Shaders

Shaders são pequenos programas que executam diretamente na GPU (Unidade de Processamento Gráfico). Eles são usados principalmente para determinar a aparência final dos objetos renderizados no jogo ou aplicação gráfica. Existem vários tipos de shaders, cada um com funções específicas, mas os dois principais são:

1. **Vertex Shaders**: Processam cada vértice de um modelo 3D. Eles são responsáveis por transformar as coordenadas 3D dos vértices em coordenadas 2D na tela, aplicando transformações como rotação, translação e escala. Além disso, podem calcular atributos como normais e coordenadas de textura.
2. **Fragment (ou Pixel) Shaders**: Processam cada fragmento (ou pixel) gerado pelo rasterizador. Eles determinam a cor final de cada pixel, aplicando cálculos de iluminação, texturas, sombras e outros efeitos visuais.

Shaders são escritos em linguagens específicas, como HLSL (High-Level Shading Language) para DirectX, GLSL (OpenGL Shading Language) para OpenGL, e ShaderLab para a Unity.

Materiais

Materiais são os objetos na engine de jogo que utilizam shaders para determinar como a superfície de um modelo 3D deve ser renderizada. Um material pode ser visto como um contêiner que agrupa um shader e define os valores dos parâmetros necessários por esse shader. Esses parâmetros podem incluir:

- **Texturas**: Imagens que são mapeadas sobre a superfície do modelo. Exemplos incluem mapas de difusão (cores base), normais (detalhes de relevo), especulares (reflexões), e outros.
- **Propriedades de Cor**: Valores de cor que podem ser usados para ajustar a aparência do material, como a cor base, cor de emissão, etc.
- **Propriedades de Iluminação**: Parâmetros que controlam como a luz interage com a superfície, como refletividade, rugosidade, transparência, etc.

Exemplo Prático

Imagine que você está criando um jogo 3D e tem um personagem principal que usa uma armadura metálica brilhante. Para conseguir o efeito visual desejado, você faria o seguinte:

1. **Criar o Shader**: Escrever um shader que calcula a aparência metálica da armadura. Este shader pode incluir cálculos para simular reflexões de luz, brilho, e talvez alguns detalhes de desgaste.
2. **Criar o Material**: Criar um material na engine de jogo (Unity, por exemplo) e atribuir o shader a este material. Em seguida, você definiria as texturas e parâmetros necessários para o shader, como:
    - Textura de difusão (cor base da armadura)
    - Textura de normais (detalhes de relevo na armadura)
    - Textura especular (definindo as áreas mais brilhantes)
    - Valores de cor e propriedades de iluminação específicos.
3. **Aplicar o Material**: Aplicar o material ao modelo 3D do personagem. O motor de jogo usará o shader e as configurações do material para renderizar a armadura com o efeito metálico desejado

Workflow

1. **Criação do Shader**:
    - O shader é escrito ou escolhido, definindo como os dados do objeto serão processados para renderização.
2. **Criação do Material**:
    - Um material é criado e associado a um shader específico.
    - Parâmetros do shader são ajustados dentro do material (como texturas, cores, valores numéricos, etc.).
3. **Aplicação do Material ao Objeto**:
    - O material é aplicado a um objeto 3D.
    - O objeto é então renderizado de acordo com as instruções do shader contidas no material.

Render Pipelines

Embora a estrutura básica do shader seja a mesma, diferentes render pipelines podem adicionar etapas adicionais ou variáveis específicas para otimizar e estilizar a renderização. As três principais render pipelines na Unity são:

1. **Built-in Render Pipeline**:
    - **Descrição**: O pipeline padrão mais antigo e amplamente utilizado. É muito flexível e altamente compatível com uma ampla gama de shaders.
    - **Utilização**: Usado em muitos projetos devido à sua simplicidade e compatibilidade.
2. **Universal Render Pipeline (URP)**:
    - **Descrição**: Projetado para oferecer um bom equilíbrio entre desempenho e qualidade visual, sendo eficiente para uma variedade de plataformas, especialmente móveis.
    - **Utilização**: Ideal para jogos e aplicações que precisam ser executados em uma ampla gama de dispositivos, incluindo dispositivos móveis e consoles.
3. **High-Definition Render Pipeline (HDRP)**:
    - **Descrição**: Focado em fornecer a mais alta qualidade visual possível, com suporte para iluminação física precisa, sombras realistas e outros efeitos avançados.
    - **Utilização**: Usado para jogos de alta qualidade, aplicações AR/VR e projetos que exigem gráficos de ponta.

Resumo

- **Shader**: Define como renderizar.
- **Material**: Aplica o shader a um objeto e define valores específicos para as propriedades do shader.
- **Objeto**: Recebe o material e é renderizado de acordo com as definições do shader contidas no material.
- **Vertex Shader**: Transforma as coordenadas dos vértices e prepara dados para o fragment shader.
- **Fragment Shader**: Calcula a cor final dos pixels.
- **Render Pipelines**: Built-in, URP, e HDRP oferecem diferentes níveis de otimização e qualidade visual.

Componentes do Material

Smoothness

Smoothness é uma propriedade dos materiais que controla a aparência de uma superfície quanto à sua reflexão e espalhamento de luz. Em termos simples, a smoothness determina o quão polida ou rugosa uma superfície é.

- **Alta Smoothness**: Superfícies com alta smoothness são altamente polidas, refletindo a luz de forma clara e nítida, quase como um espelho. Exemplos incluem superfícies de vidro, metal polido e água calma.
- **Baixa Smoothness**: Superfícies com baixa smoothness são rugosas, espalhando a luz em várias direções, o que resulta em reflexões difusas e sem brilho. Exemplos incluem superfícies de concreto, madeira não tratada e tecidos.

Na Unity, quando você ajusta a smoothness de um material, você está essencialmente controlando a extensão da reflexão especular (nítida) versus difusa (espalhada) da luz.

Metalness/Specularity

Metalness (ou **Metalicidade**) e **Specularity** (ou **Especularidade**) são propriedades que controlam como a luz interage com a superfície de um material de maneiras diferentes, mas relacionadas.

Metalness

Metalness é uma propriedade usada em shaders PBR (Physically Based Rendering) para definir se um material é metálico ou não. Essa propriedade ajuda a determinar como a luz é refletida na superfície de um material.

- **Valor de Metalness 0 (Não Metálico)**: Materiais não metálicos (dielétricos) como madeira, plástico e cerâmica. A luz que atinge a superfície desses materiais é principalmente espalhada e a cor da reflexão é dominada pela cor difusa do material.
- **Valor de Metalness 1 (Metálico)**: Materiais metálicos refletem a luz de forma mais direta e a cor da reflexão é a mesma que a cor do material. Exemplos incluem ouro, prata e cobre.

Specularity

Specularity refere-se ao nível de brilho e à cor da luz refletida por uma superfície. Enquanto a metalness é uma abordagem simplificada e mais comum nos shaders modernos, a specularity oferece um controle mais detalhado sobre a reflexão especular.

- **Alto Valor de Specularity**: A superfície tem reflexões mais intensas e nítidas. Materiais com alta especularidade parecem mais brilhantes e polidos.
- **Baixo Valor de Specularity**: A superfície tem reflexões mais difusas e suaves. Materiais com baixa especularidade parecem mais opacos e menos polidos.

Exemplos Práticos na Unity

- **Smoothness Slider**: Controla o grau de polidez da superfície.
- **Metallic Slider**: Define a propriedade metálica do material.
- **Specular Color (em shaders especulares)**: Ajusta a cor e intensidade da reflexão especular.

Texturas

Texturas são imagens aplicadas a objetos 3D para dar-lhes cor e detalhes de superfície. Elas definem como a superfície de um modelo 3D deve parecer em termos de cor, brilho, relevo, transparência e outras características visuais. Existem diferentes tipos de texturas, cada uma com um propósito específico:

1. **Diffuse Texture (Albedo Map)**: Define a cor base do objeto sem considerar a iluminação ou efeitos de luz.
2. **Normal Map**: Simula pequenos detalhes e variações na superfície do objeto, como rugosidades e relevos, sem alterar a geometria real do modelo.
3. **Specular Map**: Define a quantidade e cor da luz refletida pela superfície, determinando a aparência do brilho especular.
4. **Bump Map**: Semelhante ao normal map, mas utiliza valores de escala de cinza para simular variações na superfície.
5. **Height Map**: Usa valores de escala de cinza para representar variações de altura na superfície do objeto, frequentemente usado para efeitos de deslocamento.
6. **Occlusion Map**: Representa as áreas que recebem menos luz ambiente, adicionando sombras sutis nas partes menos expostas do objeto.
7. **Emissive Map**: Define as áreas da superfície que emitem luz própria, fazendo com que essas partes brilhem mesmo sem iluminação externa.
8. **Metallic Map**: Define quais partes do objeto têm propriedades metálicas, influenciando como a luz interage com a superfície.
9. **Roughness Map**: Controla a suavidade ou aspereza da superfície, afetando a dispersão da luz refletida.

Técnicas Avançadas

1. **Tiling and Offset**: Ajuste o tiling e o offset das texturas para evitar repetição visível e melhorar o detalhamento.
2. **Blending Textures**: Combine múltiplas texturas usando shaders para criar efeitos mais realistas, como misturar texturas de sujeira com as de superfícies limpas.
3. **Dynamic Textures**: Modifique texturas em tempo real para criar efeitos dinâmicos, como dano em tempo real ou mudanças de estação.

Exemplos Práticos

1. **Personagens**: Use texturas para a pele, roupas e acessórios, combinando normal maps para detalhes da pele e emissive maps para acessórios que brilham.
2. **Cenários**: Aplique texturas em paredes, chão e objetos, usando occlusion maps para adicionar sombras sutis e normal maps para detalhes de relevo.
3. **Efeitos Visuais**: Crie texturas para efeitos visuais como fogo, água e magia, utilizando emissive maps para brilho e normal maps para detalhes de superfície.

Com essas ferramentas e técnicas, você pode criar ambientes e objetos 3D altamente detalhados e realistas, aprimorando a qualidade visual do seu jogo ou aplicaç