CG – Shaders - Godot - Parte 3

COMPUTAÇÃO GRÁFICA

Godot

Vamos começar com o shader do final da aula passada, então repetiremos aqui o passo a passo do que foi feito.

Godot - Começando

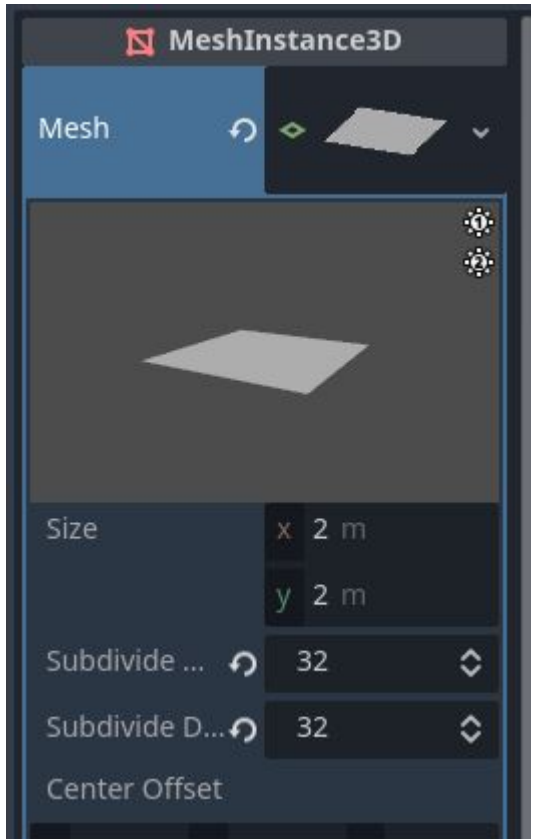
Crie uma cena 3D e adicione um novo Node MeshInstance3D à sua cena.

Na aba do inspetor ao lado de "Mesh", clique em "[vazio]" e selecione "Novo PlaneMesh". Em seguida, clique na imagem de um plano que aparece. Isso adiciona um PlaneMesh à nossa cena.

Em seguida, na janela de visualização principal, clique no canto superior esquerdo no botão que diz “Perspectiva”. Um menu aparecerá. No meio do menu estão opções de como exibir a cena. Selecione 'Exibir Wireframe'. Isso permitirá que você veja os triângulos que compõem o plano.

Godot - Começando

Agora modificamos o subdivide para 32x32 para aumentar o número de faces.



Godot - Começando

Clique ao lado de "Material" onde diz "[vazio]" e selecione "Novo ShaderMaterial". Em seguida, clique na esfera que aparece.

Agora clique ao lado de "Shader" onde diz "[vazio]" e selecione "Novo Shader".

Utilizaremos o código completo da aula anterior:

```
shader_type spatial;

uniform float height_scale = 0.5;
uniform sampler2D noise;
uniform sampler2D normalmap;

varying vec2 tex_position;

void vertex() {
  tex_position = VERTEX.xz / 2.0 + 0.5;
  float height = texture(noise, tex_position).x;
  VERTEX.y += height * height_scale;
}

void fragment() {
  NORMAL_MAP = texture(normalmap, tex_position).xyz;
}
```

Godot - Ruído

Agora procure no inspetor seu material. Você deverá ver uma seção chamada "Shader Params". Se você abri-lo, verá uma seção chamada “Noise”.Clique ao lado onde diz "[vazio]" e selecione "Novo NoiseTexture2D".

Em seguida, em seu NoiseTexture2D, clique ao lado de onde diz "Noise" e selecione "Novo FastNoiseLite".

O FastNoiseLite é usado pelo NoiseTexture2D para gerar um mapa de altura. Depois de configurá-lo e deve ficar assim

Godot - Luz

Defina esta segunda textura uniform para outro NoiseTexture2D com outro FastNoiseLite. Mas desta vez, ative a opção As Normal map.

Godot - Vertex

Primeiro vamos modificar a função padrão para cálculo da NORMAL, alteramos diretamente a NORMAL na vertex ao invés do NORMAL_MAP, isso vai ajudar mais para frente no tutorial.

```
float height(vec2 position) {
  return texture(noise, position / 10.0).x;
}
```

```
vec2 pos = VERTEX.xz;
float k = height(pos);
VERTEX.y = k;
NORMAL = normalize(vec3(k - height(pos + vec2(0.1, 0.0)), 0.1, k - height(pos + vec2(0.0, 0.1))));
```

Godot - Fragment

A partir daqui vamos manipular o fragment e fazer mais algumas alterações para ter um efeito dinâmico no nosso shader.

A Godot fornece ao usuário um monte de parâmetros que podem ser configurados opcionalmente (AO, SSS_Strength, RIM, etc.). Esses parâmetros correspondem a diferentes efeitos complexos (Oclusão de ambiente, Dispersão de subsuperfície, Iluminação de borda, etc.).Isso torna mais fácil para os usuários obterem shaders PBR complexos.

Claro, Godot também permite ignorar todos esses parâmetros e escrever um shader totalmente personalizado. Para obter uma lista completa desses parâmetros, é interessante consultar o documento de referência do spatial shader.

Godot - Fragment

Uma diferença entre a função vertex e a função fragment é que a função vertex é executada por vértice e define propriedades como VERTEX (posição) e NORMAL (direção), enquanto o shader de fragment é executado por pixel e, o mais importante, define a cor ALBEDO do MeshInstance3D.

O uso padrão da função fragment na Godot é configurar diferentes propriedades de materiais e deixar a Godot cuidar do resto. Para fornecer ainda mais flexibilidade, Godot também oferece coisas chamadas modos de renderização.

Godot - Fragment

Os modos de renderização são definidos na parte superior do shader, diretamente abaixo de shader_type, e especificam que tipo de funcionalidade você deseja que os aspectos integrados do shader tenham. Por exemplo, se você não quiser que as luzes afetem um objeto, defina o modo de renderização como unshaded

```
render_mode unshaded;
```

Também é possível agrupar vários modos de renderização. Por exemplo, se você quiser usar um shader toon em vez de shader PBR mais realista, defina o modo difuso e o modo especular como toon:

```
render_mode diffuse_toon, specular_toon;
```

Godot - Fragment

Vamos ver agora como pegar o terreno acidentado da aula e transformá-lo em um oceano. Primeiro vamos definir a cor da água. Fazemos isso definindo ALBEDO. ALBEDO é um vec3 que contém a cor do objeto. Vamos configurá-lo para um tom de azul.

```
ALBEDO = vec3(0.1, 0.3, 0.5);
```

Godot - Fragment

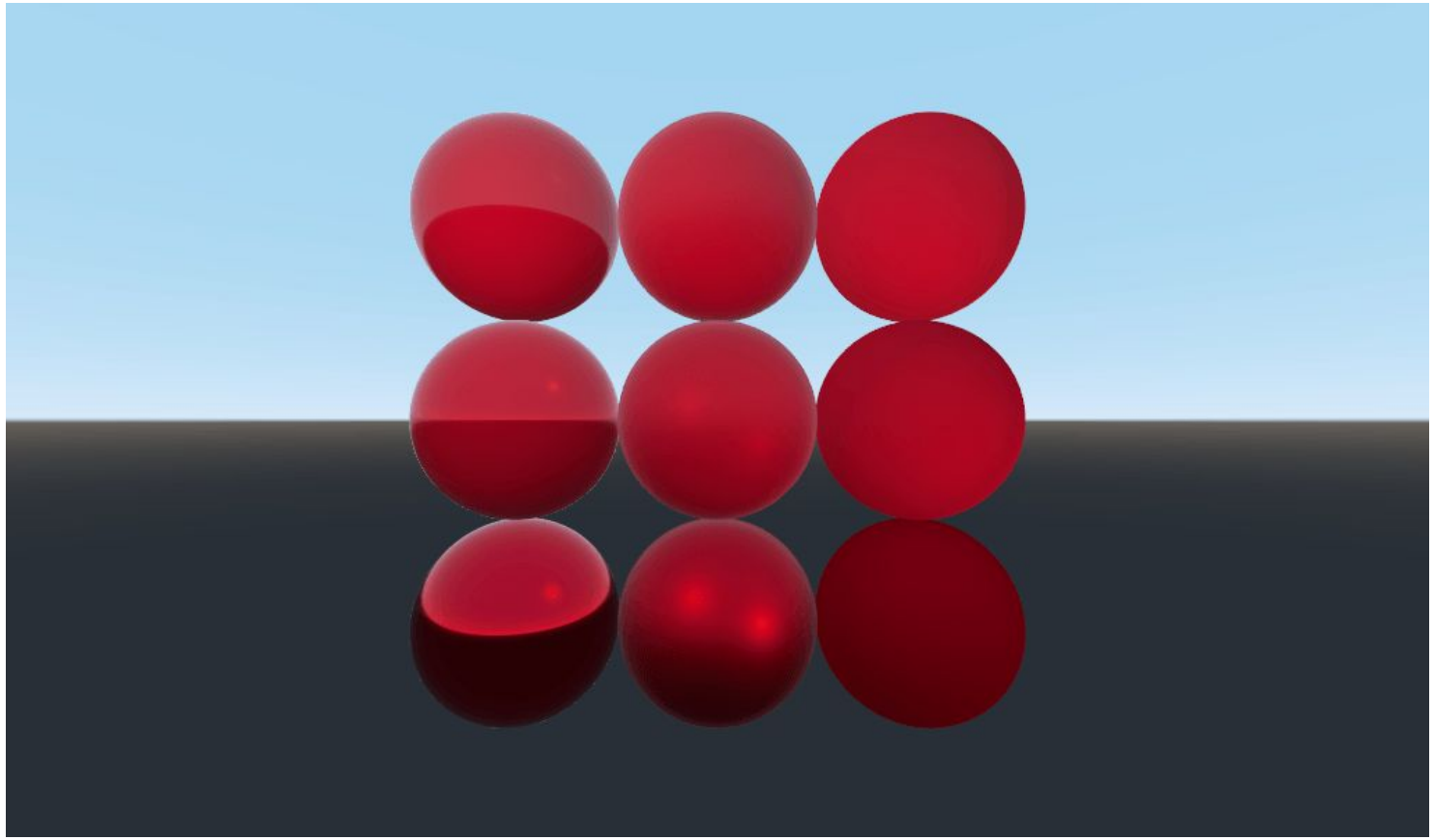
O modelo PBR que Godot utiliza depende de dois parâmetros principais: METALLIC e ROUGHNESS.

ROUGHNESS especifica quão lisa/áspera é a superfície de um material. Uma rugosidade baixa fará com que o material pareça um plástico brilhante, enquanto uma rugosidade alta fará com que a cor do material pareça mais sólida.

METALLIC especifica o quanto o objeto é parecido com um metal. É melhor próximo a 0 ou 1. Pense em METALLIC como uma alteração do equilíbrio entre o reflexo e a cor ALBEDO. Um METALLIC alto quase ignora ALBEDO completamente e parece um espelho do céu. Enquanto um METALLIC baixo tem uma representação mais igual da cor do céu e da cor ALBEDO.

Godot - Fragment

ROUGHNESS aumenta de 0 a 1 da esquerda para a direita enquanto METALLIC aumenta de 0 a 1 de cima para baixo.



Godot - Fragment

A água não é um metal, então definiremos sua propriedade METALLIC como 0,0. A água também é altamente reflexiva, por isso definiremos sua propriedade ROUGHNESS para um valor bastante baixo também.

```
METALLIC = 0.0;
ROUGHNESS = 0.01;
```

Godot - Movimento

Voltando à função de vertex, podemos animar as ondas usando a variável integrada TIME.

TIME é uma variável integrada que pode ser acessada pelas funções de vertex e fragment.

Faremos uma função chamada height receber TIME.

```
float height(vec2 position, float time) {
  return texture(noise, position / 2.0 + 0.5).x * height_scale;
}
```

Godot - Movimento

Faremos primeiro a função height trabalhar através do tempo por uma função de cosseno.

```
float height(vec2 position, float time) {
  vec2 offset = 0.01 * cos(position + time);
  return texture(noise, (position / 2.0 + 0.5) * height_scale - offset).x;
}
```

Godot - Movimento

O que torna os shaders tão poderosos é que você pode obter efeitos complexos usando matemática.

Para ilustrar isso, vamos levar nossas ondas para o próximo nível, modificando a função height() e introduzindo uma nova função chamada wave().

wave() tem um parâmetro, position, que é o mesmo de height(). Vamos chamar wave() várias vezes em height() para simular a aparência das ondas.

Godot - Movimento

A função desloca a posição pela textura do ruído. Isso fará com que as ondas se curvem, de modo que não sejam linhas retas completamente alinhadas com a grade.

Depois Definimos uma função semelhante a uma onda usando sin() e posição. Normalmente as ondas sin() são muito redondas. Usamos abs() para absoluto para dar-lhes uma crista acentuada e restringida ao intervalo 0-1. E então subtraímos de 1,0 para colocar o pico no topo.

Multiplicamos então a onda direcional x pela onda direcional y e aplicamos uma potência para aguçar os picos.

```
float wave(vec2 position){
  position += texture(noise, (position / 2.0 + 0.5) * height_scale).x * 2.0 - 1.0;
  vec2 wv = 1.0 - abs(sin(position));
  return pow(1.0 - pow(wv.x * wv.y, 0.65), 4.0);
}
```

Godot - Movimento

Podemos agora modificar a função height()

```
float height(vec2 position, float time) {
  float h = wave(position);
  return h;
}
```

Godot - Movimento

Podemos fazer ainda melhor se colocarmos múltiplas ondas umas sobre as outras em frequências e amplitudes variadas.

O que isso significa é que vamos dimensionar a posição de cada uma para tornar as ondas mais finas ou mais largas (frequência). E vamos multiplicar a saída da onda para torná-la mais curta ou mais alta (amplitude).

Aqui está um exemplo de como você pode colocar as quatro ondas em camadas para obter ondas com aparência mais bonita.

Godot - Movimento

```
float height(vec2 position, float time) {
  float d = wave((position + time) * 0.4) * 0.3;
  d += wave((position - time) * 0.3) * 0.3;
  d += wave((position + time) * 0.5) * 0.2;
  d += wave((position - time) * 0.6) * 0.2;
  return d;
}
```

Godot - Movimento

Observe que adicionamos tempo a dois e subtraímos dos outros dois. Isso faz com que as ondas se movam em direções diferentes criando um efeito complexo. Observe também que todas as amplitudes (o número pelo qual o resultado é multiplicado) somam 1,0. Isso mantém a onda na faixa de 0-1.