



Documentação

Criadores

- Lucas Gomes Cecchini
- Gabriel Henrique Pereira

Visão Geral

Este documento o ajudará a usar o Assets **Unity MMD Collection**.

Com ele você tem ferramentas que podem facilitar o uso de modelos **MMD** na **Unity**, além de **Shaders** customizados para isso.

Os **Shaders** foram feitos com base em estudos no pacote [MMD4Mecanim](#), sendo recriados e aprimorados em duas ferramentas. O [Shader Graph](#) e [Amplify Shader Editor](#) devido às capacidades e aplicações de cada recurso.

Ele também possui **Scripts** que moldam a interface da Unity para fazê-lo ficar semelhante às ferramentas do **MMD** e tornar mais prático e rápido a utilização. Além de outras coisas que permitem acelerar o desenvolvimento.

Instruções

Você pode obter mais informações na Playlist no YouTube:

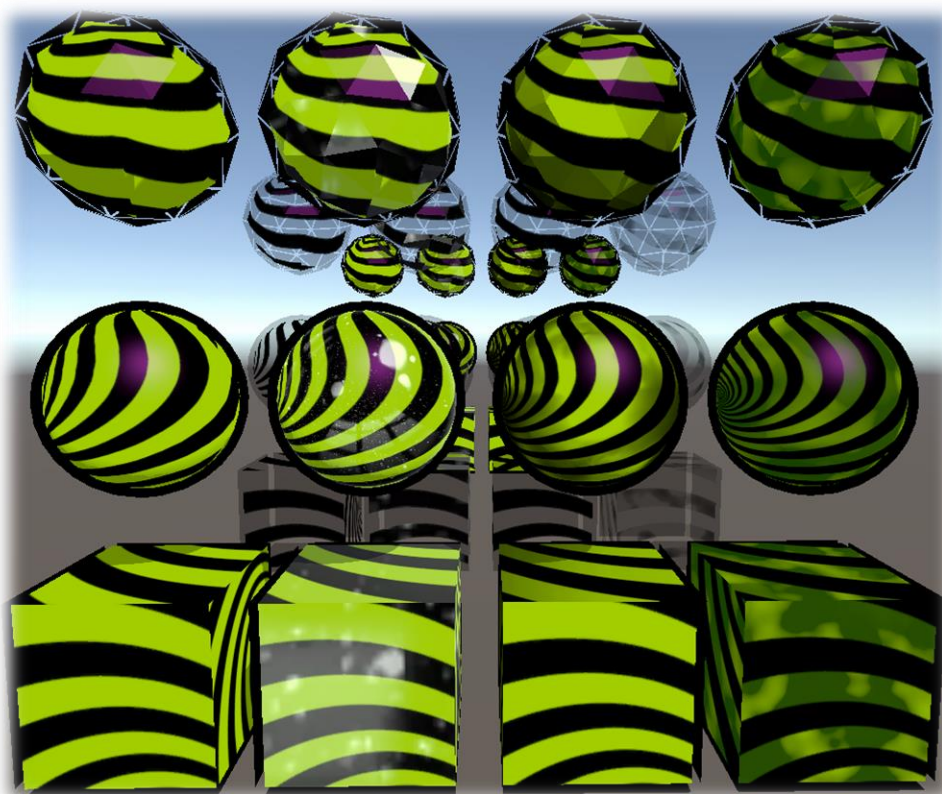
<https://youtube.com/playlist?list=PL5hnfx09yM4IWSWveW0NKCfX1Anec4dw7&si=UIqEtAoFcXZH7WFM>

Compatibilidade

Este pacote é compatível com:

- **Unity 6000.1.7f1**
- **Amplify Shader Editor v 1.9.9.1**
- **MMD4Mecanim 2020-01-05**

Explicações do Shader



MMD Shader

Add Sphere (Amplify Shad...
Add Sphere (Amplify Shader Editor) (Material)
Shader MMD Collection/URP/MMD (Amplify St Edit...

Show Default Systems: ☒

Surface Options
Surface Type Opaque
Render Face Front
Depth Write ForceEnabled
Depth Test LEqual
Alpha Clipping
Cast Shadows ☒
Receive Shadows ☒

Material Color
Diffuse
Specular
Ambient
Opaque 1
Reflection 50

Rendering
S-Shad ☒

Edge (Outline)
On ☒
Color
Size 0.1

Texture (Memo)
Effects
Texture
Tiling X 1 Y 1
Offset X 0 Y 0
Toon
SPH
UV Layer
SPH SubTex
Tiling X 1 Y 1
Offset X 0 Y 0

Custom Effects Settings
Specular Intensity 1
SPH Opacity 1
Shadow Luminescence 1.5
HDR 1
Toon Tone X 1 Y 0.5 Z 0.5 W 0
Multiple Lights ☒
Fog ☒
Render Queue Geometry+2 2002
Enable GPU Instancing
Double Sided Global Illumination
Global Illumination None

Add Sphere (Amplify Shad...
Add Sphere (Amplify Shader Editor) (Material)
Shader MMD Collection/URP/MMD (Amplify St Edit...

Show Default Systems: ☐

Mat-Name: (JP) 球体の追加(Amplify シ) (EN) Add Sphere (Amplify S

Material Color
Diffuse: R 1 G 0 B 0
Specular: R 0.6 G 0.2 B 0.6
Ambient: R 0 G 1 B 0
Opaque: 1
Reflection: 50

Rendering
2-SIDE: ☐ G-SHAD: ☒
S-MAP: ☒ S-SHAD: ☒

Edge (Outline)
On: ☒ Size: 0.1

Texture/Memo
Effects: + Add-Sphere
Texture:
Toon:
SPH:
Memo: AGAMENOM

Custom Effects Settings
Specular Intensity 1
SPH Opacity 1
Shadow Luminescence 1.5
HDR 1
Toon Tone 1 0.5 0.5
Multiple Lights ☒
Fog ☒

Surface Options
Surface Type Opaque
Render Face Front
Depth Write ForceEnabled
Depth Test LEqual
Alpha Clipping
Cast Shadows ☒
Receive Shadows ☒

Advanced Options
Render Queue Geometry+2 2002
Enable GPU Instancing
Double Sided Global Illuminir
Global Illumination None

Add Sphere (Amplify Shader Editor)
Add Sphere (Amplify Shader Editor)

AssetBundle None None AssetBundle None None

Propriedades do Shader

==Propriedades de Sistemas==

- **Show Default Systems** = Mostra às propriedades padrões ou customizadas.
- **Mat-Name (JP)** = Nome do material em Japonês.
- **Mat-Name (EN)** = Nome do material em Inglês.
- **Memo** = Cabeçalho de texto para anotações.

==Material Color==

- **Diffuse** = Cor do material.
 - **_Color ("Diffuse", Color)**: Define a cor principal do material quando iluminado diretamente.
- **Specular** = Cor do reflexo da luz/brilho.
 - **_Specular ("Specular", Color)**: Define a cor dos reflexos especulares no material.
- **Ambient** = Afeta a cor e a iluminação do modelo, similar a um raycast.
 - **_Ambient ("Ambient", Color)**: Define a cor com luz ambiente, contribuindo para a iluminação geral.
- **Opaque** = Valor alfa (opacidade)
 - **_Opaque ("Opaque", Range(0, 1))**: Controla a opacidade do material.
- **Reflection** = Valor de reflexão
 - **_Shininess ("Reflection", Float)**: Define a intensidade do brilho especular (reflexão) do material.

==Rendering==

- **2-SIDE** = Renderiza ambos os lados da malha [**Equivalente a Render Face/_Cull**]
 - **_Cull ("Render Face", Float)**: Controla quais faces da malha (frente, verso ou ambas) devem ser renderizadas.
- **G-SHAD** = Sombra no chão [**Equivalente a Cast Shadows/ShaderPass"SHADOWCASTER"**]
 - **_CastShadows ("Cast Shadows", Float)**: Define se o material projeta sombras no chão.
- **S-MAP** = Sombra na malha (incluindo a própria malha) [**Equivalente a Receive Shadows/_ReceiveShadows/Keyword"_RECEIVE_SHADOWS_OFF"**]
 - **_ReceiveShadows ("Receive Shadows", Keyword)**: Controla se a malha recebe sombras de outras fontes.
- **S-SHAD** = Recebe sombra apenas de si mesmo
 - **_SShad ("S-SHAD", Float)**: Controla a sombra que a malha projeta apenas sobre si mesma.

==Edge (Outline)==

- **On** = Ativa o contorno
 - **_On ("On", Float)**: Controla a ativação do contorno (borda).
- **Color** = Cor do contorno, incluindo transparências
 - **_OutlineColor ("Color", Color)**: Define a cor do contorno ao redor da malha.
- **Size** = Tamanho/distância do contorno
 - **_EdgeSize ("Size", Float)**: Controla a espessura do contorno.

==Texture/Memo==

- **Texture** = Textura do material
 - **_MainTex ("Texture", 2D)**: Textura principal aplicada ao material.
- **Toon** = Complementa o shading do objeto
 - **_ToonTex ("Toon", 2D)**: Textura toon, usada para shading estilo cartoon.
- **UV Layer** = Camada de UV a ser usada
 - **_UVLayer ("UV Layer", Float)**: Seleciona qual camada de UV será aplicada à textura.
- **SPH** = Reflexão artificial para complementar o Specular
 - **_SphereCube ("SPH", CUBE)**: Textura cúbica usada para reflexões artificiais.

==Effects==

- **Disabled** = Não faz nada
 - **_EFFECTS ("Effects", Float)**: Desativa os efeitos.
- **Multi-Sphere** = Multiplica um mapa de esfera brilhante (reflexão metálica)
 - **_EFFECTS ("Effects", Float)**: Ativa o efeito de múltiplas esferas brilhantes.
- **Add-Sphere** = Cria um mapa de esfera brilhante (reflexão prática)
 - **_EFFECTS ("Effects", Float)**: Ativa o efeito de adição de esferas brilhantes.
- **Sub-Tex** = Adiciona uma camada de textura UV extra para efeitos mais complexos
 - **_EFFECTS ("Effects", Float)**: Troca o uso da textura cúbica para usar uma subtextura, aplicando-a a outra camada UV.

==Configurações de Efeitos Personalizados==

- **Specular Intensity** = Define a intensidade do brilho especular
 - **SpecularIntensity ("Specular Intensity", Range(0, 1))**: Intensidade do brilho especular.
- **SPH Opacity** = Define a opacidade da Textura cúbica
 - **SPHOpacity ("SPH Opacity", Range(0, 1))**: Opacidade da textura cúbica.
- **Shadow Luminescence** = Intensidade da sombra
 - **_ShadowLum ("Shadow Luminescence", Range(0, 10))**: Luminescência das sombras.
- **HDR** = Faz o objeto brilhar no escuro
 - **_HDR ("HDR", Range(1, 1000))**: Controla o mapeamento HDR.
- **Toon Tone** = Ajusta o tom da sombra Toon
 - **_ToonTone ("Toon Tone", Vector)**: Define o tom do shading toon.
- **Multiple Lights** = Permite o objeto receber múltiplas luzes
 - **_MultipleLights ("Multiple Lights", Float)**: Ativa ou desativa o suporte para múltiplas luzes.
- **Fog** = Liga e desliga a neblina
 - **_Fog ("Fog", Float)**: Ativa ou desativa o suporte para neblina.

O **"Surface Options"** e **"Advanced Options"** são padrões da Unity que exigem um conhecimento prévio de suas funcionalidades.

Bugs conhecidos

Decal não aparece na Build: Para resolver isso, adicione o módulo de Decal a todos os 'Universal Renderer Data' no seu projeto.

Sombras distorcidas ao usar Morph: Para resolver isso, habilite a opção 'Legacy Blend Shape

Normals' no **.FBX** (ou modelo 3D) ao importar do Unity.

Luz atravessando objetos: Adicione um 'Directional Light' com pelo menos 0,001 de Intensity para atualizar o Dynamic Lightmap

Explicações dos Scripts

Free Camera

Este script define uma classe **FreeCamera** em C# para ser usada em um projeto Unity. Ele permite que a câmera se mova livremente no espaço 3D, respondendo a entradas do teclado e do mouse. A seguir está uma explicação detalhada do script:

Declarações e Atributos da Classe

1. Dependências e Configurações Iniciais:

```
[RequireComponent(typeof(Camera))]  
[AddComponentMenu("MMD Collection/Free Camera")]  
public class FreeCamera : MonoBehaviour
```

- **RequireComponent(typeof(Camera))**: Garante que o objeto ao qual este script é anexado tenha um componente **Camera**.
- **AddComponentMenu("MMD Collection/Free Camera")**: Adiciona este script ao menu de componentes, sob o caminho especificado.

2. Variáveis de Configuração da Câmera:

```
[Header("Camera Settings")]  
[SerializeField] private float movementSpeed = 10f;  
[SerializeField] private float fastMovementSpeed = 100f;  
[SerializeField] private float sensitivity = 3f;  
[SerializeField] private float zoomSensitivity = 10f;  
[SerializeField] private float fastZoomSensitivity = 50f;
```

- **movementSpeed**: Velocidade padrão de movimento da câmera.
- **fastMovementSpeed**: Velocidade aumentada de movimento quando o modo rápido está ativado.
- **sensitivity**: Sensibilidade do mouse para rotação.
- **zoomSensitivity** e **fastZoomSensitivity**: Sensibilidade para o zoom (normal e rápido).

3. Configurações de Teclas:

```
[Header("Key Settings")]
public KeyCode left = KeyCode.A;
public KeyCode right = KeyCode.D;
public KeyCode front = KeyCode.W;
public KeyCode back = KeyCode.S;
[Space(10)]
public KeyCode up = KeyCode.Q;
public KeyCode down = KeyCode.E;
[Space(10)]
public KeyCode upGlobal = KeyCode.R;
public KeyCode downGlobal = KeyCode.F;
[Space(10)]
public KeyCode observe = KeyCode.Mouse1;
[Space(10)]
public KeyCode run = KeyCode.LeftShift;
```

- Define teclas para mover a câmera em diferentes direções e alternar modos.

4. Configurações de Eixos:

```
[Header("Axis Settings")]
public string zoom = "Mouse ScrollWheel";
public string axisX = "Mouse X";
public string axisY = "Mouse Y";
```

- Define os eixos do mouse para zoom e rotação.

5. Variável Privada:

```
private bool looking = false;
```

- `looking`: Indica se a câmera está em modo de observação (rotação livre com o mouse).

Método `Update`

6. Movimento da Câmera:


```

var fastMode = Input.GetKey(run);
var currentMovementSpeed = fastMode ? fastMovementSpeed : movementSpeed;

Vector3 moveDirection = Vector3.zero;
if (Input.GetKey(left)) moveDirection += -transform.right;
if (Input.GetKey(right)) moveDirection += transform.right;
if (Input.GetKey(front)) moveDirection += transform.forward;
if (Input.GetKey(back)) moveDirection += -transform.forward;
if (Input.GetKey(up)) moveDirection += transform.up;
if (Input.GetKey(down)) moveDirection += -transform.up;
if (Input.GetKey(upGlobal)) moveDirection += Vector3.up;
if (Input.GetKey(downGlobal)) moveDirection += Vector3.down;

transform.position += currentMovementSpeed * Time.deltaTime * moveDirection;

```

- Verifica se o modo rápido está ativado e ajusta a velocidade de movimento.
- Calcula a direção do movimento baseado nas teclas pressionadas.
- Atualiza a posição da câmera.

7. Rotação da Câmera:

```

if (looking)
{
    float newRotationX = transform.localEulerAngles.y + Input.GetAxis(axisX) * sensitivity;
    float newRotationY = transform.localEulerAngles.x - Input.GetAxis(axisY) * sensitivity;
    transform.localEulerAngles = new Vector3(newRotationY, newRotationX, 0f);
}

```

- Ajusta a rotação da câmera com base no movimento do mouse se o modo de observação estiver ativo.

8. Zoom da Câmera:

```

float zoomAxis = Input.GetAxis(zoom);
if (zoomAxis != 0)
{
    var currentZoomSensitivity = fastMode ? fastZoomSensitivity : zoomSensitivity;
    transform.position += zoomAxis * currentZoomSensitivity * transform.forward;
}

```

- Ajusta o zoom da câmera baseado na rolagem do mouse.

9. Modo de Observação:

```

if (Input.GetKeyDown(observe))
{
    StartLooking();
}
else if (Input.GetKeyUp(observe))
{
    StopLooking();
}

```

Métodos Auxiliares

10. Método `OnDisable`:

```
private void OnDisable()
{
    StopLooking();
}
```

- Garante que o modo de observação seja desativado quando o script é desabilitado.

11. Métodos para Iniciar e Parar o Modo de Observação:

```
public void StartLooking()
{
    looking = true;
    Cursor.visible = false;
    Cursor.lockState = CursorLockMode.Locked;
}

public void StopLooking()
{
    looking = false;
    Cursor.visible = true;
    Cursor.lockState = CursorLockMode.None;
}
```

- **StartLooking**: Ativa o modo de observação, escondendo e travando o cursor.
- **StopLooking**: Desativa o modo de observação, mostrando e liberando o cursor.

Resumo

Este script fornece um controle detalhado e personalizável para movimentação e rotação de uma câmera em um ambiente 3D, usando entradas do teclado e do mouse. Ele permite um movimento suave e intuitivo, semelhante ao de um jogo FPS, e inclui funcionalidades para acelerar o movimento e o zoom, bem como alternar entre modos de observação e movimento normal.

Copy Animation

Este script Unity, chamado CopyAnimation, é um componente responsável por copiar rotações de um conjunto de ossos fonte para um conjunto de ossos alvo em tempo real. Abaixo está uma explicação detalhada de cada parte do script:

Definições de Componentes e Variáveis

1. Imports e Atributos de Classe

```
using UnityEngine;

[AddComponentMenu("MMD Collection/Copy Animation")]
public class CopyAnimation : MonoBehaviour
```

- `using UnityEngine;`: Importa o namespace UnityEngine, necessário para usar as classes e métodos do Unity.
- `[AddComponentMenu("MMD Collection/Copy Animation")]`: Adiciona este script ao menu de componentes do Unity sob a categoria "MMD Collection".

2. Variáveis Públicas e Privadas

```
[Header("Settings")]
public bool update = true;
[Header("Copy and Paste Rotations")]
[SerializeField] private Transform[] copyBone;
[Space(10)]
[SerializeField] private Transform[] pasteBone;
```

- `public bool update`: Controla se as rotações devem ser atualizadas ou não.
- `[SerializeField] private Transform[] copyBone`: Array de ossos fonte dos quais as rotações serão copiadas.
- `[SerializeField] private Transform[] pasteBone`: Array de ossos alvo para onde as rotações serão coladas.

Método `LateUpdate`

3. Método `LateUpdate`

```

private void LateUpdate()
{
    if (!update) return;
    if (copyBone.Length != pasteBone.Length)
    {
        Debug.LogError("Copy and paste arrays must be of the same length!", this);
        return;
    }

    for (int i = 0; i < copyBone.Length; i++)
    {
        if (copyBone[i] != null && pasteBone[i] != null)
        {
            pasteBone[i].rotation = copyBone[i].rotation;
        }
        else
        {
            Debug.LogWarning("Copy or paste transform is null!", this);
        }
    }
}

```

- `if (!update) return;`: Se `update` for `false`, o método retorna e não faz nada.
- Verifica se os arrays `copyBone` e `pasteBone` têm o mesmo comprimento. Se não, exibe um erro no console e retorna.
- Itera por cada par de ossos, copiando a rotação do osso fonte para o osso alvo.

Considerações Finais

- Este script é útil em animações, especialmente em sistemas de rigging e animações esqueléticas, onde a rotação de certos ossos precisa ser sincronizada ou limitada.
- O uso de arrays para `copyBone` e `pasteBone` permite a cópia de múltiplos ossos de uma só vez, garantindo que a hierarquia esquelética mantenha as rotações corretas.

Draw Mesh Instanced

Este script é um componente do Unity que desenha instâncias de malhas (meshes) usando a técnica de instanciamento gráfico.

Imports e Declaração da Classe

```
using System.Collections.Generic;
using UnityEngine;

[AddComponentMenu("MMD Collection/Draw Mesh Instanced")]
public class DrawMeshInstanced : MonoBehaviour
{
```

- O script começa importando namespaces necessários: `'System.Collections.Generic'` para manipulação de listas e `'UnityEngine'` para acesso à API do Unity.
- A classe `'DrawMeshInstanced'` herda de `'MonoBehaviour'`, permitindo que seja anexada a um GameObject no Unity.
- O atributo `'[AddComponentMenu("MMD Collection/Draw Mesh Instanced")]'` adiciona essa classe ao menu de componentes no editor do Unity.

Variáveis de Configuração

```
[Header("Settings")]
[SerializeField] private bool OnDrawSelected = false;
[SerializeField] private bool reuseMaterials = true;
[Space(10)]
[SerializeField] private List<DrawMeshInstancedList> drawMeshInstancedLists = new();
```

- `'OnDrawSelected'`: Se verdadeiro, desenha as instâncias da malha apenas quando o objeto está selecionado no editor.
- `'reuseMaterials'`: Se verdadeiro, reutiliza os materiais para submeshes.
- `'drawMeshInstancedLists'`: Uma lista de parâmetros para instanciar as malhas.

Método 'Update'

```
private void Update()
{
    DrawMesh();
}
```

- O método `'Update'` é chamado a cada frame. Ele chama o método `'DrawMesh'` para desenharmos as instâncias da malha.

Método 'DrawMesh'

```

private void DrawMesh()
{
    foreach (var drawList in drawMeshInstancedLists)
    {
        if (drawList.mesh != null && drawList.materials != null && drawList.materials.Length > 0)
        {
            for (int i = 0; i < drawList.mesh.subMeshCount; i++)
            {
                Material materialToUse = drawList.materials[i % drawList.materials.Length];

                if (!reuseMaterials && i >= drawList.materials.Length)
                {
                    break;
                }

                Matrix4x4[] matrices = new Matrix4x4[] { drawList.transform.localToWorldMatrix };
                Graphics.DrawMeshInstanced(drawList.mesh, i, materialToUse, matrices);
            }
        }
    }
}

```

- Este método itera através de `drawMeshInstancedLists`.
- Para cada item na lista, verifica se a malha e os materiais são válidos.
- Para cada submalha (`subMesh`), seleciona o material apropriado.
- Se `reuseMaterials` for falso e todos os materiais já tiverem sido usados, a iteração é interrompida.
- Cria uma matriz de transformação e chama `Graphics.DrawMeshInstanced` para desenhar a instância da malha.

Método `OnDrawGizmosSelected`

```

private void OnDrawGizmosSelected()
{
    if (OnDrawSelected)
    {
        DrawMesh();
    }

    Gizmos.color = Color.green;
    Gizmos.DrawSphere(transform.position, 0.15f);

    if (drawMeshInstancedLists.Count > 0)
    {
        foreach (var drawList in drawMeshInstancedLists)
        {
            if (drawList.transform != null)
            {
                Gizmos.color = Color.red;
                Gizmos.DrawSphere(drawList.transform.position, 0.15f);
            }
        }
    }
}

```

- Este método é chamado quando o objeto é selecionado no editor do Unity.
- Se `OnDrawSelected` for verdadeiro, chama `DrawMesh`.
- Desenha esferas verdes na posição do objeto e esferas vermelhas nas posições dos `Transform` de `drawMeshInstancedLists`.

Classe `DrawMeshInstancedList`

```
[System.Serializable]
public class DrawMeshInstancedList
{
    public Transform transform;
    [Space(5)]
    public Mesh mesh;
    [Space(5)]
    [Tooltip("Materials must support 'Enable GPU Instancing'")]
    public Material[] materials;
}
```

- Esta classe armazena os parâmetros necessários para instanciar uma malha.
- **transform**: Define a posição, rotação e escala das instâncias.
- **mesh**: A malha a ser instanciada.
- **materials**: Os materiais aplicados às instâncias. Os materiais devem suportar o recurso "Enable GPU Instancing".

Resumo

O script **DrawMeshInstanced** permite desenhar múltiplas instâncias de uma malha com diferentes materiais e posições definidas por transformações. Ele também fornece a funcionalidade de desenhar essas instâncias apenas quando o objeto é selecionado no editor do Unity. A classe auxiliar **DrawMeshInstancedList** facilita a configuração desses parâmetros no editor.

Visão Geral

Ele define um `ScriptableObject` chamado `CustomMMDData` para armazenar dados personalizados relacionados ao MMD (MikuMikuDance), que é um software de animação 3D popular no Japão.

Detalhamento do Código

Namespace e Bibliotecas

```
using System.Collections.Generic;
using UnityEngine;
```

- `using System.Collections.Generic;`: Importa a biblioteca para utilizar coleções genéricas, como listas.
- `using UnityEngine;`: Importa a biblioteca principal do Unity que contém funcionalidades essenciais para o desenvolvimento de jogos.

Definição do ScriptableObject

```
public class CustomMMDData : ScriptableObject
{
    [Header("MMD Material Settings")]
    public bool showSystemsDefault;
    public List<MMDMaterialInfo> materialInfoList = new();
}
```

- `public class CustomMMDData : ScriptableObject`: Define uma classe que herda de `ScriptableObject`. `ScriptableObject` é uma maneira conveniente de armazenar grandes conjuntos de dados que podem ser facilmente editados no editor do Unity e usados em diferentes partes de um jogo.
- `[Header("MMD Material Settings")]`: Adiciona um cabeçalho no Inspetor do Unity para organizar e identificar melhor os campos.
- `public bool showSystemsDefault;`: Um campo booleano que indica se deve mostrar mais sistemas de shaders (padrões do sistema).
- `public List<MMDMaterialInfo> materialInfoList = new();`: Uma lista para armazenar informações sobre materiais MMD. A lista é inicializada como uma nova lista vazia.

Classe Serializable

```
[System.Serializable]
public class MMDMaterialInfo
{
    public Material mmdMaterial;
    public string materialNameJP;
    public string materialNameEN;
    public string materialMeno;
}
```


- `[System.Serializable]`: Permite que a classe seja serializável, ou seja, seus dados podem ser exibidos e editados no Inspetor do Unity.
- `public class MMDMaterialInfo`: Define uma classe para armazenar informações sobre um material MMD específico.
- `public Material mmdMaterial`: Referência a um material do Unity.
- `public string materialNameJP`: Nome do material em japonês.
- `public string materialNameEN`: Nome do material em inglês.
- `public string materialMeno`: Campo para notas adicionais sobre o material.

Funcionamento Geral

1. CustomMMDData:

- É um `ScriptableObject` que armazena uma lista de objetos `MMDMaterialInfo` e uma flag booleana (`showSystemsDefault`).
- Pode ser usado para organizar e gerenciar dados de materiais MMD no Unity.

2. MMDMaterialInfo:

- Armazena detalhes sobre um material específico, incluindo referências ao material no Unity e seus nomes em japonês e inglês, além de um campo para notas.

Utilização no Unity

- Esse `ScriptableObject` pode ser criado e editado diretamente no editor do Unity, permitindo fácil gerenciamento e organização dos dados de materiais MMD.
- Os desenvolvedores podem adicionar, remover e modificar entradas na lista `materialInfoList` através do Inspetor do Unity.
- A flag `showSystemsDefault` pode ser usada para alternar configurações específicas de exibição ou comportamento dentro do jogo ou ferramenta.

Custom MMD Data Utility Editor

Esse script em C# é usado no Unity Editor para gerenciar assets do tipo `CustomMMDDData`.

Visão Geral

Este script fornece uma classe utilitária estática chamada `CustomMMDDDataUtilityEditor`, que inclui métodos para encontrar, criar e gerenciar assets `CustomMMDDData` dentro do Unity Editor.

Detalhamento do Código

Namespace e Bibliotecas

```
using System.Collections.Generic;
using System.IO;
using UnityEditor;
using UnityEngine;
```

- `using System.Collections.Generic;`: Importa a biblioteca para utilizar coleções genéricas, como listas.
- `using System.IO;`: Importa funcionalidades de entrada e saída, como manipulação de arquivos e diretórios.
- `using UnityEditor;`: Importa funcionalidades específicas do editor do Unity.
- `using UnityEngine;`: Importa a biblioteca principal do Unity.

Classe Utilitária

```
// Utility class for managing CustomMMDDData assets within the Unity Editor.
public static class CustomMMDDDataUtilityEditor
{
    // Retrieves or creates the CustomMMDDData asset.
    public static CustomMMDDData GetOrCreateCustomMMDDData()
    {
        // Attempt to find an existing CustomMMDDData asset.
        CustomMMDDData customMMDDData = FindCustomMMDDData();

        // If not found, create a new one.
        #pragma warning disable IDE0270
        if (customMMDDData == null)
        {
            customMMDDData = CreateCustomMMDDData();
        }
        #pragma warning restore IDE0270

        return customMMDDData;
    }
}
```

- `public static class CustomMMDDDataUtilityEditor`: Define uma classe estática para métodos utilitários relacionados a `CustomMMDDData`.
- `public static CustomMMDDData GetOrCreateCustomMMDDData()`: Método que recupera ou cria um asset `CustomMMDDData`.

- Tenta encontrar um asset ``CustomMMDData`` existente chamando ``FindCustomMMDData()``.
- Se não encontrar, cria um novo chamando ``CreateCustomMMDData()``.

Método para Encontrar um Asset Existente

```
// Finds an existing CustomMMDData asset in the project.
private static CustomMMDData FindCustomMMDData()
{
    string[] guids = AssetDatabase.FindAssets("Custom MMD Data t:CustomMMDData");

    // If found, load the first asset.
    if (guids.Length > 0)
    {
        string path = AssetDatabase.GUIDToAssetPath(guids[0]);
        return AssetDatabase.LoadAssetAtPath<CustomMMDData>(path);
    }

    return null; // No existing asset found.
}
```

- ``private static CustomMMDData FindCustomMMDData()``: Método que encontra um asset ``CustomMMDData`` existente no projeto.
- Utiliza ``AssetDatabase.FindAssets`` para procurar por assets do tipo ``CustomMMDData``.
- Se encontrar, carrega o primeiro asset encontrado usando ``AssetDatabase.LoadAssetAtPath``.

Método para Criar um Novo Asset

```
// Creates a new CustomMMDData asset.
private static CustomMMDData CreateCustomMMDData()
{
    string folderPath = "Assets/Resources"; // Set folder path.

    // Check if the folder exists; if not, create it.
    if (!Directory.Exists(folderPath))
    {
        Directory.CreateDirectory(folderPath);
    }

    string assetPath = AssetDatabase.GenerateUniqueAssetPath(folderPath + "/Custom MMD Data.asset");
    CustomMMDData customMMDData = ScriptableObject.CreateInstance<CustomMMDData>(); // Create a new instance of CustomMMDData.

    // Create the asset and save it.
    AssetDatabase.CreateAsset(customMMDData, assetPath);
    AssetDatabase.SaveAssets();
    AssetDatabase.Refresh();

    return customMMDData; // Return the newly created asset.
}
```

- ``private static CustomMMDData CreateCustomMMDData()``: Método que cria um novo asset ``CustomMMDData``.
- Define o caminho da pasta onde o asset será armazenado.
- Verifica se a pasta existe e, se não existir, cria a pasta.
- Gera um caminho único para o asset e cria uma nova instância de ``CustomMMDData``.
- Cria o asset no caminho especificado, salva e atualiza o banco de dados de assets.

Método para Remover Materiais Inválidos

```
// Removes any invalid materials from the CustomMMDData asset.
public static void RemoveInvalidMaterials(CustomMMDData customMMDDMaterialData)
{
    // Ensure the asset is not null.
    if (customMMDDMaterialData == null)
    {
        return; // Exit if the asset is null.
    }

    List<MMDMaterialInfo> validMaterials = new(); // Create a list to store valid materials.

    // Iterate through each material info in the asset.
    foreach (MMDMaterialInfo materialInfo in customMMDDMaterialData.materialInfoList)
    {
        // Check if the material reference is not null.
        if (materialInfo.mmdMaterial != null)
        {
            validMaterials.Add(materialInfo); // Add to the valid materials list.
        }
    }

    customMMDDMaterialData.materialInfoList = validMaterials; // Replace the material info list with the list of valid materials.
}
```

- **public static void RemoveInvalidMaterials(CustomMMDData customMMDDMaterialData)**: Método que remove materiais inválidos do asset **CustomMMDData**.
- Verifica se o asset é nulo e sai do método se for.
 - Cria uma nova lista para armazenar materiais válidos.
 - Itera sobre cada **MMDMaterialInfo** na lista **materialInfoList** e adiciona à lista de materiais válidos apenas se a referência ao material não for nula.
 - Substitui a lista **materialInfoList** com a lista de materiais válidos.

Funcionamento Geral

1. Criação ou Recuperação de **CustomMMDData**:

- O método **GetOrCreateCustomMMDData** tenta encontrar um asset **CustomMMDData** existente.
- Se não encontrar, cria um novo asset.

2. Encontrar **CustomMMDData**:

- O método **FindCustomMMDData** procura por assets do tipo **CustomMMDData** no projeto.
- Se encontrar, carrega e retorna o primeiro asset encontrado.

3. Criar Novo **CustomMMDData**:

- O método **CreateCustomMMDData** cria um novo asset **CustomMMDData** em uma pasta específica.
- Se a pasta não existir, cria a pasta e então cria o asset.

4. Remover Materiais Inválidos:

- O método **RemoveInvalidMaterials** remove entradas na lista **materialInfoList** que possuem referências a materiais nulas.

Esse script é muito útil para garantir que os assets **CustomMMDData** estejam sempre válidos e facilmente acessíveis dentro do Unity Editor, facilitando o gerenciamento de dados de materiais MMD no projeto.

Explicação Detalhada do Script

Este script é uma extensão personalizada para o editor de materiais no Unity, fornecendo funcionalidades avançadas para inspeção e modificação de materiais que utilizam shaders personalizados. Ele estende a classe ``ShaderGUI``, permitindo controle detalhado sobre várias propriedades dos shaders e materiais diretamente no editor.

Classe ``CustomInspectorUtilityEditor``

A classe ``CustomInspectorUtilityEditor`` herda de ``ShaderGUI`` e serve como a base para criar inspetores personalizados de shaders. Ela oferece várias funções estáticas e métodos para carregar, salvar e renderizar propriedades do material de forma dinâmica e interativa.

Variáveis Principais

- ``private Material currentMaterial``: Armazena uma referência ao material atualmente sendo inspecionado.
- ``private CustomMMDMaterialData customMMDMaterialData``: Um objeto que armazena dados específicos do material, como nomes e outros metadados.
- ``private bool showSystemsDefault``: Um booleano que determina se as configurações padrão do sistema devem ser exibidas.

Método ``LoadData``

Este método carrega os dados existentes do material atual (``currentMaterial``) e os armazena nas variáveis relevantes. Ele verifica se o objeto ``customMMDMaterialData`` existe e, em caso afirmativo, carrega informações como os nomes em japonês e inglês do material (``materialNameJP``, ``materialNameEN``), a descrição (``materialMemo``), e a configuração sobre a exibição dos valores padrão do sistema (``showSystemsDefault``).

Método ``SaveData``

Este método salva os dados do material atual. Ele utiliza o método ``DetectChanges`` para verificar se houve modificações nos dados do material. Caso haja mudanças, ele atualiza as informações e marca o objeto como "sujo" (``SetDirty``), indicando que há alterações a serem salvas. Finalmente, ele persiste essas mudanças no sistema de assets do Unity.

Método ``DetectChanges``

Este método compara os valores atuais das propriedades do material com os valores armazenados anteriormente em ``customMMDMaterialData``. Ele retorna ``true`` se houver alterações e ``false`` caso contrário, permitindo que o sistema saiba quando salvar as mudanças.

Métodos de Renderização

Estes métodos são responsáveis por renderizar as várias propriedades do material no inspetor do Unity. Eles permitem que o desenvolvedor visualize e modifique atributos específicos do shader diretamente na interface do Unity.

- ``RenderSurfaceOptions``: Renderiza opções de superfície, como tipo de superfície, modo de blending, culling, entre outras.
- ``RenderLightmapFlags``: Renderiza opções de lightmap, incluindo realtime, baked, e emissive.
- ``RenderColorProperty``: Renderiza uma propriedade de cor com um rótulo associado.
- ``RenderSliderFloatProperty``: Renderiza uma propriedade do tipo ``float`` como um slider.
- ``RenderDoubleSidedToggle``: Renderiza um toggle para definir se a renderização será de dupla face.

- **RenderShaderPassToggle**: Renderiza um toggle para habilitar ou desabilitar um **pass** específico do shader.
- **RenderKeywordToggle**: Renderiza um toggle para ativar ou desativar uma **keyword** específica.
- **RenderUIToggle**: Renderiza um toggle para propriedades de UI.
- **RenderUIColorProperty**: Renderiza uma propriedade de cor específica para UI.
- **RenderFloatProperty**: Renderiza uma propriedade do tipo **float** com um rótulo.
- **RenderVector4Property**: Renderiza uma propriedade do tipo **Vector4**.
- **RenderDropdownProperty**: Renderiza uma propriedade do tipo dropdown, verificando se as opções e os valores correspondem. Caso contrário, lança uma exceção.
- **RenderTextureProperty**: Renderiza uma propriedade do tipo textura com um rótulo e exibe opcionalmente os campos de **Tiling** e **Offset**.
- **RenderCubemapProperty**: Renderiza uma propriedade do tipo cubemap com um rótulo.
- **RenderVector3Property**: Renderiza uma propriedade do tipo **Vector3**, ajustando cada componente separadamente.
- **RenderDepthWriteDropdown**: Renderiza um dropdown para opções de escrita de profundidade (Depth Write).
- **RenderBlendingModeDropdown**: Renderiza um dropdown para opções de modo de blending, atualizando automaticamente as propriedades de **srcBlend** e **dstBlend** com base na seleção.
- **RenderSurfaceTypeDropdown**: Renderiza um dropdown para opções de tipo de superfície (opaco ou transparente), configurando as tags de renderização do material adequadamente.

Métodos Auxiliares

- **IsToggleUIPropertyEnabled**: Verifica se uma propriedade de toggle de UI está habilitada, baseado no valor da propriedade de float (1.0 = habilitado).
- **HasFloatPropertyValue**: Verifica se uma propriedade **float** possui um valor específico, comparando diretamente o valor atual da propriedade.
- **CheckBlendingMode**: Verifica e ajusta o modo de blending com base nas propriedades atuais do material. Esta função garante que as propriedades de blending estejam configuradas corretamente, dependendo do modo de superfície.

Conclusão

Este script **CustomInspectorUtilityEditor** oferece uma interface robusta e intuitiva para personalizar e gerenciar materiais que utilizam shaders avançados no Unity. Ele permite que desenvolvedores e artistas tenham controle detalhado sobre como os materiais são renderizados, oferecendo uma experiência de edição rica e flexível diretamente no editor do Unity. Com ele, é possível configurar desde propriedades simples, como cores e texturas, até opções avançadas como blending e depth write.

Explicação Detalhada dos Scripts

As duas classes, `MMDMaterialCustomInspector_AmplifyShaderEditor`` e `MMDMaterialCustomInspector_ShaderGraph``, são scripts de inspetor customizado para materiais do MikuMikuDance (MMD) no Unity, utilizando o Amplify Shader Editor. Ele estende a interface do Unity Editor, permitindo que os usuários manipulem propriedades específicas dos materiais MMD com facilidade, oferecendo tanto opções padrão quanto customizadas.

Principais Funcionalidades:

1. Variáveis de Controle:

- Armazena nomes de materiais em japonês e inglês, além de um campo de memo para anotações.
- A variável `showSystemsDefault`` controla a exibição do inspetor customizado ou do inspetor padrão do Unity.
- `customMMDMaterialData`` é utilizado para armazenar e gerenciar dados específicos dos materiais MMD.

2. Método `OnGUI``:

- Inicializa o inspetor de materiais (`materialInspector``), propriedades do material (`materialProperties``), e o material atual sendo editado (`currentMaterial``).
- Apresenta uma opção de alternância para mostrar ou ocultar o inspetor padrão do Unity.
- Carrega os dados customizados do material MMD, se ainda não estiverem carregados.
- Se `showSystemsDefault`` estiver desativado, renderiza o inspetor customizado, caso contrário, renderiza o inspetor padrão do Unity.
- Salva automaticamente as alterações feitas nos dados dos materiais MMD.

3. Método `RenderCustomMaterialInspector``:

- Oferece controles detalhados para o ajuste de propriedades dos materiais MMD, incluindo:
 - Nomes de materiais (em japonês e inglês).
 - Propriedades de cores como `Diffuse``, `Specular`` e `Ambient``.
 - Configurações de opacidade e reflexo.
 - Opções de renderização avançadas (como faces duplas, sombreador de sombras, recepção de sombras).
 - Configurações de borda (contorno), incluindo tamanho e cor.
 - Configurações de texturas (principal, toon e efeitos esféricos como `SPH``).
 - Um campo de memo para notas ou informações adicionais.
 - Ajustes de efeitos customizados, como intensidade especular, opacidade de sombras, HDR, entre outros.
- Opções avançadas, incluindo fila de renderização, instanciação de GPU e iluminação global de face dupla.

Ambos os scripts utilizam o utilitário personalizado `CustomMMDDataUtilityEditor`` para carregar, salvar e renderizar propriedades e dados customizados dos materiais MMD. Esse utilitário é essencial para manter o código organizado e reutilizável, além de facilitar a adição de novas funcionalidades no futuro.

Também tem uma variante do script chamado `MMDTessellationMaterialCustomInspector`` para adicionar `Edge Length``, `Phong Tess Strength`` e `Extrusion Amount`` que são fusões de Tessellation que é uma técnica usada em shaders para subdividir polígonos de uma malha em triângulos menores em tempo de execução. Isso permite adicionar mais detalhes geométricos a objetos.

Create Prefab From Model

Este script é uma ferramenta personalizada para o Unity Editor, projetada para criar prefabs a partir de modelos selecionados. Vamos detalhar cada parte do script:

Importações e Definições Iniciais

```
using UnityEngine;
using UnityEditor;
using System.Collections.Generic;
using System.IO;
```

Essas linhas importam os namespaces necessários:

- **UnityEngine**: Para acessar as funcionalidades principais do Unity.
- **UnityEditor**: Para criar ferramentas personalizadas dentro do Unity Editor.
- **System.Collections.Generic**: Para usar listas genéricas.
- **System.IO**: Para manipular caminhos de arquivos e diretórios.

Classe Principal

```
public class CreatePrefabFromModel : EditorWindow
```

Define uma classe chamada **CreatePrefabFromModel**, que herda de **EditorWindow**, permitindo criar janelas personalizadas no Unity Editor.

Método Estático para Criar Prefabs

```
[MenuItem("Assets/MMD Collection/Create Prefabs From Selected Model")]
private static void CreatePrefabsFromModel()
```

Este método cria uma entrada de menu no Unity Editor sob "Assets/MMD Collection/Create Prefabs From Selected Model". Quando essa entrada é clicada, o método **CreatePrefabsFromModel** é executado.

Seleção e Validação de Objetos

```
Object[] selectedObjects = Selection.objects;
List<GameObject> models = new();

foreach (Object obj in selectedObjects)
{
    if (obj is GameObject selectedObject && PrefabUtility.GetPrefabAssetType(selectedObject) == PrefabAssetType.Model)
    {
        string modelPath = AssetDatabase.GetAssetPath(selectedObject);
        GameObject model = AssetDatabase.LoadAssetAtPath<GameObject>(modelPath);

        if (model != null)
        {
            models.Add(model);
        }
        else
        {
            Debug.LogError($"Failed to load model from path: {modelPath}");
        }
    }
}

if (models.Count == 0)
{
    Debug.LogWarning("Select one or more valid 3D model files to create Prefabs.");
}
```

- **Selection.objects**: Obtém os objetos selecionados no Unity Editor.
- **List<GameObject> models**: Lista para armazenar modelos válidos.
- Itera pelos objetos selecionados:

- Verifica se o objeto é um `GameObject` e se é um modelo (`PrefabAssetType.Model`).
- Obtém o caminho do ativo e carrega o modelo.
- Adiciona o modelo à lista de modelos válidos, se carregado com sucesso.
- Exibe um aviso se nenhum modelo válido for encontrado.

Conversão de Modelos em Prefabs

```
foreach (GameObject model in models)
{
    ModelConverter(model);
}
```

Para cada modelo válido, chama o método `ModelConverter`.

Método `ModelConverter`

```
private static void ModelConverter(GameObject model)
{
    List<GameObject> createdObjects = new();
    SkinnedMeshRenderer[] skinnedMeshRenderers = model.GetComponentsInChildren<SkinnedMeshRenderer>();
    MeshFilter[] meshFilters = model.GetComponentsInChildren<MeshFilter>();

    if (skinnedMeshRenderers.Length == 0 && meshFilters.Length == 0)
    {
        Debug.LogError("The selected model does not contain any SkinnedMeshRenderer or MeshFilter components.");
        return;
    }

    SkinnedMeshRendererConverter(skinnedMeshRenderers, createdObjects);
    MeshFilterConverter(meshFilters, createdObjects);

    if (createdObjects.Count > 0)
    {
        CreatePrefab(model, createdObjects);
    }
}
```

- Cria uma lista para armazenar objetos criados.
- Obtém todos os componentes `SkinnedMeshRenderer` e `MeshFilter` do modelo.
- Se não houver componentes válidos, exibe um erro e retorna.
- Converte os `SkinnedMeshRenderer` e `MeshFilter` em objetos separados.
- Se objetos válidos forem criados, chama o método `CreatePrefab`.

Conversão de `SkinnedMeshRenderer` e `MeshFilter`

```

private static void SkinnedMeshRendererConverter(SkinnedMeshRenderer[] skinnedMeshRenderers, List<GameObject> createdObjects)
{
    foreach (SkinnedMeshRenderer skinnedMeshRenderer in skinnedMeshRenderers)
    {
        GameObject emptyObject = CreateGameObjectWithMesh(skinnedMeshRenderer.sharedMesh, skinnedMeshRenderer.sharedMaterials);
        createdObjects.Add(emptyObject);
    }
}

private static void MeshFilterConverter(MeshFilter[] meshFilters, List<GameObject> cre, List<GameObject> createdObjects)
{
    foreach (MeshFilter meshFilter in meshFilters)
    {
        if (meshFilter.TryGetComponent<MeshRenderer>(out var meshRenderer))
        {
            GameObject emptyObject = CreateGameObjectWithMesh(meshFilter.sharedMesh, meshFilter.sharedMaterials, meshRenderer.sharedMaterials);
            createdObjects.Add(emptyObject);
        }
        else
        {
            Debug.LogError("MeshFilter does not have a MeshRenderer.");
        }
    }
}

```

- **SkinnedMeshRendererConverter**: Converte componentes **SkinnedMeshRenderer** em novos objetos.

- **MeshFilterConverter**: Converte componentes **MeshFilter** em novos objetos, verificando se possuem um **MeshRenderer**.

Criação de GameObject com Mesh e Materiais

```

private static GameObject CreateGameObjectWithMesh(Mesh mesh, Material[] materials)
{
    GameObject emptyObject = new(mesh.name);
    MeshFilter meshFilter = emptyObject.AddComponent<MeshFilter>();
    MeshRenderer meshRenderer = emptyObject.AddComponent<MeshRenderer>();

    meshFilter.sharedMesh = mesh;
    meshRenderer.sharedMaterials = materials;

    return emptyObject;
}

```

Cria um novo **GameObject** com componentes **MeshFilter** e **MeshRenderer**, atribuindo a ele a malha e os materiais fornecidos.

Criação de Prefab

```

private static void CreatePrefab(GameObject model, List<GameObject> createdObjects)
{
    if (createdObjects == null || createdObjects.Count == 0)
    {
        Debug.LogError("No valid objects created to make a prefab.");
        return;
    }
    ...
}

```

- Verifica se há objetos válidos para criar o prefab.

- Se houver apenas um objeto, usa-o diretamente; caso contrário, cria um novo **GameObject** para agrupar todos os objetos criados.

- Obtém o caminho do diretório do modelo e constrói o caminho do prefab.

- Tenta salvar o objeto criado como um prefab.
- Exibe mensagens de log para sucesso ou erro na criação do prefab.
- Destrói o objeto criado após salvar o prefab para evitar resíduos na cena.

Resumo

Este script automatiza o processo de criação de prefabs a partir de modelos 3D selecionados no Unity Editor. Ele valida os modelos, converte componentes de malha em objetos separados e salva esses objetos como prefabs.

Paste As Child Multiple

Esse script é um editor customizado para o Unity, que adiciona a funcionalidade de colar múltiplas instâncias de um prefab ou objeto selecionado como filhos de objetos selecionados na hierarquia. Vou explicar cada parte do script detalhadamente:

Imports e Declarações

```
using UnityEngine;
using UnityEditor;
using System.Collections.Generic;
using System;
```

Esses são os namespaces necessários:

- `UnityEngine` e `UnityEditor` são essenciais para qualquer script que interaja com o Unity e seu editor.
- `System.Collections.Generic` e `System` fornecem estruturas de dados e funções básicas.

Classe Principal

```
public class PasteAsChildMultiple : EditorWindow
{
    private GameObject objectToCopy;
    private bool enumerate;
    #pragma warning disable IDE0044
    private List<GameObject> newObjects = new();
    #pragma warning restore IDE0044
```

- `PasteAsChildMultiple` herda de `EditorWindow`, permitindo criar uma janela customizada no editor do Unity.
- `objectToCopy` armazena o objeto que será copiado.
- `enumerate` define se os novos objetos receberão nomes enumerados.
- `newObjects` mantém uma lista dos novos objetos criados.

Inicialização da Janela

```
[MenuItem("GameObject/MMD Collection/Paste as Multiple Children")]
private static void Init()
{
    PasteAsChildMultiple window = (PasteAsChildMultiple)GetWindow(typeof(PasteAsChildMultiple));
    window.titleContent = new GUIContent("Paste as Multiple Children");
    window.minSize = new Vector2(350, 200);
    window.maxSize = new Vector2(350, 200);
    window.Show();
}
```

- `MenuItem` define onde a nova opção de menu será exibida no editor do Unity.
- `Init` cria e exibe a janela customizada com título e dimensões fixas.

Layout da Janela

```

private void OnGUI()
{
    GUIStyle boldLargeStyle = new(GUI.skin.label)
    {
        fontSize = 15,
        fontStyle = FontStyle.Bold
    };

    GUILayout.Label("Select Prefab to Copy", boldLargeStyle);
    GUILayout.Space(10f);

    EditorGUILayout.BeginHorizontal();
    GUILayout.Label("Object to Copy:", GUILayout.Width(100f));
    objectToCopy = EditorGUILayout.ObjectField(objectToCopy, typeof(GameObject), true) as GameObject;
    EditorGUILayout.EndHorizontal();
    GUILayout.Space(10f);

    EditorGUILayout.BeginHorizontal();
    GUILayout.Label("Enumerate:", GUILayout.Width(100f));
    enumerate = EditorGUILayout.Toggle(enumerate);
    EditorGUILayout.EndHorizontal();
    GUILayout.Space(35f);

    EditorGUI.BeginDisabledGroup(objectToCopy == null);
    GUILayout.BeginHorizontal();
    GUILayout.FlexibleSpace();
    if (GUILayout.Button("Paste As Child", GUILayout.Width(150), GUILayout.Height(40))
    {

```

- `OnGUI` define a interface gráfica da janela.
- Cria um título com estilo negrito.
- Campo para selecionar o objeto a ser copiado.
- Toggle para definir se os novos objetos terão nomes enumerados.
- Botão "Paste As Child", que chama o método `PasteAsChild` ao ser pressionado.

Método para Colar o Objeto

```
private void PasteAsChild()
{
    GameObject[] selectedObjects = Selection.gameObjects;

    if (selectedObjects.Length == 0)
    {
        Debug.LogWarning("No objects selected.");
        return;
    }

    if (objectToCopy == null)
    {
        Debug.LogWarning("No object selected to copy.");
        return;
    }
    ...
}
```

- **PasteAsChild** é o método principal que lida com a colagem dos objetos.
- Verifica se há objetos selecionados e um objeto a ser copiado.
- Itera sobre os objetos selecionados, instanciando o prefab e configurando-o como filho de cada objeto selecionado.
- Ajusta o nome do novo objeto se **enumerate** estiver habilitado.
- Registra a operação de desfazer para cada novo objeto criado.
- Expande a hierarquia para mostrar os novos filhos.
- Seleciona os novos objetos e fecha a janela.

Método para Expandir a Hierarquia

```
private void ExpandHierarchy(GameObject obj)
{
    try
    {
        var type = typeof(EditorWindow).Assembly.GetType("UnityEditor.SceneHierarchyWindow");
        var hierarchyWindow = GetWindow(type);
        var expandMethod = type.GetMethod("SetExpandedRecursive");

        if (hierarchyWindow != null && expandMethod != null)
        {
            expandMethod.Invoke(hierarchyWindow, new object[] { obj.GetInstanceID(), true });
        }
    }
    catch (Exception e)
    {
        Debug.LogError($"Error expanding hierarchy: {e.Message}");
    }
}
```

- **ExpandHierarchy** usa reflexão para acessar e invocar métodos da **SceneHierarchyWindow**, expandindo a hierarquia para mostrar os novos objetos.

Considerações Finais

Este script oferece uma ferramenta útil para duplicar e organizar objetos na hierarquia do Unity, facilitando o processo de desenvolvimento e prototipagem. A interface é simples, mas funcional, permitindo que os desenvolvedores rapidamente instanciem múltiplos objetos e configurem suas propriedades de forma eficiente.

Material Property Cleaner

Este script é um editor de Unity que limpa propriedades inválidas de materiais selecionados. Vou explicar cada parte em detalhes.

Imports

```
using UnityEngine;
using UnityEditor;
using System.Collections.Generic;
```

Esses são os namespaces importados:

- **UnityEngine**: A principal biblioteca do Unity.
- **UnityEditor**: Fornece funcionalidades específicas do editor do Unity.
- **System.Collections.Generic**: Permite o uso de coleções genéricas como **HashSet** e **List**.

Declaração da Classe

```
public class MaterialPropertyCleaner : MonoBehaviour
```

MaterialPropertyCleaner é uma classe pública que herda de **MonoBehaviour**, mas neste contexto, ela é usada principalmente para acessar funcionalidades do editor.

Menu Item e Método Principal

```
[MenuItem("Assets/MMD Collection/Clean Invalid Material Properties")]
public static void CleanMaterialProperties()
```

Este atributo adiciona um item de menu no editor do Unity. Quando clicado, ele chama o método **CleanMaterialProperties**.

Janela de Confirmação

```
if (!EditorUtility.DisplayDialog(
    "Confirm Material Clean",
    "Are you sure you want to clear invalid properties from the selected material?\nThis operation cannot be undone.",
    "Yes",
    "No"))
{
    return;
}
```

Mostra uma janela de diálogo de confirmação ao usuário. Se o usuário clicar "No", o método é encerrado.

Loop para Objetos Selecionados

```
foreach (Object selectedObject in Selection.objects)
{
    if (selectedObject is Material material)
    {
        Shader shader = material.shader;

        if (shader == null)
        {
            Debug.LogError($"The material '{material.name}' does not have a shader.");
            continue;
        }
    }
}
```

Para cada objeto selecionado no editor:

- Verifica se o objeto é um **Material**.
- Obtém o shader do material e, se não houver shader, registra um erro e continua para o próximo objeto.

Propriedades Válidas do Shader

```
var validProperties = new HashSet<string>();

for (int i = 0; i < ShaderUtil.GetPropertyCount(shader); i++)
{
    string propertyName = ShaderUtil.GetPropertyIndex(shader, i);
    validProperties.Add(propertyName);
}
```

Cria um conjunto de propriedades válidas, iterando pelas propriedades do shader e adicionando seus nomes ao conjunto `validProperties`.

Propriedades Salvas do Material

```
var materialSerializedObject = new SerializedObject(material);
var savedProperties = materialSerializedObject.FindProperty("m_SavedProperties");
```

Obtém um objeto serializado do material e acessa as propriedades salvas.

Remoção de Propriedades Inválidas

```
RemoveInvalidProperties(savedProperties.FindPropertyRelative("m_TexEnvs"), validProperties);
RemoveInvalidProperties(savedProperties.FindPropertyRelative("m_Ints"), validProperties);
RemoveInvalidProperties(savedProperties.FindPropertyRelative("m_Floats"), validProperties);
RemoveInvalidProperties(savedProperties.FindPropertyRelative("m_Colors"), validProperties);

materialSerializedObject.ApplyModifiedProperties();
```

Chama o método `RemoveInvalidProperties` para diferentes tipos de propriedades (`m_TexEnvs`, `m_Ints`, `m_Floats`, `m_Colors`) e aplica as modificações.

Verificação de Objetos Não Materiais

```
else
{
    Debug.LogWarning($"Selected object '{selectedObject.name}' is not a material.");
}
```

Se o objeto selecionado não for um material, registra um aviso.

Método de Remoção de Propriedades Inválidas

```
private static void RemoveInvalidProperties(SerializedProperty properties, HashSet<string> validProperties)
{
    for (int i = properties.arraySize - 1; i >= 0; i--)
    {
        var property = properties.GetArrayElementAtIndex(i);
        string propertyName = property.FindPropertyRelative("first").stringValue;

        if (!validProperties.Contains(propertyName))
        {
            properties.DeleteArrayElementAtIndex(i);
        }
    }
}
```

Este método remove propriedades inválidas:

- Itera pelas propriedades de trás para frente.

- Se a propriedade não estiver em ``validProperties``, ela é removida.

Resumo

Este script limpa propriedades inválidas de materiais selecionados no Unity Editor, garantindo que apenas propriedades válidas definidas pelo shader do material sejam mantidas. Isso ajuda a evitar dados redundantes ou inválidos nos materiais, potencialmente melhorando o desempenho e a organização dos projetos no Unity.

Finalidade do script

Este script simplifica a conversão de shaders de material para uso no Universal Render Pipeline (URP) do Unity. Ele foi projetado para shaders MikuMikuDance (MMD), permitindo uma transição mais suave das propriedades do shader e garantindo compatibilidade com o URP.

Visão geral de variáveis e métodos

1. `ShaderModel` Enum`

Define os tipos de shaders para os quais o script pode converter:

- `Default``: shader básico.
- `Tessellation``: shader com detalhes de superfície mais finos.
- `Empty``: shader mínimo sem efeitos adicionais.
- `FourLayers`` e `EightLayers``: shaders que suportam várias camadas de contorno.
- `NoShadow`` e `NoShadowAndTessellation``: desabilita sombras, com uma opção para tessellation.

2. Método `ConvertShader()`

- Este é o ponto de entrada principal, permitindo que um usuário selecione materiais no editor Unity para serem convertidos.
- Ele itera sobre cada material selecionado e, com base no nome do shader, seleciona a conversão apropriada chamando `ChangeShader()` com parâmetros específicos.

3. Método `ChangeShader()`

- Ajusta as propriedades do shader para o material selecionado, registrando-o para possíveis ações de desfazer.
- Define parâmetros básicos de renderização (`instancing``, configurações `GI``).
- Chama métodos auxiliares (`ApplyStandard``, `ApplyEmpty``, `ApplyMultiplePass``) com base no tipo de shader.
- Limpa propriedades incompatíveis usando `CleanMaterialProperties()`.

4. Método `ApplyStandard()`

- Configura a cor primária, a transparência e o tamanho da borda de um material.
- Ajusta as propriedades de sombra e contorno com base no `ShaderModel`` fornecido.
- Lida com as configurações de tesselação para shaders que precisam de detalhes de superfície mais finos.

5. Método `ApplyEmpty()`

- Aplica um shader mínimo, limpando propriedades extras e definindo transparência.
- Desativa a projeção de sombras e ajusta a fila de renderização para exibição transparente.

6. Método `ApplyMultiplePass()`

- Configura shaders que exigem renderização multipassagem (por exemplo, vários contornos).
- Mantém as configurações de cor do contorno e tamanho da borda para continuidade.

7. Método `RenderQueueToTransparent()`

- Ajusta a fila de renderização do material se a transparência estiver habilitada, colocando-a no intervalo correto para renderização transparente.

8. Método `CleanMaterialProperties()`

- Remove propriedades do material que não são necessárias para o novo shader.
- Usa `RemoveInvalidProperties`` e `CleanInvalidKeywords`` para limpeza precisa.

9. Método `RemoveInvalidProperties()`

- Remove propriedades inválidas do material com base nas suportadas pelo novo shader, o que garante transições limpas entre shaders.

10. Método CleanInvalidKeywords()

- Desativa quaisquer palavras-chave de shader não suportadas pelo novo shader, garantindo compatibilidade.

Uso

1. Selecionando materiais no Unity Editor

Selecione os materiais que deseja converter no editor Unity e execute `ConvertShader()` no menu em `Assets > MMD Collection > Convert Material Shader (MMD4Mecanim)`.

2. Conversão de shader

O script detectará o nome do shader de cada material, selecionará um shader compatível com URP apropriado e aplicará as configurações com base no tipo de shader.

3. Capacidade de desfazer

Como as alterações são registradas, os usuários podem desfazer a conversão do shader no editor.

4. Otimização de renderização

Por meio das funções auxiliares, o script garante que apenas as propriedades necessárias permaneçam no material, o que otimiza o desempenho da renderização.

Esta configuração automatiza a conversão de shader URP, garantindo que os shaders MMD permaneçam visualmente consistentes em projetos URP.

Manage Objects

Este script é um editor personalizado para Unity. O script tem como objetivo gerenciar a visibilidade de uma lista de GameObjects no editor do Unity. Vou dividir a explicação em duas partes: o script principal (**ManageObjects**) e o editor personalizado (**ManageObjectsEditor**).

Script Principal: **ManageObjects**

O script **ManageObjects** é um componente MonoBehaviour que pode ser adicionado a um GameObject na cena. Ele fornece funcionalidades para gerenciar a visibilidade de uma lista de GameObjects.

1. Definições e Variáveis

- **public ManageObjectsList[] manageObjects**: Um array de **ManageObjectsList**, que é uma classe serializável (definida ao final do script). Cada item no array contém uma referência a um GameObject e seu estado de visibilidade.
- **[HideInInspector] public bool state**: Variável que controla o estado global de visibilidade para todos os objetos na lista. Se **true**, todos os objetos são visíveis; se **false**, todos são invisíveis.
- **[HideInInspector] public bool hide**: Variável que controla se os objetos serão ocultados no inspetor.
- **[HideInInspector] public bool hideInspector**: Variável que controla se o inspetor padrão do componente será ocultado.

2. Métodos

- **public void Toggle(int i)**: Alterna a visibilidade do GameObject na posição **i** do array **manageObjects**. O método registra a ação para permitir o desfazer (undo) e altera o estado de ativação do GameObject.
- **public void ToggleAll()**: Alterna a visibilidade de todos os GameObjects na lista, de acordo com o estado global **state**. Também registra a ação para desfazer e aplica o estado a todos os GameObjects.
- **public void RemoveItem(int i)**: Remove o GameObject na posição **i** do array **manageObjects**, registrando a ação para desfazer.
- **Destruição Condicional**: Fora de builds de desenvolvimento, o componente **ManageObjects** é automaticamente destruído no método Start() para otimizar a performance em builds de produção.

Editor Personalizado: **ManageObjectsEditor**

O **ManageObjectsEditor** é uma classe que estende **Editor** e é responsável por criar uma interface personalizada no inspetor do Unity para o componente **ManageObjects**.

1. Método **OnInspectorGUI()**

Este método é chamado para desenhar a interface do inspetor personalizado:

- **Botão "Toggle All"**: Alterna a visibilidade de todos os objetos na lista e exibe o estado global.
- **DrawObjectBox(script)**: Desenha uma área onde você pode arrastar e soltar GameObjects para adicioná-los à lista. O método lida com eventos de arrastar e soltar e adiciona os objetos ao array se não estiverem já presentes.
- **EditorGUILayout.Toggle()**: Adiciona toggles para esconder objetos e o inspetor padrão.
- **DrawDisplayButtons(script)**: Desenha botões para alternar a visibilidade e remover cada

GameObject da lista, se os objetos não estiverem escondidos.

- **DrawDefaultInspector()**: Desenha o inspetor padrão do componente, se não estiver escondido.

2. Métodos Auxiliares

- **DrawObjectBox(ManageObjects script)**: Desenha a área de arrastar e soltar para adicionar novos GameObjects à lista. Verifica os eventos de arrastar e soltar e adiciona os GameObjects ao array **manageObjects** se não estiverem já presentes.

- **IsObjectInList(ManageObjects script, GameObject gameObject)**: Verifica se um GameObject já está na lista **manageObjects**.

- **DrawDisplayButtons(ManageObjects script)**: Desenha botões para alternar a visibilidade e remover objetos da lista, além de exibir mensagens de aviso se não houver objetos.

Classe **ManageObjectsList**

- **public GameObject gameObjects**: Referência ao GameObject que será gerenciado.

- **public bool objectState**: Estado de visibilidade do GameObject.

Resumo

O script fornece uma maneira prática de gerenciar a visibilidade de múltiplos GameObjects a partir de um editor personalizado no Unity. O componente **ManageObjects** gerencia a lista de GameObjects e seu estado de visibilidade, enquanto o **ManageObjectsEditor** oferece uma interface de usuário no inspetor para adicionar, remover e alternar a visibilidade dos objetos. Em builds de produção, o componente é removido automaticamente para evitar impacto na performance.

Descrição Geral

O objetivo deste script é encontrar todos os GameObjects na cena que possuem scripts faltando e selecioná-los no editor do Unity. Ele adiciona um item de menu para facilitar a execução dessa verificação.

Estrutura do Script

1. Declarações de Namespace e Biblioteca

```
using UnityEngine;
using UnityEditor;
using System.Collections.Generic;
using System.Linq;
```

- `UnityEngine`: Contém a API do Unity para manipulação de objetos de jogo.
- `UnityEditor`: Fornece classes e funções para criar ferramentas personalizadas no Editor do Unity.
- `System.Collections.Generic`: Inclui classes de coleções genéricas como `List`.
- `System.Linq`: Fornece funções de consulta para coleções, como `Any`.

2. Classe `FindMissingScripts`

```
public class FindMissingScripts : MonoBehaviour
{
```

- `public class FindMissingScripts : MonoBehaviour`: Define uma classe pública que herda de `MonoBehaviour`. Esta classe será anexada a um GameObject como um componente.

3. Método `FindAllMissingScripts`

```
[MenuItem("GameObject/MMD Collection/Find Missing Scripts")]
private static void FindAllMissingScripts()
```

- `[MenuItem("GameObject/MMD Collection/Find Missing Scripts")]`: Adiciona um item de menu no Unity Editor sob `GameObject -> MMD Collection -> Find Missing Scripts`. Este atributo faz com que o método `FindAllMissingScripts` seja executado quando o item de menu é selecionado.
- `private static void FindAllMissingScripts()`: Define um método estático privado que encontra todos os GameObjects com scripts faltando.

4. Lista de GameObjects com Scripts Faltando

```
List<GameObject> objectsWithMissingScripts = new();
```

- `List<GameObject> objectsWithMissingScripts = new();`: Cria uma lista para armazenar os GameObjects que possuem scripts faltando.

5. Encontrar Todos os GameObjects

```
GameObject[] allObjects = Resources.FindObjectsOfTypeAll<GameObject>();
```

- ``GameObject[] allObjects = Resources.FindObjectsOfTypeAll<GameObject>();``: Obtém todos os GameObjects no projeto, incluindo aqueles que não estão carregados na cena atual.

6. Iterar Sobre Todos os GameObjects

```
foreach (GameObject obj in allObjects)
{
    if (!obj.scene.IsValid() || !obj.scene.isLoaded || (obj.hideFlags & (HideFlags.NotEditable | HideFlags.HideAndDontSave)) != 0)
    {
        continue;
    }

    Component[] components = obj.GetComponents<Component>();
    if (components == null) continue;

    if (components.Any(component => component == null))
    {
        objectsWithMissingScripts.Add(obj);
    }
}
```

- ``foreach (GameObject obj in allObjects)``: Itera sobre todos os GameObjects encontrados.

- ``if (!obj.scene.IsValid() || !obj.scene.isLoaded || (obj.hideFlags & (HideFlags.NotEditable | HideFlags.HideAndDontSave)) != 0)``: Verifica se o objeto pertence a uma cena válida, se a cena está carregada e se o objeto não está marcado como não editável ou não salvável. Se alguma dessas condições for verdadeira, o objeto é ignorado (``continue``).

- ``Component[] components = obj.GetComponents<Component>();``: Obtém todos os componentes anexados ao GameObject atual.

- ``if (components == null) continue;``: Pula para o próximo objeto se não houver componentes (o que não deveria ocorrer, mas é uma verificação de segurança).

- ``if (components.Any(component => component == null))``: Verifica se algum dos componentes é nulo (indicando um script faltando).

- ``objectsWithMissingScripts.Add(obj);``: Adiciona o GameObject à lista se possuir algum script faltando.

7. Log e Seleção dos Objetos

```
Debug.Log($"Found {objectsWithMissingScripts.Count} objects with missing scripts.");
Selection.objects = objectsWithMissingScripts.ToArray();
```

- ``Debug.Log($"Found {objectsWithMissingScripts.Count} objects with missing scripts.");``: Exibe uma mensagem no console do Unity com o número de objetos encontrados com scripts faltando.

- ``Selection.objects = objectsWithMissingScripts.ToArray();``: Seleciona os objetos com scripts faltando no Editor do Unity, facilitando a localização e correção.

Conclusão

Este script é útil para desenvolvedores que precisam garantir que não há scripts faltando nos GameObjects da cena, o que pode causar erros ou comportamentos inesperados. Ao adicionar um item de menu no editor, ele torna o processo de verificação rápido e acessível.

Shader Keyword Checker

Este script foi criado para verificar palavras-chave de shader global em materiais selecionados no Unity Editor. Ele adiciona uma nova opção no menu de contexto Asset do Unity Editor, permitindo que os usuários acionem a verificação em materiais selecionados.

Variáveis e métodos

1. Atributo MenuItem

Este atributo é usado para criar um novo item no menu Unity Editor. O caminho do menu é `"Assets/MMD Collection/Check Shader Keywords"`, e ele vincula ao método `CheckShaderKeywords`. Quando selecionado no menu, o método é executado.

2. Método CheckShaderKeywords

Este é o método principal acionado quando o usuário seleciona o item de menu. Seu objetivo principal é iterar por todos os objetos selecionados no Unity Editor e verificar se cada um é um material.

- **Selection.objects**: Esta é uma propriedade Unity que contém a lista de objetos selecionados no editor. O método a usa para acessar esses objetos.

- **Material material**: O script verifica se cada objeto selecionado é um `Material`. Se for, o `Shader` associado ao material é recuperado.

- **shader**: Esta variável representa o shader associado ao material. Se nenhum shader for atribuído ao material, uma mensagem de erro será registrada.

- Se o material tiver um shader, o script chamará o método `CheckGlobalShaderKeywords` para inspecionar as palavras-chave do shader global.

3. Método CheckGlobalShaderKeywords

Este método recebe um objeto `Shader` e verifica suas palavras-chave do shader global.

- **ShaderUtil**: O Unity tem uma classe interna chamada `ShaderUtil` que contém utilitários para operações relacionadas ao shader. No entanto, esta classe não é acessível publicamente.

- **Reflection (GetMethod)**: Como `ShaderUtil` é interno, o script usa reflection para acessar seu método oculto `GetShaderGlobalKeywords`. O Reflection permite acessar métodos e propriedades privadas ou internas em C#.

- **getKeywordsMethod**: Esta variável armazena o resultado da chamada da função `GetMethod`, que tenta encontrar o método `GetShaderGlobalKeywords` usando seu nome e sinalizadores de vinculação especiais (`Static` e `NonPublic`).

- Se o método for encontrado, ele será invocado para recuperar as palavras-chave do shader global, que são registradas no console do Unity. Se o método não for encontrado, um erro será registrado, indicando a falha.

Uso

1. Selecione materiais no Unity Editor.

2. Clique com o botão direito do mouse nos ativos selecionados e navegue até a opção `"MMD Collection"` no menu `"Assets"`.

3. Escolha `"Check Shader Keywords"`.

4. O script verificará se os ativos selecionados são materiais e recuperará as palavras-chave globais do shader. Essas palavras-chave serão registradas no console do Unity.

MMD Bone Renderer

Visão geral (Objetivo)

Este script é um componente utilitário criado para ajudar desenvolvedores a visualizar os ossos de um personagem ou rig no Editor Unity. Ele funciona enquanto você edita a cena (não apenas no modo de jogo) e permite que você veja as linhas entre os ossos, as pontas das cadeias de ossos e defina opções de exibição específicas para cada osso.

É especialmente útil ao lidar com personagens 3D complexos, como aqueles usados em modelos MikuMikuDance (MMD), onde entender a estrutura óssea visualmente pode ser de grande ajuda.

Variáveis Principais

- **drawBones** : Uma opção que controla se os ossos devem ser desenhados na cena ou não.
- **jointSize** : Um número que controla o tamanho dos pontos de articulação quando visualizados na cena.
- **boneRendererList** : Uma lista de grupos, onde cada grupo tem:
 - uma **cor específica** para os ossos.
 - uma lista de **ossos individuais** e suas configurações (como se eles são rotativos, móveis ou visíveis).

Classes de dados

- **Dados do BoneRenderer** :
 - Isso define um grupo de ossos com uma cor compartilhada.
 - Cada grupo contém vários ossos e suas configurações individuais.
- **BonesTransform** :
 - Isto representa um único osso e seu comportamento visual:
 - Deve girar?
 - Deveria se mover?
 - Deveria ser visível na cena?
- **Par de transformação** :
 - Apenas um par de ossos: um é o pai e o outro é o filho.
 - Usado para desenhar linhas entre ossos conectados.

Principais Características e Métodos

Ativar / Desativar

- Quando este componente é ativado ou desativado, ele irá:
 - Reconstrua a lista de ossos.
 - Notifique outras partes do editor que este renderizador foi adicionado ou removido (por meio de eventos estáticos).

AltoGetBones

- Esta é uma opção de menu de contexto que você pode acionar manualmente.
- Ele verifica **todos os filhos do objeto atual** e os adiciona a um novo grupo de ossos.
- Esta é uma maneira rápida de preencher automaticamente a lista de ossos, embora um aviso deixe claro que esta pode não ser sempre a melhor maneira de usar a ferramenta.

Transformações (Propriedade)

- Isso permite que você **obtenha** uma lista de todas as transformações ósseas atualmente armazenadas no renderizador.
- Você também pode **definir** uma nova lista de transformações ósseas, que substituirão os grupos existentes e reconstruirão os dados internos.

Redefinir / ClearBones

- Esses métodos removem todas as referências ósseas existentes e limpam os dados visuais.
- É como clicar em “reiniciar” na configuração do seu osso visual.

Invalidar / Extrair Ossos

- Esses métodos atualizam ou reconstróem a estrutura visual com base nos ossos armazenados atualmente.
- O script verifica:

- Quais ossos estão conectados.
- Quais ossos não têm filhos e são considerados dicas.
- Ele armazena essas informações em matrizes para serem usadas no desenho.
- Ele também ignora ossos que estão ocultos no editor ou não visíveis na visualização da camada atual.

Eventos

- O script suporta eventos que são acionados quando um renderizador de osso é adicionado ou removido.
- Outros sistemas (como editores ou gerenciadores personalizados) podem escutar esses eventos e atualizar adequadamente.

Execução somente para editores

Tudo está encapsulado para ser executado **somente no Editor Unity** :

- A visualização, seleção de ossos e extração não afetam a jogabilidade.
- Elas servem apenas para ajudar na hora de projetar ou editar a cena.

Como usar

1. **Anexá-lo a um GameObject** : você pode fazer isso através do menu "Adicionar Componente" do editor Unity, em "Coleção MMD/Renderizador Bone".
2. **Adicionar ossos manualmente** : por meio do Inspetor, atribua ossos à lista, agrupados por cor.
3. **Use AltoGetBones** : Opcionalmente, clique com o botão direito do mouse no componente e escolha "Alto Get Bones" para preencher automaticamente a lista com todos os ossos filhos.
4. **Personalize cada osso** : alterne movimento, rotação e visibilidade por osso.
5. **Visualizar na Visualização de Cena** : se ativado, os ossos aparecerão como linhas com articulações e segmentos coloridos, com base nas suas configurações.

MMD Bone Renderer Inspector

Visão geral (Objetivo)

Este script é um **editor personalizado** para o componente MMDBoneRenderer. Ele substitui a interface de usuário padrão do inspetor do Unity por uma personalizada que:

- Permite **atualizações ao vivo** quando os valores mudam.
- Lida corretamente com operações **desfazer** e **refazer** .
- Garante que os dados internos do osso permaneçam atualizados sem que o desenvolvedor precise pressionar um botão manualmente.

Isso é parte do que faz com que o sistema visual pareça responsivo e fácil de usar no Editor Unity.

Principais componentes e o que eles fazem

- **Editor personalizado para MMDBoneRenderer** :
 - Isso significa que sempre que você selecionar um GameObject que tenha o componente MMDBoneRenderer, o Unity usará essa interface personalizada em vez da interface interna.
- **Pode editar vários objetos** :
 - Isso permite que você selecione vários GameObjects com componentes MMDBoneRenderer e edite-os ao mesmo tempo.

Método principal: OnInspectorGUI

Este método controla o que acontece quando o Unity desenha o inspetor na janela do Editor. Aqui está um detalhamento do processo:

1. **Atualizar dados serializados** :
 - Ele obtém as informações mais recentes dos componentes MMDBoneRenderer selecionados e as sincroniza com o inspetor.
2. **Detectar alterações** :
 - Ele começa a rastrear se o usuário altera alguma coisa na janela Inspetor — como alternar a visibilidade, ajustar o tamanho da articulação ou modificar listas de ossos.
3. **Desenhe a interface do usuário do Inspetor** :
 - Ele usa o método integrado do Unity para desenhar todos os campos normalmente, para

que o usuário ainda veja e edite os controles esperados.

4. **Verifique se algo mudou :**

- Após o desenho, ele verifica se alguma propriedade realmente mudou.

5. **Lidar com eventos desfazer/refazer :**

- Ele também verifica se o usuário pressionou **Desfazer** ou **Refazer** , porque essas ações podem alterar propriedades sem nenhuma entrada direta do usuário no inspetor.

6. **Aplicar as alterações :**

- Se alguma alteração for feita, ela será confirmada no componente.

7. **Atualização do Trigger a Bone :**

- Se alguma coisa mudar (por edição ou desfazer), ele percorre cada alvo selecionado, encontra o `MMDBoneRenderer` associado e diz para **recalcular os ossos** usando seu método interno.
- Isso garante que os visuais e os dados internos sempre reflitam o estado mais recente no inspetor.



Uso

Este script é executado **automaticamente** . Você não precisa chamá-lo ou anexá-lo manualmente. Veja o que acontece quando ele está no seu projeto:

1. **Selecione um objeto com `MMDBoneRenderer`** : Você verá os campos usuais do inspetor (como `drawBones`, `jointSize` e a lista de ossos).
2. **Fazer alterações** : ajustar o tamanho das articulações, alterar a visibilidade, adicionar ossos, etc.
3. **Atualização ao vivo** : assim que você altera algo, o renderizador de ossos é atualizado internamente, para que suas alterações apareçam na visualização da cena sem demora.
4. **Desfazer ou Refazer** : Pressione `Ctrl+Z` ou `Ctrl+Y` (ou use o menu) para desfazer ou refazer uma alteração. O sistema atualiza os ossos automaticamente sem comprometer o estado visual.



Resumo

Este script não adiciona novos controles chamativos — ele melhora a usabilidade e a estabilidade do componente `MMDBoneRenderer` ao:

- Garantir que as edições do inspetor sejam rastreadas corretamente.
- Reagindo a Desfazer/Refazer como um profissional.
- Mantendo os dados visuais do osso sempre atualizados.

É um reforço de qualidade de vida nos bastidores que torna a experiência do editor mais suave e previsível para desenvolvedores e artistas.

MMD Bone Renderer Utils

Este script é um utilitário do Editor Unity para renderizar estruturas ósseas, especificamente para modelos MikuMikuDance (MMD), usando instanciação de GPU para melhorar o desempenho. Ele foi projetado para visualizar hierarquias ósseas na visualização da cena e oferecer suporte a interações como seleção e transformação. Veja como funciona, detalhado por seção:



Classe principal: `MMDBoneRendererUtils`

Marcado com `[InitializeOnLoad]`, o que significa que ele executa seu construtor estático quando o Unity carrega o editor.



Classe interna: `BatchRenderer`

Lida com renderização em lote de muitas malhas ósseas usando instanciação de GPU.

- **Constantes:**
 - `kMaxDrawMeshInstanceCount`: Número máximo de instâncias de malha por chamada de desenho.
 - `SubMeshType`: enumeração que distingue faces sólidas e wireframe para ossos.
- **Campos:**
 - `malha`: A malha a ser desenhada (por exemplo, uma pirâmide para um osso).
 - `material`: O material compartilhado para desenho.
 - `m_Matrices`: Transforma matrizes para cada instância.
 - `m_Colors`: Cor base por instância.
 - `m_Highlights`: Cor de destaque por instância (passar o mouse/seleção).

- **Métodos principais:**

- `AddInstance(...)`: Enfileira uma malha para renderização com transformação e cores.
- `Clear()`: Redefine as filas de instâncias.
- `Render()`: Desenha todas as instâncias enfileiradas usando instanciação de GPU e buffers de comando.

Campos Globais

- `s_BoneRendererComponents`: instâncias ativas de `MMDBoneRenderer` na cena.
- `s_PyramidMeshRenderer`: Uma instância do `BatchRenderer` criada preguiçosamente usando uma malha piramidal personalizada.
- `s_Material`: Um material compartilhado carregado via `EditorGUIUtility.LoadRequired(...)`.
- `s_ButtonHash`: ID exclusivo para controles de GUI por osso.
- `s_VisibleLayersCache`: Armazena camadas visíveis para detectar alterações.

Construtor estático

Configura ganchos de eventos:

- Adiciona/remove renderizadores.
- Reage a mudanças de visibilidade e modificações de hierarquia.
- Conecta-se ao `SceneView.duringSceneGui` para desenhar ossos.

Configuração de malha e material

- **Propriedade do material (`Material`)** : Carrega e armazena em cache um shader usado para renderização de ossos com instanciação de GPU.
- **Malha de pirâmide (`PyramidMeshRenderer`)** : cria uma malha personalizada em forma de pirâmide para representar um osso.
 - **Faces (`SubMesh 0`)** : Triângulos formando a pirâmide.
 - **Fio (`SubMesh 1`)** : Linhas que contornam a malha para wireframe.

`ComputeBoneMatrix(...)`

Calcula a matriz de transformação para desenhar um osso entre dois pontos 3D, alinhando uma pirâmide no espaço para representá-la visualmente.

Desenhar Esqueletos(`SceneView`)

O coração do script. Executa cada quadro na visualização da cena.

- Ignora objetos invisíveis ou irrelevantes (como aqueles em um estágio pré-fabricado diferente).
- Itera sobre todos os ossos visíveis:
 - Desenha linhas ósseas entre pai/filho.
 - Desenha "dicas" para ossos sem filhos.
 - Renderiza alças (cubos/esferas) para interação (selecionar/mover/girar).
- Chama `PyramidMeshRenderer.Render()` para desenhar todas as instâncias em lote.

`SãoOssosNoMesmoGrupo(...)`

Verifica se dois ossos pertencem ao mesmo grupo `BoneRendererData` — usado para decidir se um osso deve ser renderizado.

`DoBoneRender(...)`

Manipula todas as interações para um único osso:

- Seleção com o mouse.
- Define a ferramenta Unity correta (mover/girar/transformar).
- Renderiza osso usando `PyramidMeshRenderer.AddInstance(...)`.

Adicionar/Remover Renderizador de Osso

- `OnAddBoneRenderer(...)` e `OnRemoveBoneRenderer(...)`: mantêm a lista de renderizadores ativos.

Mudanças de visibilidade e hierarquia

- `OnVisibilityChanged()` e `OnHierarchyChanged()`: marcam renderizadores como invalidados e atualizam a visualização da cena.

Bones Transform Drawer

Este script define uma **gaveta de propriedades personalizada** no Editor Unity para uma classe chamada **BonesTransform**. Seu objetivo é melhorar a forma como o Unity exibe objetos **BonesTransform** no **Inspetor**. Em vez de exibir uma lista simples com subcampos, este script os apresenta de forma limpa e compacta para melhor usabilidade ao trabalhar com muitos ossos em modelos MMD.

Vamos analisar isso passo a passo:

Declaração de classe: **BonesTransformDrawer**

Esta classe é marcada com `[CustomPropertyDrawer(typeof(BonesTransform))]`, o que significa:

- O Unity usará automaticamente esta gaveta sempre que encontrar um campo **BonesTransform** no **Inspetor**.
- O objetivo é personalizar visualmente como os dados no **BonesTransform** são exibidos.

Ele herda de **PropertyDrawer**, que dá acesso a dois métodos principais:

- `OnGUI(...)`: Renderiza a GUI personalizada.
- `GetPropertyHeight(...)`: Calcula quanto espaço vertical a propriedade ocupa.

Método: **OnGUI(posição, propriedade, rótulo)**

Este método é chamado toda vez que o Unity precisa desenhar um campo **BonesTransform** no **Inspetor**.

EditorGUI.BeginProperty(...) e **EndProperty(...)**

Essas funções encapsulam o processo de desenho, permitindo substituições de predefinições e desfazendo o rastreamento dos campos.

lineHeight e espaçamento

Esses valores de loja para layout da interface do usuário:

- `lineHeight`: Altura padrão da linha para uma linha de propriedade.
- `espaçamento`: Espaço vertical entre linhas.

Linha de rótulos

```
'Rect labelRect = novo(posição.x, posição.y, posição.largura, alturaDaLinha);'  
'EditorGUI.LabelField(labelRect, rótulo);'
```

- Desenha um rótulo para toda a propriedade (opcional, mas melhora a clareza).

Campo de Ossos

```
'Rect ossoRect = novo(...);'  
'EditorGUI.PropertyField(boneRect, property.FindPropertyRelative("bone"), new  
GUILayout("Bone"));'
```

- Mostra a referência a uma transformação que representa o osso.
- É um campo de referência, permitindo que os usuários arrastem/soltem uma transformação.

Alternar linha

Esta seção manipula três sinalizadores booleanos:

- `girar`: se este osso deve girar.
- `mover`: Se este osso deve se mover.
- `visível`: Se este osso deve ser mostrado.

```
'Rect toggleRow = novo(...);'
```

- Calcula uma linha horizontal para os botões de alternância.
- Divide a linha em 3 botões de alternância de tamanho igual usando `toggleWidth`.

Antes de desenhá-los:

- Ele reduz `EditorGUIUtility.labelWidth` para 50, para que os nomes dos rótulos não ocupem muito espaço.

Cada alternância é desenhada usando `EditorGUI.PropertyField`, e os rótulos são definidos de forma personalizada como "Girar", "Mover", "Visível".

Após o desenho, ele restaura a largura do rótulo ao seu valor anterior.

Método: GetPropertyHeight(propriedade, rótulo)

Este método informa ao Unity qual deve ser a altura de toda a gaveta.

- Ele usa:
 - 3 linhas de altura: uma para o rótulo, uma para o campo do osso, uma para os botões de alternância.
 - 2 espaços de espaçamento vertical entre as 3 linhas.

Então o valor de retorno é:

'lineHeight * 3 + espaçamento * 2'

Isso garante que tudo se encaixe perfeitamente.

Resumo de Uso

Quando um desenvolvedor adiciona um campo BonesTransform a um MonoBehaviour ou ScriptableObject, esta gaveta:

- Exibe o campo como uma pequena caixa personalizada no Inspetor.
- Mostra a referência do osso e as três bandeiras (girar, mover, visível) em um layout organizado e intuitivo.
- Torna a edição de dados relacionados aos ossos muito mais fácil durante o desenvolvimento e a configuração da animação.