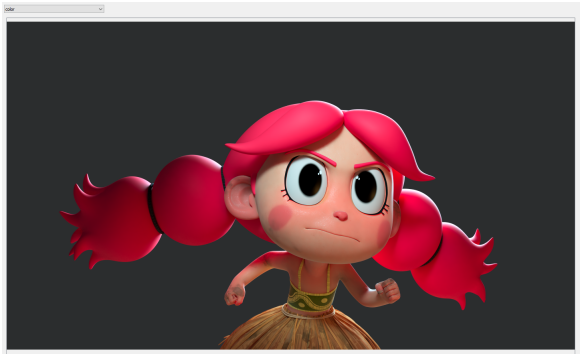
A sequência (Figura 9) deve ser renderizada com separação de alguns passos específicos para serem utilizados no método: imagem final (cor), normais, mascara de objetos (id por objeto), e profundidade. O uso dos passos é descrito na subseção 3.2.2.

3.2.2 Carregamento dos passos de cada frame

Para esta implementação utilizando *OpenCV* (OPEN SOURCE VISION FOUNDATION, 2022), cada *frame* foi renderizado com passos separados em imagens *png* com profundidade de 16 bits por canal. Cada imagem é normalizada com valores de 0 a 1, com tipo *np.float32*, da biblioteca *NumPy* (NUMFOCUS FOUNDATION, 2022). A normalização simplifica as manipulações sobre as imagens.

Quatro passos são necessários para a solução: imagem final (cor), normais, máscara de objetos (id por objeto), e profundidade. Um passo de uma máscara de alfa opcional pode ser utilizado para isolar objetos.

Figura 10 – Passo da imagem composta (cor)

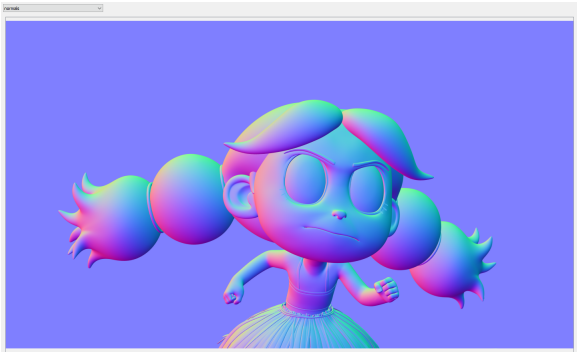


Fonte: Autoria própria (2022).

- Imagem final (cor): é o passo com a renderização completa em uma única imagem (Figura 10). Este passo pode ser utilizado para detecção de bordas e influencia prin-

cipalmente para detecção de bordas dentro de texturas, porém é influenciado pela iluminação da imagem, criando linhas em áreas de transição de luz e sombra. Também é utilizado para a definição da cor e iluminação dos contornos quando essa opção é utilizada.

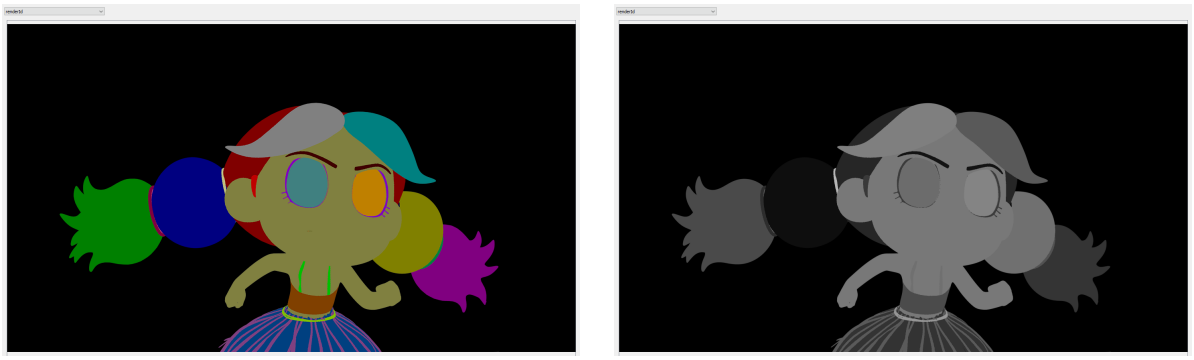
Figura 11 – Passo de normais



Fonte: Autoria própria (2022).

- Normais: é um passo padrão em renderizadores que descreve em uma imagem os vetores normais em relação a superfície dos objetos (Figura 11). Os 3 canais R, G, B, são utilizados para guardar as coordenadas X, Y, Z de cada vetor. Valores negativos são normalizados dentro do intervalo de valores dos canais da imagem, por exemplo: em uma imagem de 8 bits, os valores de 0 a 127 representam coordenadas de -1 a 0 e valores de 128 a 255 coordenadas de 0 a 1. Este passo não é utilizado diretamente para detecção de bordas, mas é utilizado para o cálculo de um passo de curvatura de superfície dos objetos, descrito na subseção 3.2.4, que é utilizado para detecção de bordas.

Figura 12 – Passo de máscara de objetos (id por objeto)
(a) *renderId* **(b) *renderId* em escala de cinza e *medianBlur***



Fonte: Autoria própria (2022).

- Máscara de objetos (id por objeto): também comum em renderizadores, esse passo cria uma máscara com uma cor para cada geometria independente (objeto) na cena 3D (Figura 12). Esse passo é utilizado para detecção de bordas e contribui com bordas entre os limites dos objetos, sendo a principal fonte de bordas da solução. Esse

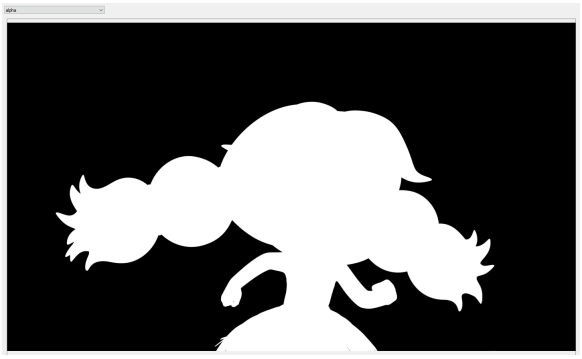
passo já é lido em escala de cinza, pois as informações *RGB* não são necessárias em nenhum momento. No caso do passo de *renderId* gerado pelo *VRay* (CHAOS SOFTWARE EOOD, 2022), é aplicado um filtro de *medianBlur* com *kernel* tamanho 3 para remover ruído do tipo sal e pimenta causado pelas bordas duras.

Figura 13 – Passo de profundidade

Fonte: Autoria própria (2022).

- Profundidade: geralmente nomeado *zDepth* nos renderizadores, esse passo traz informação da profundidade da superfície do objeto na cena (Figura 13). Essa informação é utilizada para recuperar a cor do objeto mantendo sobreposições, buscando pela cor do objeto mais próximo da visão da câmera, descrito na subseção 3.2.11.

Figura 14 – Passo de alfa



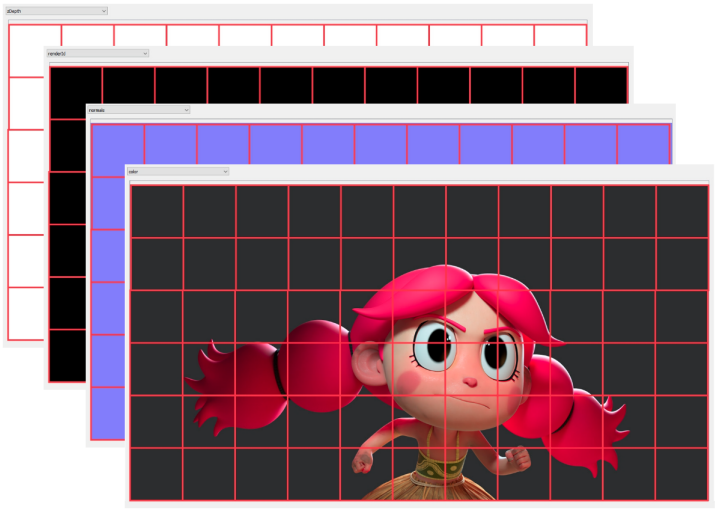
Fonte: Autoria própria (2022).

- Máscara de alfa (opcional): pode ser utilizada para isolar objetos para a geração de contornos, por exemplo, separar personagens e fundo (Figura 14).

3.2.3 Divisão em regiões para computação paralela

Cada imagem é dividida em sub-regiões (Figura 15) e armazenadas em um dicionário, este dicionário é adicionado a outro dicionário contendo informações de tamanho da região, posição na imagem original, identificação da região e um objeto com parâmetros dos contornos. Este objeto com parâmetros pode ser global, utilizando o padrão de projeto *singleton*, cada

Figura 15 – Divisão em regiões para computação paralela



Fonte: Autoria própria (2022).

frame pode ter sua própria configuração de contorno, ou ainda, cada sub-região. O dicionário representando a sub-região é então adicionado a uma fila concorrente de renderização, acessível por múltiplos processos.

A partir deste ponto, as operações são realizadas em subprocessos chamados **buckets**. Cada *bucket* funciona como um consumidor, esperando por novos dicionários adicionados à fila de renderização concorrente. Após o termino do processamento, o dicionário representando a sub-região, com a saída da renderização, é adicionado à fila concorrente de saída e o *bucket* volta a aguardar novos dicionários na fila de renderização. O subprocesso é finalizado quando recebe um sinal de parada na fila de renderização.

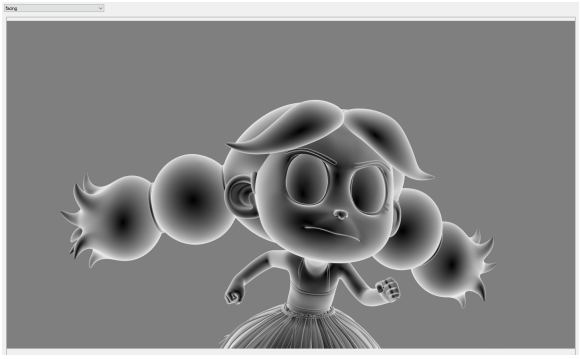
Os *buckets* são gerenciados por um objeto *singleton BucketRenderer*, que cria e mantém comunicação com os subprocessos, é responsável por encerrar os processos de forma segura e mantém referências das filas de entrada e saída de dados.

3.2.4 Cálculo de curvatura de superfície usando normais

Já em cada *bucket*, a sub-região do passo de normal é utilizado para calcular a curvatura da superfície dos objetos. Como o passo de normal representa o vetor normal da superfície dos objetos em cada ponto, é possível calcular o ângulo da superfície em relação a visão da câmera, atribuindo preto para superfícies voltadas para a câmera e branco para superfícies voltadas 90º em relação a visão da câmera.

Este passo de curvatura é utilizado para detecção de bordas e contribui com bordas relacionadas a mudanças de curvatura na superfície de um mesmo objeto, como dobras e protuberâncias, exemplo são dobras dos braços e orelhas, e o nariz de personagens, respectivamente.

Figura 16 – Passo de curvatura de superfície calculado a partir do passo de normal



Fonte: Autoria própria (2022).

Como a imagem renderizada é sempre perpendicular a visão da câmera que a renderizou, temos que o vetor de visualização da câmera é sempre $\vec{v} = (0, 0, 1)$. Logo, tendo dois vetores, o vetor normal da superfície $\vec{n}$ e o vetor de visualização $\vec{v}$, podemos encontrar o ângulo θ entre eles através da fórmula:

$$\theta = \arccos \frac{\vec{n} \cdot \vec{v}}{\|\vec{n}\|\|\vec{v}\|}$$

Como $\vec{n}$ e $\vec{v}$ são vetores unitários, temos:

$$\theta = \arccos \vec{n} \cdot \vec{v}$$

E, como queremos uma imagem normalizada entre 0 e 1, representando valores entre preto e branco, podemos dividir o resultado por $\frac{\pi}{2}$ para obter o valor do pixel final. Utilizamos o valor absoluto do produto vetorial, pois, apesar de inesperado, erros na orientação de normais nas malhas 3D podem acabar produzindo valores negativos:

$$pixel = \frac{2 \arccos |\vec{n} \cdot \vec{v}|}{\pi}$$

A Listagem 1 mostra o código para se calcular a curvatura a partir do passo de normal utilizando funções da biblioteca *NumPy* (NUMFOCUS FOUNDATION, 2022). Na *linha 12* é definido o vetor de visualização, com coordenadas ZYX, consequência da ordem de trabalho da biblioteca *OpenCV*(OPEN SOURCE VISION FOUNDATION, 2022) que é *BGR* e não *RGB*. A *linha 13* transforma o vetor de 3 posições em uma matriz multidimensional com tamanho 1x1, 3 dimensões, e tipo de dados *float32*, como se fosse uma imagem *RGB* de um pixel.

Precisamos calcular o produto escalar entre o vetor de visualização e cada pixel da imagem de normais. Como a biblioteca *NumPy* (NUMFOCUS FOUNDATION, 2022) não oferece uma função que calcula o produto escalar entre cada valor de duas matrizes multidimensionais, precisamos iterar cada pixel da imagem utilizando o interpretador *Python* (PYTHON SOFTWARE FOUNDATION, 2022), o que é lento, porém é minimizado pela implementação concorrente. Uma solução eficiente seria implementar uma pequena biblioteca com essa função em

Listagem 1 – Função calculate_facing

```
1  def calculate_facing(normals):
2      """
3      Calculates a normalized facing image from normal pass.
4      Black when towards the camera and white when away from camera.
5
6      Args:
7          normals: normals pass image.
8
9      Returns:
10         The facing image.
11     """
12     view_vector = np.array((1.0, 0.0, 0.0))  # cv2 BGR model ZYX
13     view_vector = view_vector.reshape((1, 1, 3)).astype(np.float32)
14
15     def image_dot_product(img_a, img_b):
16         return np.abs(np.dot(img_a, img_b))
17
18     im_dot = np.vectorize(image_dot_product, [np.float32])
19     cos = im_dot(normals * 2 - 1, view_vector)[:, :, 0]
20     arccos = np.arccos(cos)
21     facing_img = arccos * 2 / np.pi
22
23     return facing_img
```

Fonte: Autoria própria (2022).

C/C++ pré-compilada, utilizando a biblioteca *OpenCV* (OPEN SOURCE VISION FOUNDATION, 2022) para C++, e utilizá-la dentro do código *Python* (PYTHON SOFTWARE FOUNDATION, 2022).

Esta solução iterativa é gerada através da função *vectorize*, da biblioteca *NumPy* (NUMFOCUS FOUNDATION, 2022), que opera como um *for loop* aninhado sobre as matrizes de entrada, aplicando a função atribuída em sua definição para cada valor na matriz. O custo computacional é o mesmo de se utilizar *for loops* aninhados em *Python* (PYTHON SOFTWARE FOUNDATION, 2022), porém sua sintaxe é mais organizada.

Nas linhas *15* e *16* é definida a função *image_dot_product* que visualmente aplicaria o produto escalar (*np.dot*) entre duas imagens *img_a* e *img_b* e retornaria o valor absoluto dos resultados obtidos. Na *linha 18* é aplicado a função *vectorize* que faz com que a função *image_dot_product* seja aplicada pixel a pixel entre as imagens de entrada, com retorno definido como *float32*.

A *linha 19* gera uma matriz unidimensional com o resultado do produto escalar entre cada vetor da imagem de normais e o vetor de visualização. O trecho $normals * 2 - 1$ desfaz a transformação descrita no item de **normais** na subseção 3.2.2, retornando o valor de coordenadas para o intervalo de -1 a 1. O trecho $[:, :, 0]$ retorna somente a primeira dimensão da matriz de saída, que é a que possui os escalares resultantes do produto escalar.

A *linha 20* aplica a função arco cosseno para cada valor de cosseno adquirido como resultado do produto escalar.

Na *linha 21* a matriz com os valores dos ângulos é normalizada entre os valores 0 e 1, funcionando como uma imagem em escala de cinza.

A *linha 23* retorna a imagem que descreve a curvatura dos objetos no *frame*.

3.2.5 Pré-desfoque dos passos para redução de ruídos

É possível que em cada passo utilizado para detecção de bordas seja aplicado um filtro de *Gaussian Blur* para redução de detalhes e consequente redução de ruídos (DAVIES, 2012) na detecção de bordas. Esse filtro é especialmente útil para o passo de cor, que costuma possuir bastantes informações de textura que nem sempre são desejáveis na detecção de borda.

3.2.6 Detecção de borda utilizando (filtro *Canny*)

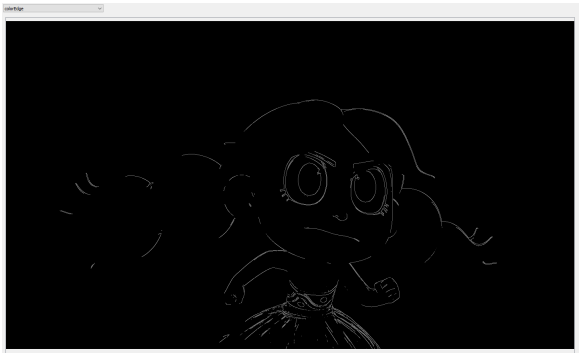
Nesta etapa é utilizado o filtro de detecção de bordas *Canny* (CANNY, 1986) nos passos indicados para detecção. O uso em cada passo é facultativo, dependendo do tipo de bordas que se deseja obter. O filtro *Canny* (CANNY, 1986) foi escolhido pois possui uma abordagem multi-passos com redução de ruído, variedade de detecção de bordas em diferentes orientações e uso de limiares (limitação de valores entre intervalos) que permitem um controle mais fino da detecção. Os dois limiares utilizados pelo filtro podem ser indicados independentemente para cada passo.

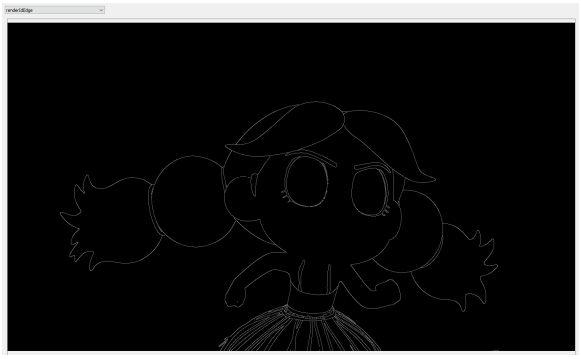
A Figura 17 mostra as bordas obtidas pelo filtro *Canny* (CANNY, 1986) em cada um dos três passos aplicáveis, respectivamente os passos de cor, id por objeto e curvatura. É possível observar que o passo de **cor** traz informações de texturas aplicadas aos objetos. O passo de **id por objeto** traz informações de limites entre malhas (objetos) diferentes; e o passo de **curvatura** traz informações relacionadas a mudanças de curvatura dentro de uma mesma malha.

Como cada passo traz contribuições diferentes para o resultado visual, é possível escolher diferentes combinações de passos, como mostrado na Figura 18. A Figura 18 (a) mostra a combinação dos passos de id por objeto e curvatura, trazendo informações de silhuetas e dobras mas não de texturas. A Figura 18 (b) possui os três passos combinados e nela é possível observar a contribuição de informações sobre textura, como na região dos olhos e no interior da blusa, o passo de cor, porém, pode gerar bordas em regiões de transição de luz e sombra, o que nem sempre é desejado.

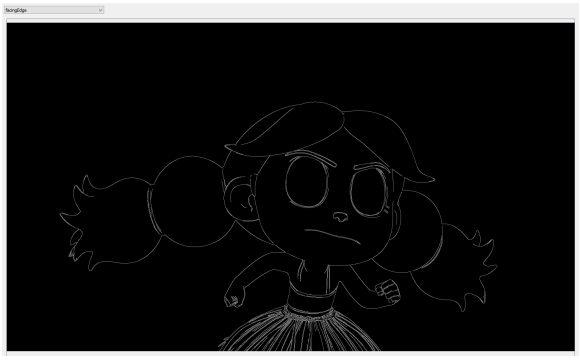
Figura 17 – Detecção de bordas em cada passo utilizando o filtro *Canny* (CANNY, 1986)

(a) cor **(b) id por objeto**





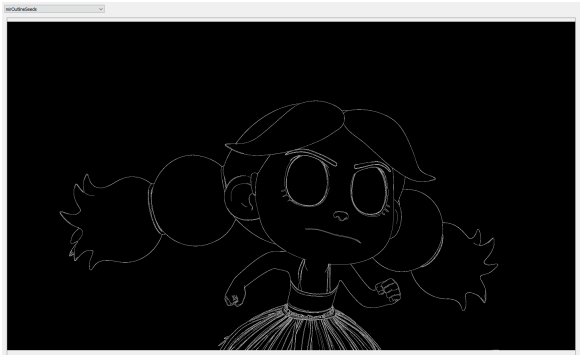
(c) curvatura

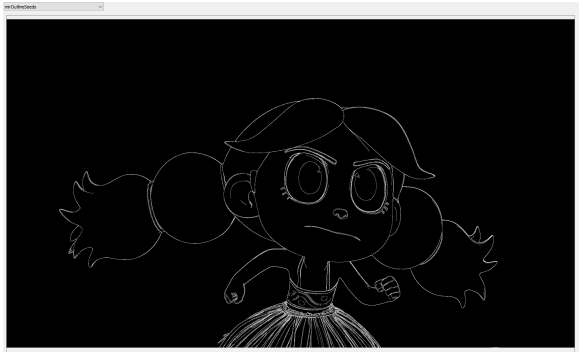


Fonte: Autoria própria (2022).

Figura 18 – Combinação de diferentes passos para atingir resultados visuais específicos

(a) id por objeto + curvatura **(b) cor + id por objeto + curvatura**





Fonte: Autoria própria (2022).

3.2.7 Remoção de pequenas regiões conectadas para redução de ruídos

Para cada passo é possível indicar um limiar com o tamanho mínimo para regiões conectadas. É utilizada a função *connectedComponentsWithStats* da biblioteca *OpenCV* (OPEN SOURCE VISION FOUNDATION, 2022) para a obtenção de informações sobre regiões conectadas na imagem binarizada, neste caso máscara e tamanho das regiões.

É realizada uma iteração sobre as informações das regiões e se a região for maior do que o limiar informado, os índices de sua máscara são utilizados para adicionar a região em

uma imagem preenchida com zeros. A função que performa esta operação está descrita na Listagem 2.

Listagem 2 – Função noise_removal_npfloat32

```
1  def noise_removal_npfloat32(image, max_size: int):
2      """
3      Performs noise removal based on connected components size.
4
5      Args:
6          image: the image to perform.
7          max_size: min number of pixels to consider non noise.
8
9      Returns:
10         The binary image in float32.
11     """
12     if image.dtype != 'float32':
13         return
14
15     src_img = (image * 255).astype(np.uint8)
16     num_areas, im_masks, stats, _ = cv2.connectedComponentsWithStats(src_img)
17
18     # get sizes info only
19     sizes = stats[:, -1]
20     # remove the background and update total num_areas
21     sizes = sizes[1:]
22     num_areas -= 1
23
24     out_image = np.zeros((image.shape[0], image.shape[1], 1), np.float32)
25     for blob in range(num_areas):
26         if sizes[blob] >= max_size:
27             out_image[im_masks == blob + 1] = 1
28     return out_image
```

Fonte: Autoria própria (2022).

Na *linha 15* a imagem é convertida para 8 bits, pois a função *connectedComponentsWithStats* só trabalha com este tipo de dado. Na *linha 16* a função *connectedComponentsWithStats* é aplicada sob a imagem binarizada e retorna, respectivamente, o número de regiões conectadas, uma imagem com máscaras para cada região, onde o valor do pixel é o índice da região, um objeto com estatísticas sobre as regiões e centroides das regiões.

A *linha 19* retorna em uma lista somente a informação de tamanho de cada região. Na *linha 20* o primeiro elemento, correspondente ao fundo, é removido da lista, e na *linha 21* o número de regiões é atualizado.

Na *linha 24* uma imagem de tipo *float32*, do tamanho da imagem original e preenchida com zeros é criada. O *loop for* nas *linhas 25-27* itera sobre as áreas, e se a área não é menor do que o limiar informado, sua máscara é utilizada para adicionar a cor branca na imagem de saída nas coordenadas correspondentes à região. Desta forma, regiões menores do que o limiar não são adicionadas a saída e esse tipo de ruído é eliminado.

3.2.8 Aplicação de espessura da linha utilizando filtros *Dilate* e *Erode*

O controle de espessura da borda é realizado com um filtro de dilatação, para aumentar a espessura (GONZALEZ; WOODS, 2018), ou de erosão, para diminuí-la (GONZALEZ; WOODS, 2018). A espessura é definida pelo tamanho do *kernel* de convolução utilizado. Nesta solução é utilizado um *kernel* circular, pois este gera bordas mais arredondadas. A Listagem 3 implementa esta funcionalidade.

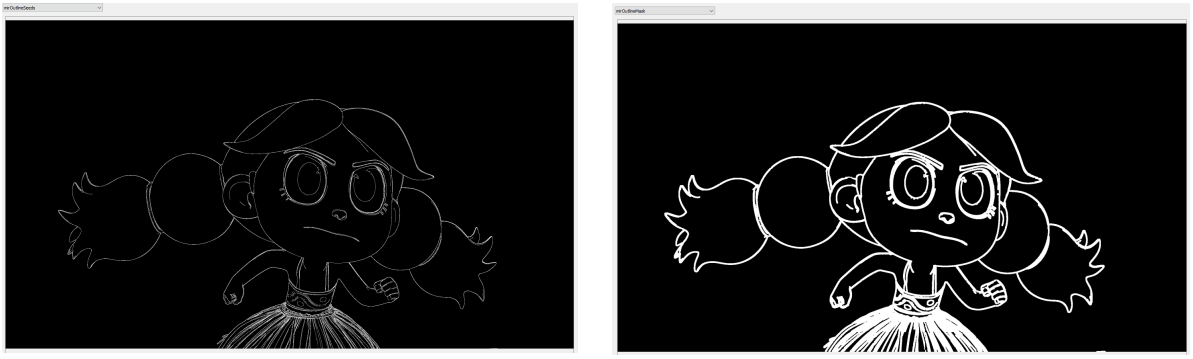
Listagem 3 – Função apply_line_width

```
1  def apply_line_width(image, line_width, kernel_format=cv2.MORPH_ELLIPSE,
2                       iterations=1):
3      """
4      Controls line width in binary images performing erode, if the line width
5      is negative, and dilate otherwise.
6
7      Args:
8          image: the input image.
9          line_width: the desired amount of line width, negative values decre-
10                     ase the width.
11         kernel_format: the cv2.MORPH format.
12         iterations: number of iterations for dilate/erode operations.
13
14     Returns:
15         The processed image.
16     """
17     if line_width == 0:
18         return image
19
20     abs_line_width = abs(line_width)
21     kernel = cv2.getStructuringElement(kernel_format,
22                                        (abs_line_width, abs_line_width))
23
24     if line_width > 0:
25         return cv2.dilate(image, kernel, iterations=iterations)
26     else:
27         return cv2.erode(image, kernel, iterations=iterations)
```

Fonte: Autoria própria (2022).

Na Listagem 3, uma espessura (*line_width*) negativa indica quando será aplicada erosão utilizando a função *erode* da biblioteca *OpenCV* (OPEN SOURCE VISION FOUNDATION, 2022), do contrário é aplicada uma dilatação, utilizando a função *dilate*. O valor absoluto do parâmetro *line_width* é utilizado para a criação do *kernel* que será utilizado pelos filtros, este *kernel* é gerado pela função *getStructuringElement* da biblioteca *OpenCV* (OPEN SOURCE VISION FOUNDATION, 2022), que recebe como parâmetro um dos formatos de *kernel* pré-definidos pela biblioteca e o tamanho do *kernel*, respectivamente.

Figura 19 – Aplicação de espessura
(a) Resultado das detecções de borda **(b) Resultado da aplicação de espessura**



Fonte: Autoria própria (2022).

Na Figura 19 (a) é mostrado o resultado da detecção de bordas, descrito da subseção 3.2.6, e na Figura 19 (b) é mostrado o resultado do processo de dilatação.

3.2.9 Suavização serrilhamento (Gaussian Blur + Unsharp)

Tanto o filtro de detecção de bordas quanto a aplicação de espessura geram bordas serrilhadas nos contornos, como pode ser observado na Figura 20 (a). Este passo é pensado para suavizar esse serrilhamento, e é útil principalmente para imagens de baixa resolução, Figura 20 (b).

Figura 20 – Suavização de serrilhamento do contorno
(a) Contornos sem suavização **(b) Contornos com suavização**

Fonte: Autoria própria (2022).

Nos testes realizados, utilizar imagens com resolução 4K, ou acima, praticamente remove a necessidade deste passo e ainda reduz a quantidade de tremulações na detecção de bordas, porém em imagens com resolução inferiores esta técnica se faz necessária.

O processo utilizado consiste em aplicar um filtro de *Gaussian Blur* (DAVIES, 2012), para suavizar as bordas, seguido de uma técnica de *unsharp masking* (DAVIES, 2012), para recuperar a nitidez mantendo a suavização. Na Listagem 4 é apresentada a função responsável pelo processo de suavização de serrilhado.

Listagem 4 – Função anti_aliasing

```
1  def anti_aliasing(image, blur_kernel=7, sharpen_kernel=9, sharpen_sigma=2.0,
2                    sharpen_amount=5.0):
3      """
4      Blur the image and apply an unsharp mask filter after, creating an anti-
5      aliasing effect.
6
7      Args:
8          image: the input image.
9          blur_kernel: pre blur amount.
10         sharpen_kernel: sharpen pre blur pass amount.
11         sharpen_sigma: sharpen pre blur sigma.
12         sharpen_amount: final sharpening influence.
13
14     Returns:
15         Anti-aliased image.
16     """
17     pre_blur_img = gaussian_blur(image, blur_kernel)
18     aa_gaussian = gaussian_blur(pre_blur_img, sharpen_kernel, sharpen_sigma)
19     aa_sharpened = pre_blur_img + \
20                    (pre_blur_img - aa_gaussian) * sharpen_amount
21
22     # clipping
23     aa_sharpened = np.clip(aa_sharpened, 0.0, 1.0)
24     return aa_sharpened
```

Fonte: Autoria própria (2022).

Na *linha 17* é aplicado o primeiro filtro de *Gaussian Blur*, responsável pela suavização da imagem (DAVIES, 2012). A função (*gaussian_blur*) intermedeia a chamada da função *GaussianBlur* da biblioteca *OpenCV* (OPEN SOURCE VISION FOUNDATION, 2022) para garantir que o valor do *kernel* sempre seja ímpar, independente do valor informado, pois a função tem essa exigência.

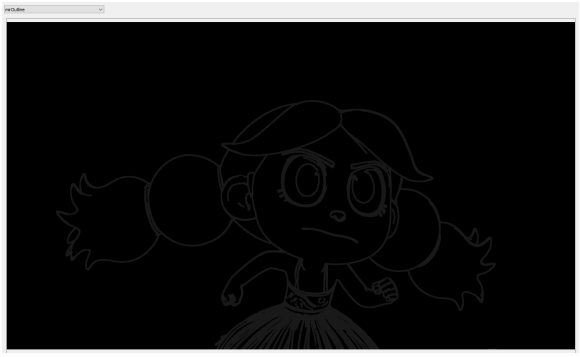
Na *linha 18* é criada uma segunda imagem com um passo adicional de *Gaussian Blur* que pertence à técnica de *unsharp masking* (DAVIES, 2012). Na *linha 19* é realizado o processo de *unsharp masking*, onde a diferença entre a imagem original (neste caso, a imagem suavizada) e a imagem borrada, multiplicada por um fator, é adicionada à imagem original (DAVIES, 2012).

Na linha 23 é utilizada a função *clip*, da biblioteca *NumPy* (NUMFOCUS FOUNDATION, 2022), para garantir que os valores da imagem permaneçam no intervalo entre 0 e 1.

3.2.10 Cor sólida

Caso seja optado por se utilizar uma cor sólida para o contorno, basicamente criamos uma imagem com a cor sólida que é então multiplicada pela máscara, obtendo-se o contorno colorido (Figura 21).

Figura 21 – Aplicação de cor sólida no contorno



Fonte: Autoria própria (2022).

3.2.11 Cor da linha respeitando iluminação e sobreposição de objetos

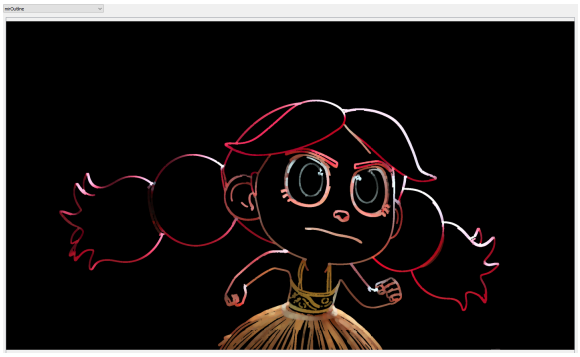
O passo da imagem final (Figura 10) é utilizado para a recuperação da cor dos objetos e sua iluminação. Não funciona simplesmente multiplicar a máscara dos contornos por este passo, pois não necessariamente os *pixels* atrás da máscara correspondem ao objeto ao qual a borda representa. Por exemplo, a borda pode ultrapassar o objeto e o conteúdo imediatamente atrás seria o fundo da cena.

A imagem utilizada para se colorir é a chamada internamente de *seeds* (Figura 18), que corresponde ao passo antes da aplicação de espessura (subseção 3.2.8) e da suavização (subseção 3.2.9).

A heurística utilizada para decidir qual a cor da borda em cada ponto (*seed*) é a de olhar em um número definido de posições vizinhas ao *pixel* no passo de profundidade (Figura 13), e procurar a posição vizinha mais próxima à câmera, essa posição é utilizada para recuperar a cor do *pixel* no passo de cor e é então atribuída ao ponto da borda.

Na sequência a imagem de *seeds* colorida é submetida à mesma função de espessura descrita na subseção 3.2.8, porém com o dobro da espessura utilizada na máscara para garantir que a cor cubra toda a máscara após o processo de suavização.

Figura 22 – Resultado da recuperação de cor dos objetos



Fonte: Autoria própria (2022).

A imagem produzida é multiplicada pela máscara do contorno e assim é obtido o contorno colorido (Figura 22).

Este processo é mais um dos processos que não podem ser executados unicamente utilizando funções pré-compiladas das bibliotecas *OpenCV* (OPEN SOURCE VISION FOUNDATION, 2022) e *NumPy* (NUMFOCUS FOUNDATION, 2022), e se faz necessária uma iteração sobre os *pixels* da imagem em *Python* (PYTHON SOFTWARE FOUNDATION, 2022), o que torna a execução mais lenta, porém mais uma vez essa perda é amenizada com a programação concorrente.O método interno ao *bucket* (subseção 3.2.3) que implementa o processo de recuperação de cor descrito é mostrado na Listagem 5. A função *get_neighbors_coordinates* é uma implementação simples que retorna em uma lista as coordenadas dos vizinhos de um ponto, dada uma coordenada de origem e uma distância, respeitando as bordas da imagem.

Listagem 5 – Método get_background_color

```
1  def get_background_color(self):
2      """
3      Gets the background color considering the ZDepth buffer.
4      """
5      out_img = np.zeros((self.bucket['height'], self.bucket['width'], 3),
6                         np.float32)
7      shape = self.out_buffers['mirOutlineSeeds'].shape
8      z_depth_img = self.internal_buffers['zDepth']
9      color_img = self.internal_buffers['original']
10     seed_indexes = np.nonzero(self.out_buffers['mirOutlineSeeds'])
11     lines, columns = seed_indexes[0], seed_indexes[1]
12
13     for i, x in enumerate(lines):
14         y = columns[i]
15         # dict of depth_value:coordinate
16         neighbors_depths = {}
17
18         closest_depth = 1
19         n_size = self.bucket['render_config'].bg_color_neighborhood_size
20
21         for n in image_proc.get_neighbors_coordinates(n_size, (x, y),
22                                                       (shape[0], shape[1])):
23             depth = z_depth_img[n[0], n[1]]
24             if depth <= closest_depth:
25                 neighbors_depths[depth] = n
26                 closest_depth = depth
27
28         closest_pixel = neighbors_depths[closest_depth]
29         # copy the color closest to the camera to the seed position
30         out_img[x, y] = color_img[closest_pixel[0], closest_pixel[1]]
31
32     out_img = image_proc.apply_line_width(out_img,
33                             self.bucket['render_config'].line_width * 2,
34                             self.bucket['render_config'].width_kernel_format)
35
36     return out_img
```

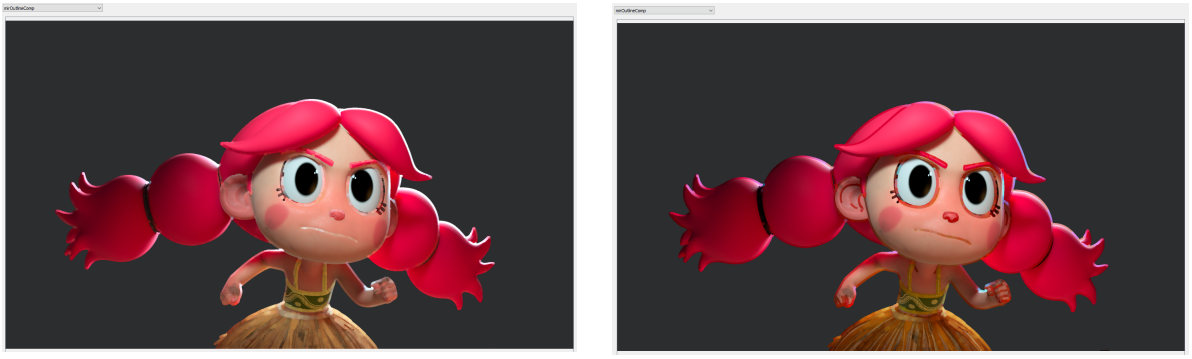
Fonte: Autoria própria (2022).

3.2.12 Alteração de matiz, saturação e valor da borda (tonalização)

Somente recuperar a cor dos objetos não produz o resultado visual esperado e um passo de tonalização da borda é necessário. Esta tonalização é realizada com deslocamento de valores de matiz, saturação e valor (*HSV*). Primeiro a imagem da borda é convertida para *HSV*, então o deslocamento é aplicado em cada canal, a função *clip* da biblioteca *NumPy* (NUMFOCUS FOUNDATION, 2022) é utilizada para garantir que os valores dos *pixels* estejam dentro do intervalo de cada canal, e então a imagem é convertida de volta para *BGR*.

Figura 23 – Tonalização dos contornos para melhor integração

(a) Contornos sem tonalização **(b) Contornos com tonalização**



Fonte: Autoria própria (2022).

Na Figura 23 é possível ver uma comparação do resultado sem tonalização na Figura 23 (a), e com tonalização na Figura 23 (b). Observe como o a tonalização do contorno se integra melhor com a renderização original.

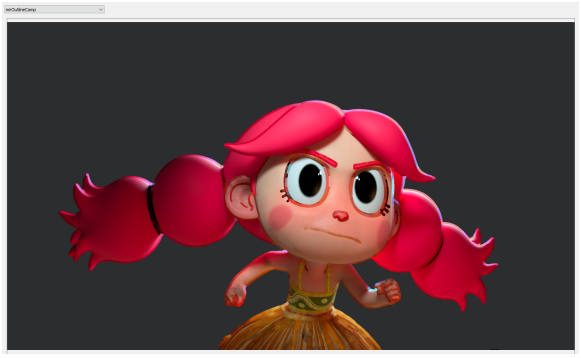
3.2.13 Composição da borda com a imagem

Depois que todas as operações envolvendo os contornos são executadas, o contorno é unido à renderização original em uma nova imagem. Na Figura 24 (a) temos a composição do contorno preservando a cor original, e na Figura 24 (b), temos a composição do contorno com cor sólida.

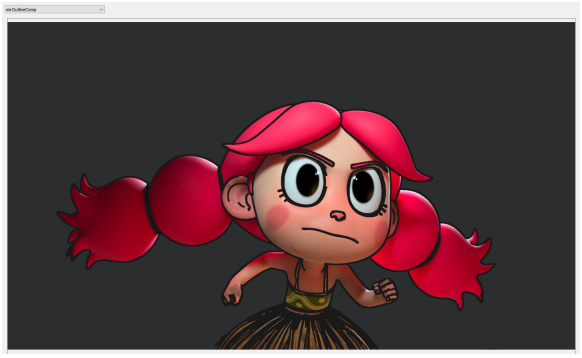
O uso do método GCE é pensado principalmente para a geração dos contornos para serem utilizados posteriormente em um software de composição, como o *Nuke* (THE FOUNDRY VISIONMONGERS LIMITED, 2022), que oferece maior controle artístico para a geração das imagens finais de um filme, porém esse passo de composição dos contornos com a renderização original fornece uma visualização rápida do resultado da solução.

Figura 24 – Composição do contorno com a renderização original

(a) Composição do contorno com preservação de cores



(b) Composição utilizando cor sólida



Fonte: Autoria própria (2022).

3.2.14 Junção das regiões processadas em paralelo em uma única imagem

Depois que os cálculos são concluídos dentro de cada *bucket* (subseção 3.2.3) o dicionário representando a sub-região, com as imagens resultantes, é adicionado a uma fila concorrente de saída. O processo principal, após enviar as sub-regiões do *frame* para renderização, aguarda o retorno das sub-regiões já computadas na fila concorrente de saída.

Conforme as sub-regiões são adquiridas, para cada imagem retornada, o processo copia o conteúdo da região em uma imagem preenchida com zeros do tamanho do *frame* original, utilizando as informações de posição disponibilizadas no dicionário que representa a região.

As imagens reconstruídas após o processo de renderização são então armazenadas em um *buffer* interno ao objeto que representa o *frame*, utilizando um dicionário contendo o nome da imagem como chave e a imagem como conteúdo, que pode ser utilizado para visualização pela interface gráfica ou então para salvar as imagens em arquivo através de um método disponibilizado para isso no objeto, descrito na subseção 3.2.15. Uma referência deste *buffer* é passada como retorno do método de renderização do *frame*.

3.2.15 Saída de dados

É disponibilizada a escrita em arquivo do resultado da renderização de um *frame* em um método pertencente ao objeto que representa o *frame*. Por padrão são salvas a imagem dos contornos coloridos, da máscara dos contornos, e a imagem composta da renderização original com os contornos, mas outros passos podem ser salvos se informados como parâmetro do método. A renderização de sequência automaticamente executa o método de salvamento em arquivo dos *frames*.

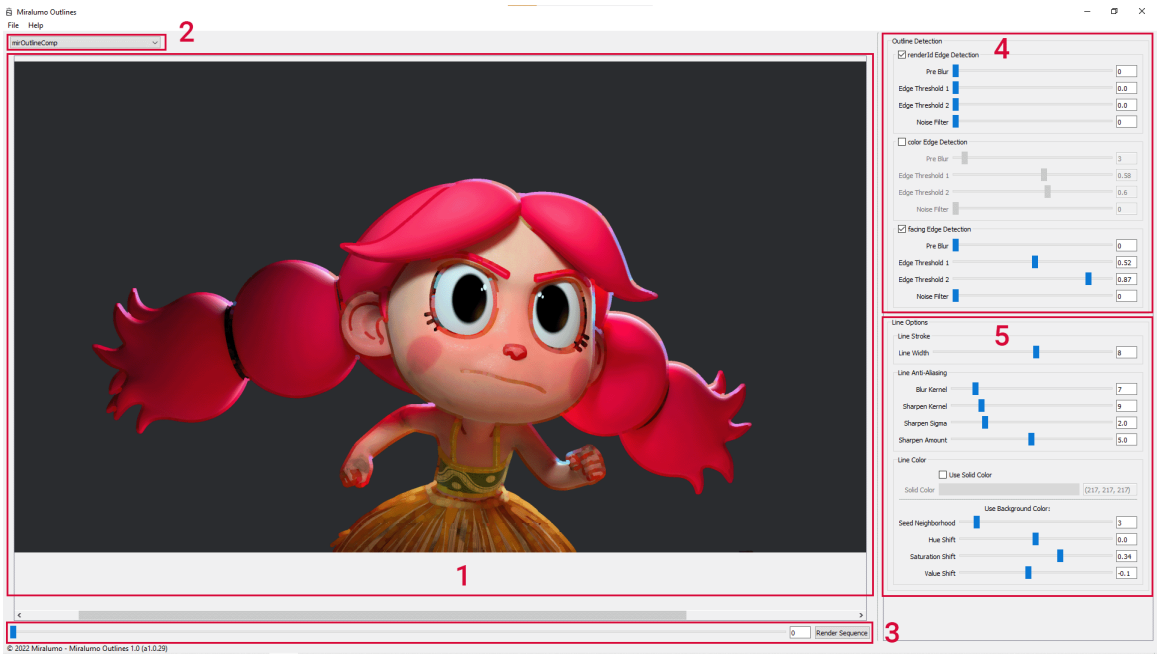
4 RESULTADOS

Neste capítulo são apresentadas a implementação do método GCE proposto utilizando uma interface gráfica, a animação utilizada para realização de testes e validação da solução, e os resultados obtidos, junto das discussões relacionadas.

4.1 Ferramenta para interação com o método GCE

Foi desenvolvida uma interface gráfica utilizando a biblioteca *PySide2* (QT GROUP, 2022) para facilitar a interação e visualização do método (Figura 25). A implementação da interface gráfica não é abordada neste trabalho pois não é o foco da proposta, que é a metodologia de geração de contornos estilizados; ela funciona como ferramenta auxiliar na utilização do método apresentado.

Figura 25 – Interface gráfica implementada para interação com o método GCE



Fonte: Autoria própria (2022).

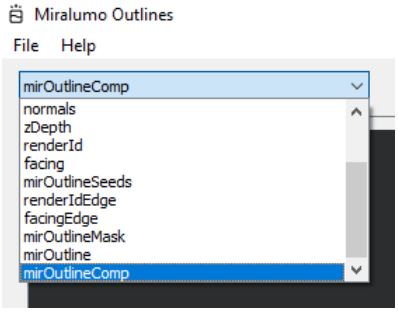
A princípio, durante a fase a fase de testes, o diretório da animação e as configuração de nomenclatura dos passos são informados em um arquivo de configurações. Uma interface gráfica para tal tarefa não foi desenvolvida nesta fase do projeto por se tratar de um protótipo.

A interface gráfica foi pensada com as seguintes funcionalidades representadas em regiões destacadas em vermelho e numeradas na Figura 25:

- (1) Visualizador de imagens: foi implementado um visualizador de imagens, onde é possível visualizar todas as imagens produzidas e utilizadas pela solução. Permitindo visualizar de forma rápida os resultados produzidos pela variação dos parâmetros res-

ponsáveis pela geração de contornos e os resultados gerados por diferentes testes de implementação.

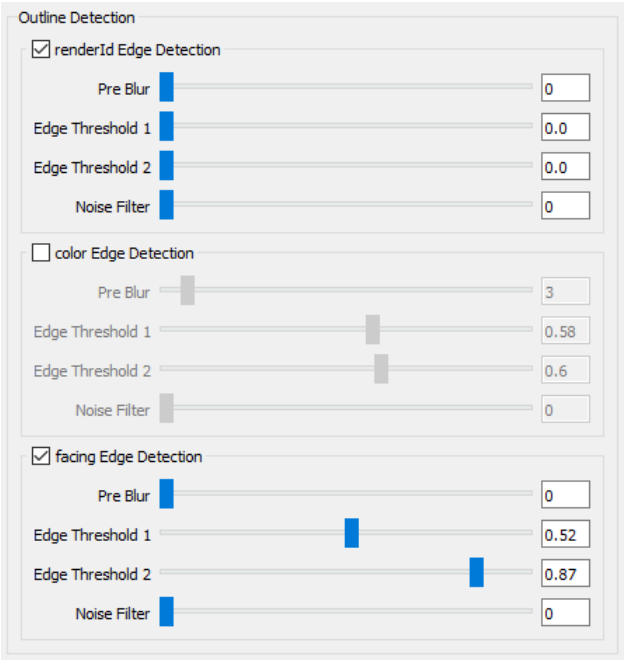
Figura 26 – Seleção de diferentes passos para visualização



Fonte: Autoria própria (2022).

- (2) Seleção de passos (Figura 26): no visualizador de imagens é mostrado o *frame* selecionado no *slider* de seleção de *frame* (região 3). Neste *combo box* é possível selecionar diferentes passos utilizados e produzidos pela solução, descritos na seção 3.2.
- (3) *Slider* de seleção de *frame* e renderização: neste slider é possível selecionar o *frame* da animação que se deseja ser visualizado. O botão *Render Sequence* é utilizado para iniciar a renderização e salvamento em arquivo dos contornos de todos os *frames* da sequência.

Figura 27 – Opções de detecção de borda

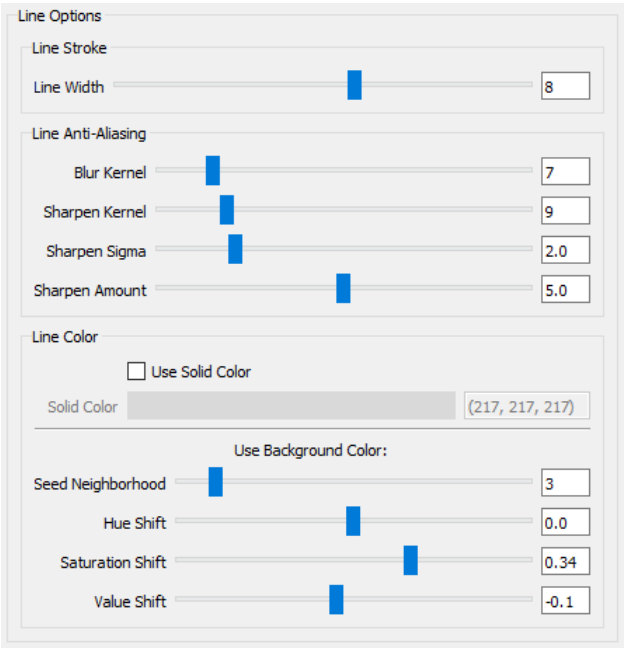


Fonte: Autoria própria (2022).

- (4) Parâmetros de detecção de borda (Figura 27): como descrito na seção 3.2, cada passo, de id por objeto e cor (subseção 3.2.2), e o passo gerado de curvatura (subseção 3.2.4), podem ser configurados independentemente para controlar sua influência

na detecção de bordas. Cada passo pode ser ativado ou desativado, controlado o nível de pré-desfoque para redução de detalhes (subseção 3.2.5), controlado os limiares utilizados pelo filtro de detecção de bordas *Canny* (subseção 3.2.6), e controlado o limiar do passo de redução de ruídos (subseção 3.2.7).

Figura 28 – Opções de configuração de linha



Fonte: Autoria própria (2022).

- (5) Parâmetros de configuração da linha (Figura 28): nesta área é possível controlar os valores de espessura da linha (subseção 3.2.8), os valores relacionados à suavização de serrilhado (subseção 3.2.9), o uso de cor sólida (subseção 3.2.10) ou cor dos objetos (subseção 3.2.11), e os controles de tonalização de borda (subseção 3.2.12).

4.2 Testes realizados

4.2.1 Animação

Para o desenvolvimento e testes da solução a produtora Miralumo forneceu uma personagem e uma animação de suas produções internas, e uma renderização específica foi computada para o teste. Na (Figura 29) é possível observar alguns dos *frames* desse animação, onde a personagem respira ofegante, limpa o rosto em um gesto de confiança e volta em posição de batalha.

A renderização foi feita gerando os passos descritos na subseção 3.2.2, em imagens separadas, utilizando o renderizador *VRay* (CHAOS SOFTWARE EOOD, 2022). Apesar do renderizador utilizado, a metodologia é agnóstica de software, e pode ser utilizada com qualquer renderizador que forneça os passos descritos.

Figura 29 – Alguns *frames* da animação produzida para testes

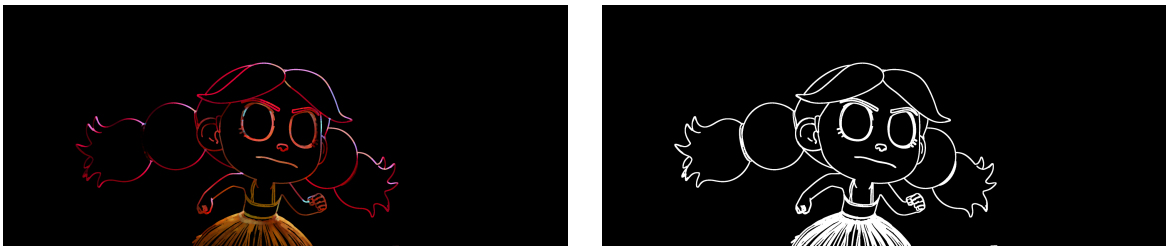
Fonte: Autoria própria (2022).

As imagens foram renderizadas em resolução 4k pois o uso de resoluções mais altas reduz a quantidade de tremulações causadas pelo processo de detecção de borda e melhora a fidelidade dos contornos, como descrito na subseção 3.2.9.

Como resultado da geração de contornos são gerados três novos passos para cada *frame*, mostrados na Figura 30.

Figura 30 – Passos gerados pela geração de contornos

(a) Contornos coloridos **(b) Máscara dos contornos**



(c) Imagem original **(d) Imagem com contorno**

Fonte: Autoria própria (2022).

A Figura 30 (a) mostra o passo dos contornos coloridos e a Figura 30 (b) mostra a máscara dos contornos. Esses passos independentes permitem que as bordas ainda possam ser trabalhadas em mais processos de pós-processamento posteriores a geração de contornos, como por exemplo, serem utilizadas dentro de um software de composição de sequências, como o *Nuke* (THE FOUNDRY VISIONMONGERS LIMITED, 2022), onde diversas modificações ainda podem ser aplicadas antes de sua integração com a renderização original.

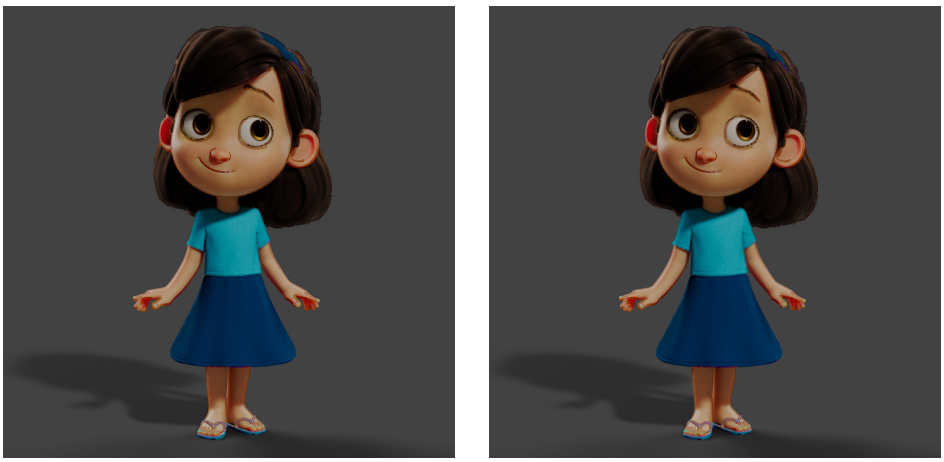
A Figura 30 (c) mostra a imagem original para comparação. A Figura 30 (d) mostra o passo do contorno composto com a renderização original, que permite uma visualização rápida do resultado gerado pela solução e, se considerado suficiente, pode ser utilizado como imagem final sem o uso de softwares adicionais como descrito anteriormente.

4.2.2 Testes estáticos

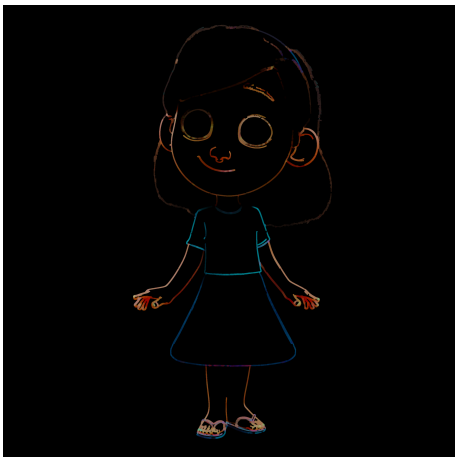
Testes com objetos e outro personagem foram realizados e são apresentados a seguir.

Figura 31 – Personagem 2

(a) Imagem original **(b) Imagem com contorno**



(c) Somente contornos



Fonte: Autoria própria (2022).

Na Figura 31 é mostrada a aplicação do método sobre outra personagem. A Figura 31 (a) mostra a imagem original, a Figura 31 (b) mostra a imagem com os contornos gerados e a Figura 31 (c) mostra somente o passo de saída com os contornos gerados.

Na Figura 32 é mostrada a aplicação do método sobre elementos de cenário, em duas renderizações de coqueiros. A Figura 32 (a) mostra a imagem original da primeira renderização, a Figura 32 (b) mostra a imagem da primeira renderização com os contornos gerados, a

Figura 32 (c) mostra a imagem original da segunda renderização e a Figura 32 (d) mostra a imagem da segunda renderização com os contornos gerados. A Figura 32 (e) e a Figura 32 (f) mostram os passo de saída com os contornos independentes.

Figura 32 – Método GCE aplicado em duas renderizações de coqueiros

(a) Imagem original coqueiro 1

(b) Imagem com contorno coqueiro 1

(c) Imagem original coqueiro 2

(d) Imagem com contorno coqueiro 2

(e) Somente contornos coqueiro 1

(f) Somente contornos coqueiro 2

Fonte: Autoria própria (2022).

4.3 Discussões

O método GCE permite uma grande variedade de variação de parâmetros, para se atingir resultados visuais diversos. Permite selecionar que tipos de contornos serão gerados, se serão gerados contornos que representam a divisão entre objetos (Figura 33 (a)), que representam mudanças de curvatura (Figura 33 (b)) ou que representam variações de textura dentro dos objetos (Figura 33 (c)). Permite controlar detalhes finos dentro de cada tipo de detecção com limiares e redução de ruídos. E permite que a borda gerada possa ser configurada, com controle de espessura (Figura 33 (f)), controle de suavização, controle da detecção de cor de objetos ou uso de cor sólida (Figura 33 (e)), e controle de tonalização da borda (Figura 33 (d)) para melhor integração com a imagem.

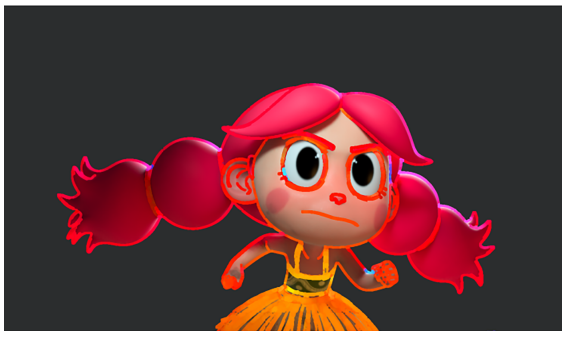
Figura 33 – Variação de parâmetros de detecção e de borda

(a) Somente divisão entre objetos

(b) Divisão entre objetos e curvatura

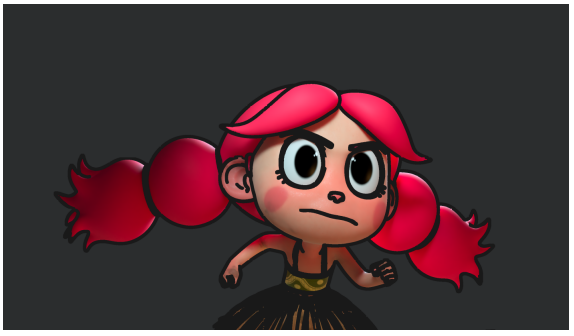
(c) Divisão entre objetos, curvatura e texturas

(d) Variação de tonalização



(e) Uso de cor sólida

(f) Variação de espessura



Fonte: Autoria própria (2022).

O poder de controle artístico do método é grande e oferece uma opção de estilização que pode ser utilizada com qualquer renderização 3D, independente de ser uma renderização já estilizada ou ser uma renderização de cunho realista, já que a solução se baseia no uso de passos comuns entre renderizadores.

O método pode ser utilizado em qualquer computador que suporte a linguagem e bibliotecas utilizadas. A velocidade de execução escala com a quantidade de núcleos disponíveis na máquina, possibilitado pela implementação concorrente.

Como limitações, o método exige especificidade de entrada com formato de imagem e passos específicos descritos na subseção 3.2.2.

5 CONCLUSÃO

Neste trabalho foi apresentado o método GCE para geração de contornos estilizados para renderizações 3D, com preservação de informações de cor, iluminação e limites dos objetos, e a implementação de uma ferramenta para interação com a solução.

A implementação do método foi realizada utilizando a linguagem de programação *Python* (PYTHON SOFTWARE FOUNDATION, 2022), a biblioteca de processamento de imagens *OpenCV* (OPEN SOURCE VISION FOUNDATION, 2022) e a biblioteca de operações matemáticas *NumPy* (NUMFOCUS FOUNDATION, 2022). O uso desta linguagem e bibliotecas permitiram um desenvolvimento mais ágil e fornecem para a comunidade científica maior replicabilidade.

O método GCE apresentado neste trabalho fornece para a comunidade uma solução de geração de contornos estilizados para animações e imagens estáticas, que é replicável e aberta. Baseada no uso de passos gerados no processo de renderização ela é utilizável praticamente com qualquer renderizador que forneça renderização multi-passos e o uso de computação concorrente no método permite também que seja amenizado o tempo de computação de alguns procedimentos implementados em *Python* (PYTHON SOFTWARE FOUNDATION, 2022).

A grande variedade de alterações de parâmetros de detecção e de alteração dos contornos permite grande controle artístico sobre a detecção de bordas, permitindo selecionar diferentes características a serem consideradas para detecção e alterar diversos aspectos das bordas geradas, como espessura, suavização e tonalização.

Como trabalhos futuros é previsto a implementação de um passo de variação de espessura de linha, para trazer mais organicidade aos contornos. É também estudada a integração do método GCE dentro de um software de composição de imagens, para que esta fique mais integrada ao contexto de produção de animações, já que se trata de um passo de pós-processamento.

REFERÊNCIAS

ARGULA, S. *et al.* Sony pictures imageworks & sony pictures animation presents: The mitchells vs. the machines. In: **ACM SIGGRAPH 2021 Production Sessions**. New York, NY, USA: Association for Computing Machinery, 2021. (SIGGRAPH '21).

CANNY, J. A computational approach to edge detection. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, 1986.

CHAOS SOFTWARE EOOD. **VRay for Maya**. 2022. Disponível em: https://www.chaos.com/vray/maya. Acesso em: 2 nov. 2022.

DAVIES, E. R. **Computer and machine vision theory, algorithms, practicalities**. 4. ed. Waltham, Massachusetts, USA: Academic Press, 2012.

DIMIAN, D. *et al.* Swing into another dimension: The making of "spider-man: Into the spider-verse". In: **ACM SIGGRAPH 2019 Production Sessions**. New York, NY, USA: Association for Computing Machinery, 2019. (SIGGRAPH '19).

GONZALEZ, R. C.; WOODS, R. E. **Digital Image Processing**. 4. ed. New York, NY, USA: Pearson, 2018.

GOOCH, B.; GOOCH, A. **Non-Photorealistic Rendering**. Natick, Massachusetts, USA: A K Peters, 2001.

HUGHES, J. F. *et al.* **Computer Graphics: Principles and Practice**. 3. ed. Upper Saddle River, NJ, USA: Addison-Wesley, 2013.

KRAUS, M. **Pixels covered by a triangle**. 2011. Disponível em: https://en.wikibooks.org/wiki/Cg_Programming/Rasterization#/media/File:Pixels_covered_by_a_triangle.png. Acesso em: 2 nov. 2022.

LAVENDER, N.; JUDSON, K.; OBRETENOV, C. The mitchells vs. the machines look of picture development. In: **The Digital Production Symposium**. New York, NY, USA: Association for Computing Machinery, 2021. (DigiPro '21).

NUMFOCUS FOUNDATION. **NumPy**. 2022. Disponível em: https://numpy.org. Acesso em: 2 nov. 2022.

OPEN SOURCE VISION FOUNDATION. **OpenCV**. 2022. Disponível em: https://opencv.org. Acesso em: 2 nov. 2022.

PYTHON SOFTWARE FOUNDATION. **Python**. 2022. Disponível em: https://www.python.org. Acesso em: 2 nov. 2022.

QT GROUP. **Qt for Python**. 2022. Disponível em: https://pypi.org/project/PySide2/. Acesso em: 2 nov. 2022.

SUFFERN, K. **Ray Tracing from the Groud Up**. Natick, Massachusetts, USA: A K Peters, 2007.

THE FOUNDRY VISIONMONGERS LIMITED. **Nuke**. 2022. Disponível em: https://www.foundry.com/products/nuke-family/nuke. Acesso em: 2 nov. 2022.