Instituto Politécnico do Cávado e do Ave
Escola Superior de Tecnologia

Engenharia em Desenvolvimento de Jogos Digitais

Bernardo Vicente Sá Neves – 23494

João Rafael Roma Antunes – 23478

Técnicas Avançadas de Programação 3D - Grupo

Junho de 2024

Bernardo Neves – 23494
João Rafael Roma Antunes – 23478

Índice

Bernardo Neves – 23494
João Rafael Roma Antunes – 23478

Bernardo Neves – 23494
João Rafael Roma Antunes – 23478

1. Introdução

O presente relatório foi elaborado no âmbito da Unidade Curricular (UC) Técnicas Avançadas de Programação 3D, parte integrante da Licenciatura em Engenharia de Desenvolvimento de Jogos Digitais, oferecida pelo Instituto Politécnico do Cávado e do Ave (IPCA). O objetivo deste documento é sintetizar e refletir sobre o conhecimento e a experiência adquiridos durante a disciplina, culminando na execução de um projeto prático que integra os saberes teóricos e técnicos assimilados.

O projeto abordado centrou-se na recriação e implementação de shaders para um ambiente de jogo de cartas desenvolvido em Unity, explorando técnicas avançadas de computação gráfica. Este desafio proporcionou uma oportunidade única para aplicar diretamente as competências e conhecimentos técnicos adquiridos ao longo da UC.

Este relatório descreve detalhadamente cada shader desenvolvido, desde a análise inicial e escolha das técnicas até à implementação final e avaliação dos resultados. A discussão abrange a fundamentação teórica por trás de cada escolha técnica, o processo detalhado de desenvolvimento, e uma avaliação crítica sobre a eficácia na consecução dos objetivos, além de possíveis melhorias para projetos futuros.

Através deste trabalho, busca-se não apenas a aplicação prática dos conhecimentos adquiridos, mas também o desenvolvimento de uma capacidade crítica e inovadora na solução de problemas de programação gráfica, essencial para enfrentar futuros desafios profissionais na indústria de jogos digitais.

Bernardo Neves – 23494
João Rafael Roma Antunes – 23478

2. Shaders e materiais

Este capítulo tem como objetivo documentar o desenvolvimento, implementação e avaliação dos três shaders criados para o projeto de jogo em Unity, conforme as especificações da Unidade Curricular de Técnicas Avançadas de Programação 3D. Cada shader foi projetado para demonstrar uma aplicação prática específica de técnicas avançadas em gráficos computacionais, refletindo os conhecimentos adquiridos ao longo da disciplina.

No decorrer deste capítulo, serão detalhadas as etapas do desenvolvimento de cada shader, incluindo a fundamentação teórica das técnicas utilizadas, o processo de implementação no ambiente Unity e a avaliação dos resultados obtidos. Além disso, será discutida a relevância de cada shader no contexto do jogo de cartas desenvolvido, destacando como estas soluções técnicas contribuem para a melhoria visual e funcional do projeto.

O primeiro shader focou-se em criar efeitos visuais realistas nas cartas, utilizando técnicas de mapeamento de texturas e iluminação dinâmica. O segundo shader foi desenvolvido para implementar efeitos de pós-processamento, aprimorando a experiência visual do jogo com filtros e ajustes de cor. O terceiro shader concentrou-se na simulação de efeitos especiais, como brilhos e animações, proporcionando um ambiente mais imersivo e visualmente atraente.

A análise crítica incluirá uma reflexão sobre a eficácia das técnicas aplicadas, os desafios enfrentados durante o desenvolvimento e sugestões de melhorias para trabalhos futuros. Dessa forma, este capítulo busca não apenas relatar o processo de criação dos shaders, mas também promover uma compreensão mais profunda das aplicações práticas de técnicas avançadas em gráficos computacionais.

2.1. Shader de Stencil

2.1.1. Análise Inicial e Escolha de Técnicas:

Este shader foi desenvolvido para renderizar detalhes escondidos no jogo, como, por exemplo, ver o lado oposto das cartas. Utiliza técnicas de stencil buffer para criar um efeito visual que distingue claramente entre a área de interesse e o fundo. A escolha de utilizar o stencil buffer permite manipular com precisão as áreas da superfície onde determinadas operações de renderização são aplicadas, garantindo a visibilidade de elementos específicos.

Bernardo Neves – 23494
João Rafael Roma Antunes – 23478

2.1.2. Implementação:

- **Propriedades**: O shader define várias propriedades ajustáveis:
    - _MainTex: A textura principal aplicada ao material.
    - _BackTex: A textura de fundo que aparece fora dos limites definidos.
    - _StencilRef: O valor de referência do stencil buffer.
    - _EdgeThreshold: O limiar que define a espessura da borda.

- **SubShader**: Define as tags e configurações de renderização, indicando que o shader é do tipo opaco e usa a técnica de iluminação Standard.
    - **Pass (Stencil Rendering)**: Este passe configura o stencil buffer para marcar áreas específicas dentro dos limites definidos.
        - Stencil Configuration: Define que o stencil buffer será sempre atualizado com o valor de referência especificado.
        - Vertex Shader (vert): Transforma as coordenadas dos vértices para o espaço de clip.
        - Fragment Shader (fragStencil): Discards pixels fora do limite definido pelo _EdgeThreshold, criando uma área marcada pelo stencil buffer.
    - **Pass (Normal Rendering)**: Este passe realiza a renderização normal fora dos limites definidos pelo stencil.
        - Vertex Shader (vert): Similar ao passe anterior, transforma as coordenadas dos vértices.
        - Fragment Shader (fragTexture): Realiza a renderização da textura principal, aplicando efeitos visuais específicos nas bordas.
        - Spinning Edge Effect: Cria um efeito visual giratório nas bordas, baseado no tempo e na posição UV.

Bernardo Neves – 23494
João Rafael Roma Antunes – 23478

2.1.3. Avaliação dos Resultados:

O shader desenvolvido implementa com sucesso a técnica de stencil buffer para distinguir entre a área de interesse e o fundo, garantindo a visibilidade dos detalhes escondidos no jogo. O uso de diferentes passes permite uma clara separação entre a configuração do stencil buffer e a renderização normal, assegurando que as áreas internas e externas sejam tratadas de maneira distinta. Esta abordagem melhora a fidelidade visual e proporciona uma flexibilidade considerável na aplicação de efeitos visuais adicionais, melhorando a experiência do usuário ao interagir com elementos do jogo.

Figura – Stencil

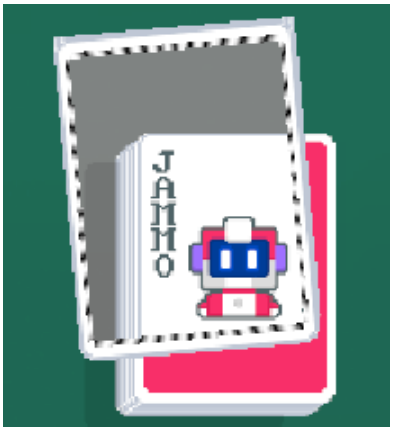

Figura – Stencil Demo

Bernardo Neves – 23494
João Rafael Roma Antunes – 23478

2.2. Shader Ripple

2.2.1. Análise Inicial e Escolha de Técnicas:

O shader Water Ripple Effect simula ondulações em uma superfície, criando um efeito visual que imita a propagação de ondas em um líquido. Esse shader é especialmente útil para superfícies aquáticas e outros contextos onde um efeito de ondulação pode enriquecer a experiência visual. Através de parâmetros ajustáveis, é possível personalizar a velocidade, frequência e amplitude das ondas, além de interagir com a posição do mouse.

2.2.2. Implementação:

- **Propriedades**: O shader define várias propriedades ajustáveis:
    - _MainTex: A textura principal aplicada ao material.
    - _TimeScale: A escala do tempo para a animação das ondulações.
    - _RippleSpeed: A velocidade das ondulações.
    - _RippleFrequency: A frequência das ondulações.
    - _RippleAmplitude: A amplitude das ondulações.
    - _EdgeThreshold: O limiar de suavização das bordas das ondulações.
    - _MousePosition: A posição do mouse para interação com o efeito.
    - _CardSelected: Booleano que indica se a carta está selecionada.
    - _CardDragging: Booleano que indica se a carta está sendo arrastada.
- **SubShader**: Define as tags e configurações de renderização, indicando que o shader é do tipo opaco com um nível de detalhe de 200.
    - Pass: Define as operações de renderização, incluindo as funções de vértice e fragmento.
- **Estrutura appdata**: Contém os dados de entrada do vértice, incluindo posição e coordenadas UV.
- **Estrutura v2f**: Contém os dados interpolados que são passados do vértice para o fragmento, incluindo as coordenadas UV e a posição do vértice.
- **Função vert**: A função de vértice transforma a posição do objeto para a posição de clip e passa as coordenadas UV.
- **Função ripple**: Calcula o efeito de ondulação baseado na posição UV e no tempo.
- **Função frag**: A função principal do fragmento calcula a cor do fragmento aplicando o efeito de ondulação:
    - Modificadores de Estado: Ajusta a velocidade das ondulações se a carta estiver selecionada e modifica o limiar das bordas se a carta estiver sendo arrastada.
    - Centro da Ondulação: Calcula a posição do centro da ondulação com base na posição do mouse.
    - Efeito de Ondulação: Aplica o efeito de ondulação calculando o deslocamento das coordenadas UV.
    - Mistura de Bordas: Ajusta a mistura das bordas das ondulações utilizando uma função smoothstep.
    - Cor da Textura: Obtém a cor da textura principal utilizando as coordenadas UV deslocadas pelo efeito de ondulação.

2.2.3. Avaliação dos Resultados:

O shader Water Ripple Effect implementado cria um efeito de ondulação dinâmico e realista nas texturas. A interação com a posição do mouse e os estados de seleção e arrasto proporciona um feedback visual interativo e responsivo. As principais melhorias incluíram a adaptação da velocidade das ondulações e do limiar das bordas com base nos estados de interação, resultando em uma experiência visualmente rica e envolvente para o usuário.

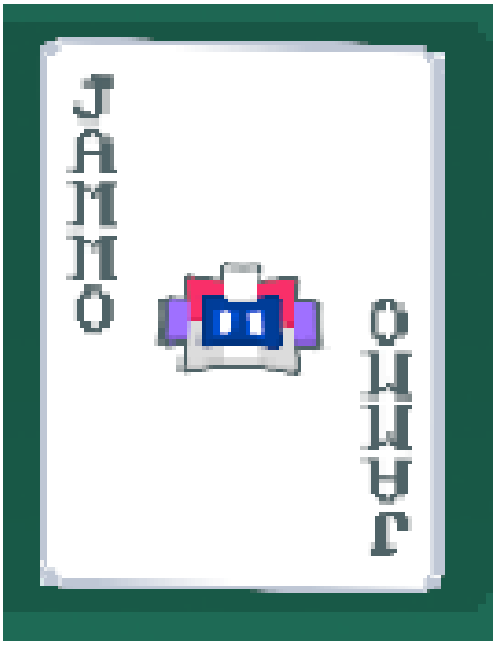

Figura - Ripple

Bernardo Neves – 23494
João Rafael Roma Antunes – 23478

2.3. Shader Swirl vortex

2.3.1. Análise Inicial e Escolha de Técnicas:

O shader Swirling Vortex aplica um efeito de redemoinho animado sobre uma textura, criando uma aparência dinâmica e interessante. Esse efeito é ideal para elementos de interface que precisam de destaque ou para efeitos visuais em jogos. A técnica utilizada envolve a rotação das coordenadas UV para criar o efeito de redemoinho e a geração de linhas de redemoinho que se movem com o tempo.

2.3.2. Implementação:

- **Propriedades**: O shader define várias propriedades ajustáveis:
    - _MainTex: A textura principal aplicada ao material.
    - _SwirlSize: O tamanho do efeito de redemoinho.
    - _SwirlGap: O espaço entre as linhas do redemoinho.
    - _LineWidth: A largura das linhas do redemoinho.
    - _TimeFactor: O fator de tempo para a animação do redemoinho.
    - _CardSelected: Booleano que indica se a carta está selecionada.
- **SubShader**: Define as tags e configurações de renderização, indicando que o shader deve ser renderizado na fila de transparência.
    - Blend SrcAlpha OneMinusSrcAlpha: Define a mistura de alpha para a transparência.
- **Estrutura appdata**: Contém os dados de entrada do vértice, incluindo posição e coordenadas UV.
- **Estrutura v2f**: Contém os dados interpolados que são passados do vértice para o fragmento, incluindo as coordenadas UV, a posição do vértice e as coordenadas UV centradas.
- **Função vert**: A função de vértice transforma a posição do objeto para a posição de clip e ajusta as coordenadas UV, centralizando-as em torno do ponto (0.5, 0.5).
- **Função frag**: A função principal do fragmento calcula a cor do fragmento aplicando o efeito de redemoinho:
    - Calcular Ângulo: Calcula o ângulo baseado nas coordenadas UV centradas.
    - Rotacionar Coordenadas: Rotaciona as coordenadas para criar o efeito de redemoinho.
    - Ajustar UV: Ajusta as coordenadas UV de volta ao intervalo [0, 1].
    - Sample da Textura: Obtém a cor da textura principal usando as coordenadas UV originais.
    - Calcular Padrão de Linha: Calcula o padrão de linha baseado nas coordenadas rotacionadas.
    - Modificador de Estado: Ajusta a largura das linhas com base no tempo e inverte o padrão de linha se a carta estiver selecionada.
    - Cor da Linha: Define a cor e a transparência das linhas do redemoinho.
    - Descarte de Fragmentos: Descarta fragmentos onde a cor da linha é totalmente transparente.
    - Retorno de Cor: Retorna a cor da linha ou a cor da textura, dependendo da transparência da linha.

Bernardo Neves – 23494
João Rafael Roma Antunes – 23478

2.3.3. Avaliação dos Resultados:

O shader Swirling Vortex Card implementado cria um efeito de redemoinho dinâmico e visualmente atraente nas texturas. A interação com o tempo e o estado de seleção da carta proporciona um feedback visual interessante e responsivo. As principais melhorias incluíram a adaptação da largura das linhas com base no tempo e a inversão do padrão de redemoinho quando a carta está selecionada, resultando em uma experiência visual rica e envolvente para o usuário.

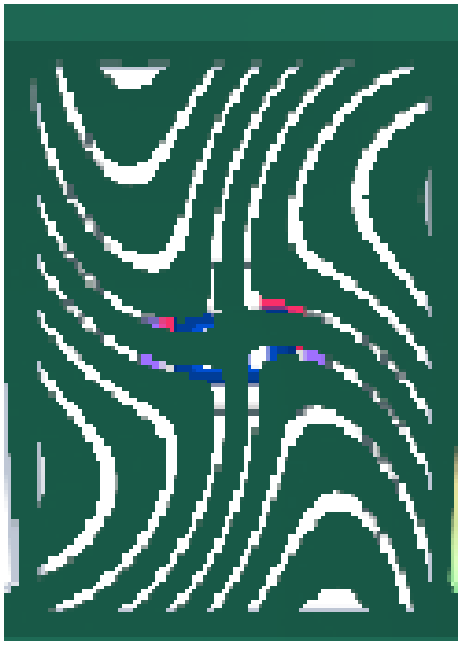

Figura - Swirl

Bernardo Neves – 23494
João Rafael Roma Antunes – 23478

2.4. Shader Polychrome

2.4.1. Análise Inicial e Escolha de Técnicas:

O shader Polychrome aplica um efeito visual vibrante aos sprites, utilizando uma combinação de alteração de matiz e um padrão de espiral dinâmico. Essa técnica é ideal para criar efeitos de destaque e animações visuais dinâmicas em elementos 2D, como cartas ou ícones. A escolha de manipulação de matiz e efeitos espirais permite personalização intensa e adaptação dinâmica ao tempo e interações do usuário.

2.4.2.Implementação:

- **Propriedades**: O shader define várias propriedades ajustáveis:
    - _MainTex: A textura principal aplicada ao material.
    - _HueOffset: O deslocamento de matiz para a coloração.
    - _SpiralSpeed: A velocidade da animação espiral.
    - _SpiralDensity: A densidade do padrão espiral.
- **SubShader**: Define as tags e configurações de renderização, indicando que o shader é do tipo opaco e deve ser renderizado na fila de transparência.
    - ZWrite Off: Desativa a gravação no buffer de profundidade.
    - Blend SrcAlpha OneMinusSrcAlpha: Define a mistura de alpha para a transparência.
- **Estrutura appdata_t**: Contém os dados de entrada do vértice, incluindo posição e coordenadas de textura.
- **Estrutura v2f**: Contém os dados interpolados que são passados do vértice para o fragmento, incluindo as coordenadas de textura e a posição do vértice.
- **Função vert**: A função de vértice que transforma a posição do objeto para a posição de clip e ajusta as coordenadas de textura.
- **Função RGBToHSV e HSVToRGB**: Funções auxiliares para conversão entre os espaços de cores RGB e HSV.
- **Função frag**: A função principal do fragmento que calcula a cor do fragmento aplicando o efeito espiral e alterando a matiz:
    - Calcula o centro da espiral ajustando as coordenadas de textura.
    - Calcula a direção e a distância do fragmento ao centro da espiral.
    - Ajusta o ângulo da espiral com base na densidade e na velocidade definidas.
    - Aplica um ajuste de velocidade extra se a carta estiver selecionada ou sendo arrastada.
    - Converte a cor do fragmento para o espaço HSV, aplica o deslocamento de matiz e retorna a cor ajustada.

2.4.3. Avaliação dos Resultados:

O shader Polychrome implementado com sucesso cria um efeito visual dinâmico e vibrante em sprites. A combinação de deslocamento de matiz e padrões espirais ajustáveis permite uma personalização significativa e interação visual atraente. A principal melhoria foi a adição de ajustes de velocidade baseados no estado de seleção e arrasto, proporcionando um feedback visual mais responsivo e intuitivo para o usuário.

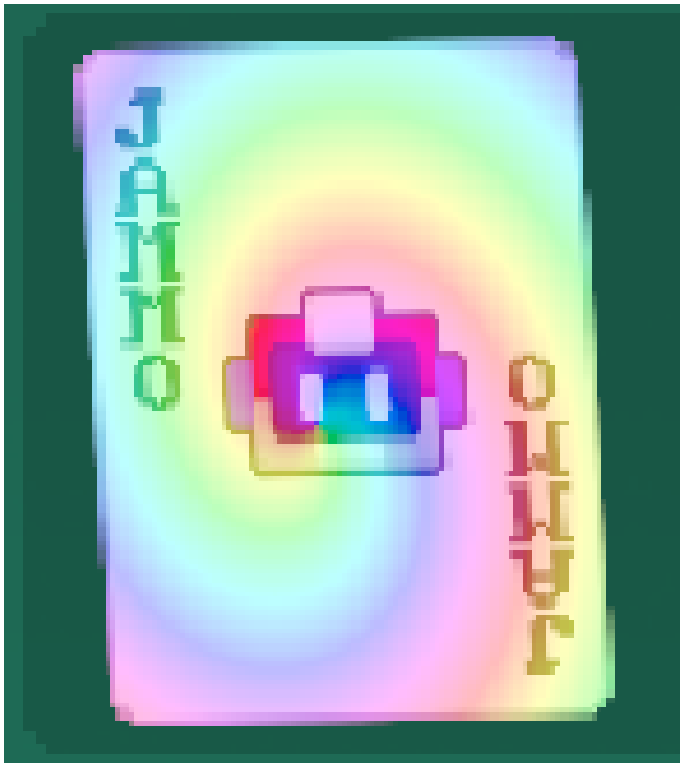

Figura – Polychrome

Bernardo Neves – 23494
João Rafael Roma Antunes – 23478

2.5. Shader de Vertex/Fragment

2.5.1. Análise Inicial e Escolha de Técnicas

Objetivo do Shader:

Este shader foi projetado para criar um efeito holográfico em um objeto renderizado em Unity. O shader utiliza uma textura base (_MainTex) e uma textura de máscara alpha (_AlphaTexture) para controlar a transparência e aplica efeitos visuais adicionais, como brilho (glow) e glitching.

Propriedades e Parâmetros:

1. **_MainTex**: Textura principal aplicada à superfície do objeto.
2. **_AlphaTexture**: Textura de máscara alpha que controla a transparência.
3. **_ScrollSpeedV**: Velocidade de rolagem vertical da máscara alpha.
4. **_GlowIntensity**: Intensidade do efeito de brilho.
5. **_GlitchSpeed**: Velocidade do efeito de glitch.
6. **_GlitchIntensity**: Intensidade do efeito de glitch.

Técnicas Escolhidas:

- **Alpha Blending**: Para criar transparência e mesclagem de cores.
- **Scroll Effect**: Para animar a textura da máscara alpha.
- **Glow Effect**: Para adicionar brilho ao objeto.
- **Glitch Effect**: Para criar um efeito de distorção visual.

2.5.2. Implementação

Shader Code:

1. **Declaração de Propriedades:**
    - Declara as texturas e parâmetros de controle.
2. **SubShader e Pass:**
    - Define as tags para a fila de renderização e configura o shader para renderização transparente.
    - Desativa iluminação e ativa a escrita de profundidade e culling.
3. **Vertex Shader (vertexFunc):**
    - Aplica distorção aos vértices baseado no tempo e na intensidade do glitch.
    - Rola as coordenadas UV da textura de máscara alpha.
4. **Fragment Shader (fragmentFunc):**
    - Amostra as texturas de cor e máscara alpha.
    - Aplica o valor alpha da máscara à textura principal.
    - Calcula e aplica o efeito de brilho (rim light).

Script de Controle (AssignToShader.cs):

1. **Inicialização:**
    - Obtém referências aos componentes necessários e inicializa o material.
    - Configura eventos para interações como seleção e arrastar do objeto.
2. **Interações do Mouse:**
    - Manipula a posição do mouse para efeitos visuais ao passar e clicar sobre o objeto.
3. **Eventos de Seleção e Arraste:**
    - Aplica efeitos visuais durante a seleção e arraste do objeto.
    - Inicia e interrompe a rotina de efeito de glitch.
4. **Efeito de Glitch (Coroutine):**
    - Aplica uma distorção temporária no brilho e intensidade de glitch em intervalos aleatórios.

2.5.3. Avaliação de Resultados

Aspectos Positivos:

1. **Efeitos Visuais:**
    - O shader cria um efeito holográfico convincente com brilho e distorções visuais dinâmicas.
    - A rolagem da textura alpha adiciona um aspecto animado ao objeto.
2. **Interatividade:**
    - A integração com eventos de seleção e arraste permite que o shader responda às interações do usuário de forma dinâmica.
    - O script ajusta visualmente o objeto baseado na posição do mouse, melhorando a experiência de usuário.

Aspectos a Melhorar:

1. **Desempenho:**
    - A frequência alta de cálculo do efeito de glitch pode impactar o desempenho em dispositivos menos potentes. Ajustes podem ser necessários para otimização.
2. **Flexibilidade:**
    - A intensificação de efeitos como o brilho pode ser configurável para permitir variações baseadas em diferentes condições ou requisitos de design.

2.5.4. Conclusão

O shader holográfico desenvolvido proporciona um visual futurista e interativo para objetos renderizados em Unity. As técnicas utilizadas, como blending, animação de textura, brilho e glitch, contribuem para um efeito visualmente atraente. A implementação bem estruturada e a consideração das interações do usuário tornam este shader uma adição valiosa para projetos que buscam um efeito holográfico dinâmico e imersivo.

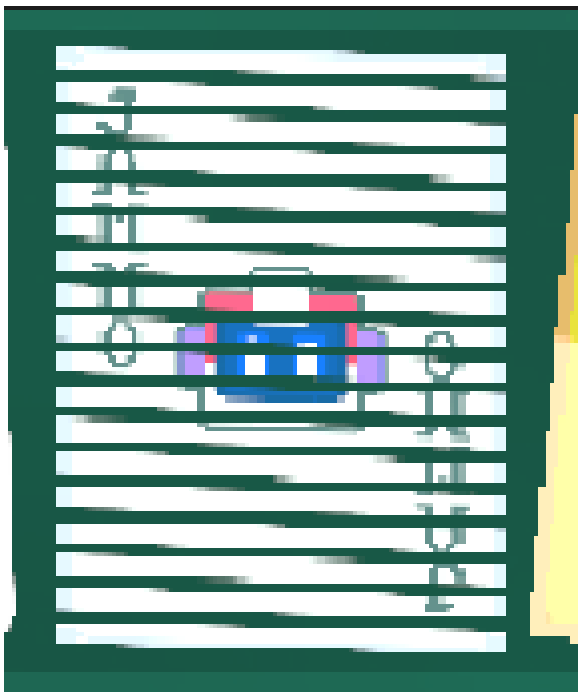

Figura - Hologram

Bernardo Neves – 23494
João Rafael Roma Antunes – 23478

2.6. Shader Gold

2.6.1. Análise Inicial e Escolha de Técnicas

Objetivo do Shader: Este shader é projetado para renderizar um plano de fundo com bordas coloridas e efeitos de destaque baseados na posição do mouse. Ele aplica uma textura principal, ajusta a cor de bordas, adiciona brilho e reage à posição do mouse para criar destaques.

Propriedades e Parâmetros:

1. **_Color**: Cor base aplicada à textura.
2. **_MainTex**: Textura principal aplicada ao plano de fundo.
3. **_HighlightColor**: Cor usada para os destaques quando o mouse se aproxima.
4. **_BorderColor**: Cor das bordas do plano de fundo.
5. **_BorderWidth**: Largura das bordas.
6. **_MousePosition**: Posição atual do mouse usada para calcular os destaques.
7. **_Glossiness**: Controla o tamanho do destaque para simular brilho.

2.6.2. Implementação

Vertex Shader (vert):

1. **Estrutura de Entrada (appdata)**: Contém a posição do vértice e coordenadas UV da textura.
2. **Estrutura de Saída (v2f)**: Passa as coordenadas UV e a posição transformada do vértice.
3. **Transformação**: Converte a posição do objeto para o espaço de recorte e mantém as coordenadas UV.

Fragment Shader (frag):

1. **Aplicação da Textura e Cor Base**:
    - Amostra a textura principal e multiplica pela cor base definida.
2. **Desenho da Borda**:
    - Calcula a distância do ponto atual às bordas do objeto.
    - Substitui a cor do pixel pela cor da borda se a distância ultrapassar um limite definido pela largura da borda.
3. **Adição de Destaques**:
    - Define duas posições de destaque baseadas na posição do mouse multiplicada por fatores negativos para criar variação.
    - Usa a função `smoothstep` para criar um gradiente suave em torno das posições de destaque.
    - Adiciona a cor do destaque à cor final do pixel, multiplicando pela cor de destaque e os gradientes calculados.

Bernardo Neves – 23494
João Rafael Roma Antunes – 23478

2.6.3. Avaliação de Resultados

Aspectos Positivos:

1. **Bordas Visíveis e Definidas**:
    - O shader cria bordas nítidas e coloridas em torno do plano de fundo, que podem ser ajustadas em largura e cor.
2. **Destaques Dinâmicos**:
    - Os destaques baseados na posição do mouse tornam o plano de fundo interativo e visualmente interessante, adicionando uma camada de dinâmica ao design.
3. **Customização**:
    - Propriedades como cor, largura da borda, cor de destaque e brilho são facilmente ajustáveis, permitindo diversas variações visuais.

Aspectos a Melhorar:

1. **Desempenho**:
    - O cálculo de destaques baseado na posição do mouse pode ser otimizado para reduzir a carga de processamento em dispositivos menos potentes.
2. **Interatividade**:
    - Adicionar mais parâmetros para controlar o comportamento dos destaques poderia oferecer uma personalização ainda maior.
3. **Efeito de Glossiness**:
    - A propriedade de brilho (`_Glossiness`) poderia ser refinada para oferecer uma transição mais suave ou mais controle sobre o efeito.

2.6.4. Conclusão

Este shader de plano de fundo oferece uma solução eficaz e personalizável para criar bordas coloridas e destaques interativos. Sua implementação é simples, mas poderosa, permitindo ajustar várias propriedades para obter o efeito visual desejado. Com algumas melhorias, especialmente no desempenho e controle de parâmetros, ele pode ser uma excelente adição para projetos que requerem um plano de fundo dinâmico e visualmente atraente.

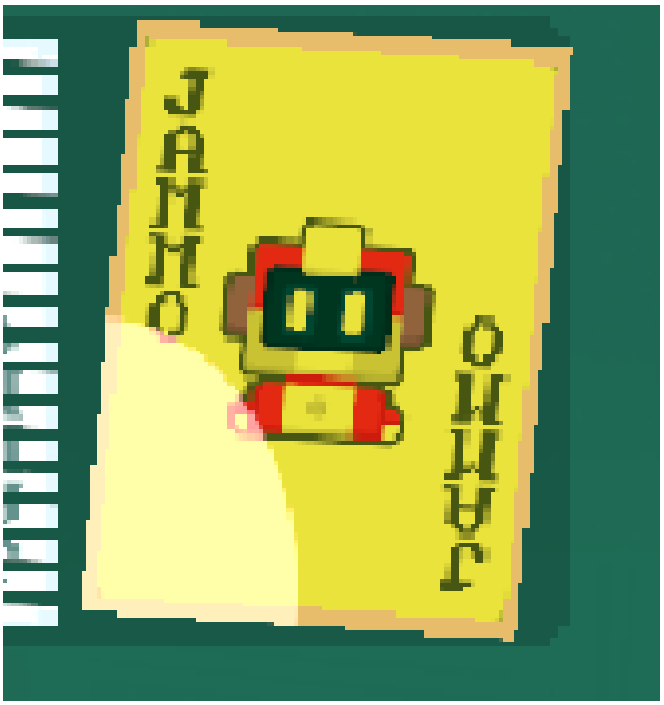

Figura – Gold Card

Bernardo Neves – 23494
João Rafael Roma Antunes – 23478

2.7. Post-Processing GrayScale

2.7.1. Análise Inicial e Escolha de Técnicas

Objetivo do Shader: Este shader é projetado para aplicar um efeito de escala de cinza em uma textura, de acordo com um parâmetro de controle (_ApplyGrayscale). Isso permite alternar entre uma exibição colorida normal e uma exibição em escala de cinza.

Propriedades e Parâmetros:

1. **_MainTex**: Textura principal aplicada à superfície.
2. **_ApplyGrayscale**: Parâmetro que controla se o efeito de escala de cinza deve ser aplicado (0.0 para desativado, >0.0 para ativado).

Técnicas Escolhidas:

- **Grayscale Conversion**: Utiliza uma fórmula ponderada para converter cores RGB em tons de cinza, preservando a percepção humana das cores.

2.7.2. Implementação

Vertex Shader (vert):

1. **Estrutura de Entrada (appdata)**: Contém a posição do vértice e coordenadas UV da textura.
2. **Estrutura de Saída (v2f)**: Passa as coordenadas UV e a posição transformada do vértice.
3. **Transformação**: Converte a posição do objeto para o espaço de recorte e mantém as coordenadas UV.

Fragment Shader (frag):

1. **Aplicação da Textura**:
    - Amostra a textura principal nas coordenadas UV fornecidas.
2. **Aplicação da Escala de Cinza**:
    - Verifica se o parâmetro _ApplyGrayscale é maior que 0.0.
    - Se for, converte a cor amostrada para escala de cinza usando a fórmula ponderada:
        - `0.299 * R + 0.587 * G + 0.114 * B`
    - Substitui os componentes RGB da cor original pelo valor cinza calculado.

Bernardo Neves – 23494
João Rafael Roma Antunes – 23478

2.7.3. Avaliação de Resultados

Aspectos Positivos:

1. **Simplicidade**:
    - O shader é simples e eficiente, realizando a conversão para escala de cinza com poucas operações.
2. **Flexibilidade**:
    - O parâmetro _ApplyGrayscale permite ativar e desativar o efeito dinamicamente, útil para diversas aplicações.
3. **Precisão na Escala de Cinza**:
    - A fórmula ponderada usada na conversão reflete a percepção humana das cores, resultando em uma conversão precisa.

Aspectos a Melhorar:

1. **Desempenho**:
    - O shader é leve e eficiente, mas para otimizações em massa, é sempre bom considerar o impacto de operações condicionais em fragment shaders.
2. **Controle Fino**:
    - Adicionar um controle mais granular (por exemplo, intensidade de escala de cinza) poderia fornecer ainda mais flexibilidade.

2.7.4. Conclusão

Este shader de escala de cinza é uma solução eficaz e direta para aplicar um efeito de desaturação controlável a texturas. A implementação é simples, mas poderosa, oferecendo a capacidade de alternar entre modos de exibição coloridos e em escala de cinza de forma dinâmica. Com poucas melhorias, como controles adicionais, ele pode ser ainda mais versátil e útil em diferentes contextos visuais.

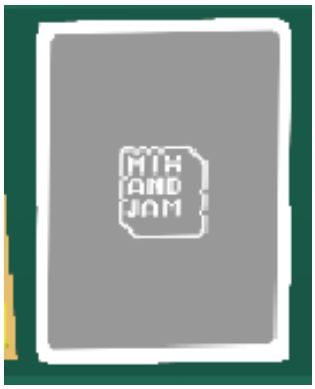

Figura – GrayScale

Figura - PostProcessing GrayScale

Bernardo Neves – 23494
João Rafael Roma Antunes – 23478

2.8. Pixalate

2.8.1. Análise Inicial e Escolha de Técnicas

Objetivo do Shader: Este shader é projetado para aplicar um efeito de pixelização a uma textura, controlado por um parâmetro (_Pixelization). Isso permite ajustar a quantidade de pixelização para criar diferentes níveis de efeito de pixel art.

Propriedades e Parâmetros:

1. **_MainTex**: Textura principal aplicada à superfície.
2. **_Pixelization**: Parâmetro que controla a quantidade de pixelização (0.0 para nenhuma pixelização, 1.0 para máxima pixelização).

Técnicas Escolhidas:

- **Pixelation Effect**: Utiliza a manipulação das coordenadas UV para agrupar pixels em blocos maiores, criando o efeito de pixelização.

2.8.2. Implementação

Vertex Shader (vert):

1. **Estrutura de Entrada (appdata_t)**: Contém a posição do vértice e coordenadas UV da textura.
2. **Estrutura de Saída (v2f)**: Passa as coordenadas UV e a posição transformada do vértice.
3. **Transformação**: Converte a posição do objeto para o espaço de recorte e mantém as coordenadas UV.

Fragment Shader (frag):

1. **Aplicação da Textura**:
    - Se o parâmetro _Pixelization for zero, retorna a cor original da textura sem aplicar pixelização.
2. **Cálculo da Pixelização**:
    - Calcula a quantidade de pixelização interpolando entre 100.0 e 1.0 com base no parâmetro _Pixelization.
    - Define o tamanho do pixel inversamente proporcional à quantidade de pixelização.
    - Ajusta as coordenadas UV da textura para agrupar pixels em blocos maiores, efetivamente aplicando a pixelização.
3. **Amostragem da Textura Pixelizada**:
    - Amostra a textura na coordenada UV ajustada para pixelização, retornando a cor pixelizada.

2.8.3. Avaliação de Resultados

Aspectos Positivos:

1. **Controle Flexível**:
    - O parâmetro _Pixelization permite ajustar dinamicamente a quantidade de pixelização aplicada, oferecendo flexibilidade visual.
2. **Efeito de Pixelização**:
    - O shader cria um efeito de pixelização convincente, útil para estilos de arte retro ou para destacar certos elementos visuais.
3. **Simplicidade e Eficiência**:
    - O shader é relativamente simples e eficiente, aplicando o efeito de pixelização com poucas operações matemáticas.

Aspectos a Melhorar:

1. **Desempenho**:
    - O cálculo da interpolação e ajuste das coordenadas UV é leve, mas em grandes superfícies ou texturas de alta resolução, pode ser necessário otimizar.
2. **Transições Suaves**:
    - Adicionar uma transição mais suave entre diferentes níveis de pixelização pode melhorar a aparência visual, especialmente ao animar o parâmetro _Pixelization.
3. **Interpolação de Pixelização**:
    - A interpolação entre 100.0 e 1.0 pode ser ajustada para oferecer uma gama mais precisa de pixelização dependendo das necessidades específicas.

2.8.4. Conclusão

Este shader de pixelização é uma solução eficaz para criar um efeito visual de pixel art ajustável. A implementação é simples, permitindo controlar dinamicamente a quantidade de pixelização aplicada a uma textura. Com algumas melhorias em transições e otimização, ele pode ser uma ferramenta valiosa para adicionar efeitos visuais retro ou destacar elementos específicos em um projeto.

Figura - Pixalate

3. Conclusões

A implementação dos shaders propostos permitiu aplicar conhecimentos avançados de programação gráfica no Unity, consolidando tanto habilidades práticas como teóricas aprendidas nas aulas. Cada shader desenvolvido apresentou desafios únicos que contribuíram significativamente para o aprendizado e crescimento na área de desenvolvimento de jogos e gráficos computacionais.

Estas implementações não só reforçaram o nosso entendimento sobre shaders e suas aplicações práticas, mas também demonstraram a importância de técnicas avançadas para melhorar a qualidade visual e a interatividade nos jogos. Através deste trabalho, tivemos a oportunidade de experimentar e superar dificuldades técnicas, aprimorando as nossas habilidades de resolução de problemas e inovação no design de gráficos.

O resultado é um conjunto de shaders eficientes e versáteis, prontos para serem utilizados em projetos futuros, evidenciando o potencial de técnicas gráficas avançadas para enriquecer experiências visuais em jogos e aplicações interativas. Este trabalho não só demonstra a aplicação prática dos conhecimentos adquiridos, como também sublinha a importância da criatividade e da inovação na resolução de problemas complexos de programação gráfica. Assim, os shaders desenvolvidos servem como um testemunho da nossa capacidade de integrar teoria e prática para alcançar resultados visuais de alta qualidade em ambientes de jogo.

4. Bibliografia

[1] https://docs.unity3d.com/Manual/SL-ShaderPrograms.html, <consultado em 06-2024>

[2] https://docs.unity3d.com/Manual/SL-SurfaceShaders.html , <consultado em 06-2024>

[3] https://docs.unity3d.com/Manual/SL-Reference.html, <consultado em 06-2024>

[4] https://docs.unity3d.com/Manual/SL-VertexFragmentShaderExamples.html, <consultado em 06-2024>

[5] https://docs.unity3d.com/Manual/SL-Properties.html, <consultado em 06-2024>

[6] Moisés Herculano de Lima Moreira, (2024). Materiais das aulas de Técnicas Avançadas de Programação 3D, Instituto Politécnico do Cávado e do Ave.