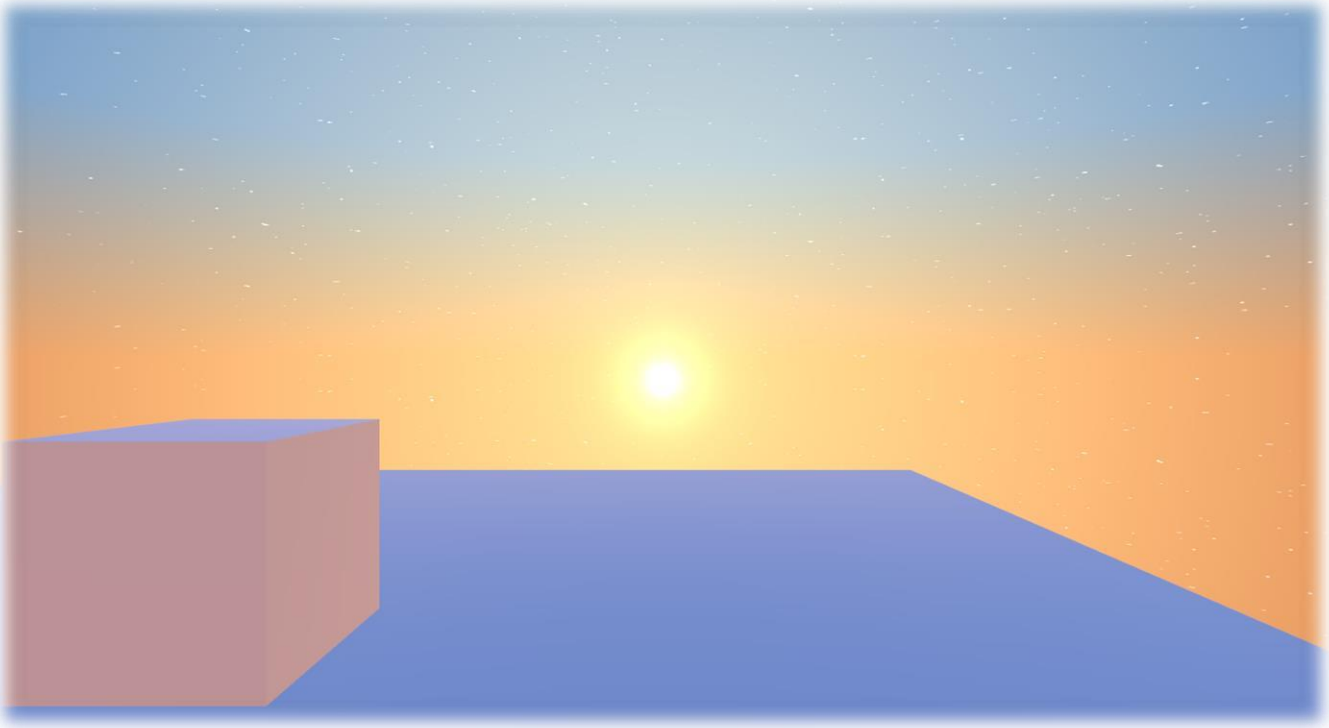


Skybox Universal RP

Documentation

_____

Creator: FlowingCrescent

Modified:

- Gabriel Henrique Pereira

- Lucas Gomes Cecchini **Pseudonym:** AGAMENOM

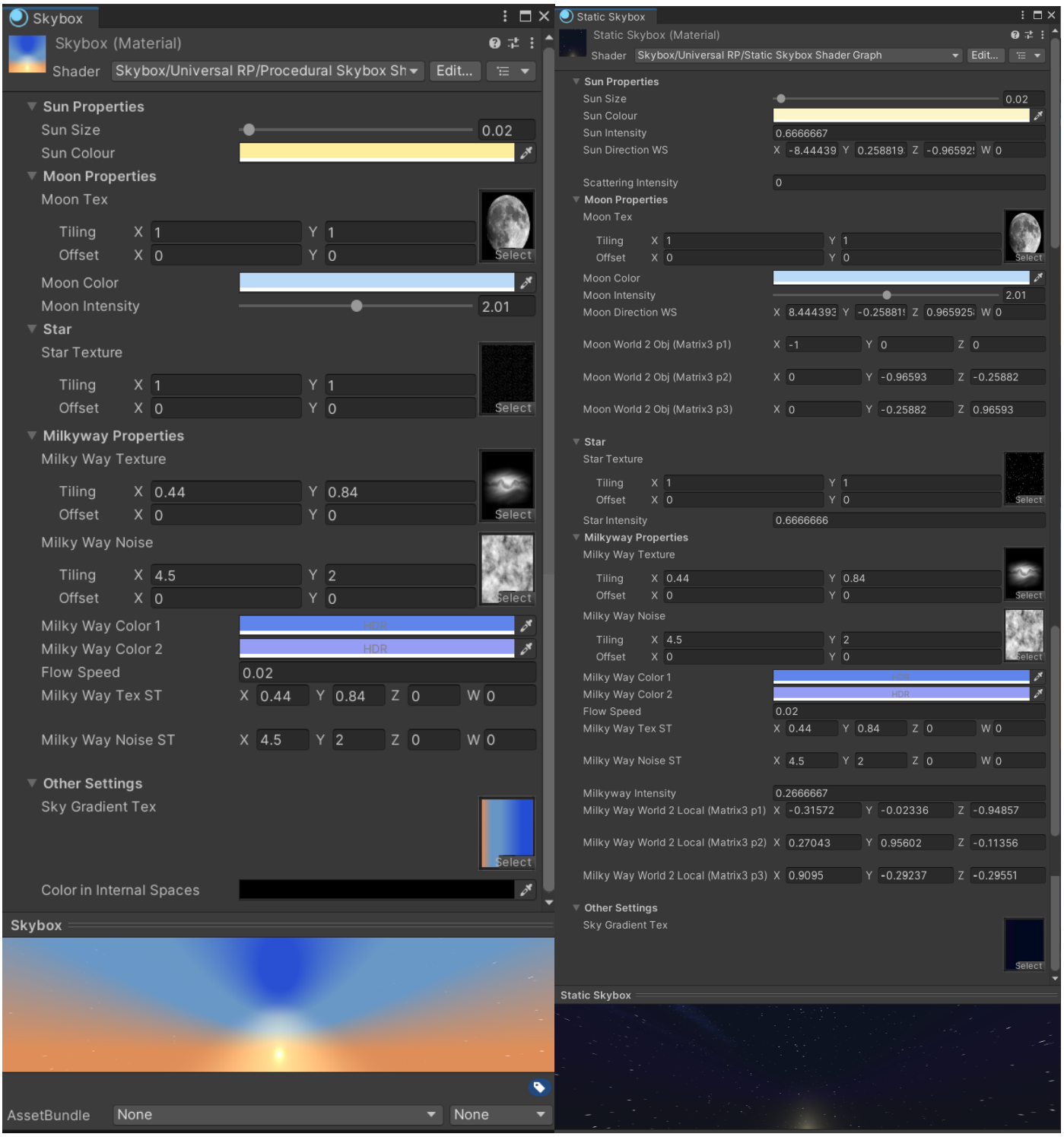
Simple Procedural Skybox

A simple procedural skybox which has day and night in unity universal render pipeline Unity version：6000.0.32f1, URP 17.0.3.

Please make sure your version of unity is similar for avoiding some errors due to version change.

Original File: https://github.com/FlowingCrescent/SimpleProceduralSkybox_URP

Shader Explanations



Skybox Shader

This shader is a procedural skybox for the Universal Render Pipeline (URP) in Unity. It creates a dynamic sky with features such as the sun, moon, stars, and Milky Way. Below is a breakdown of the shader's components:

Properties

Sun Properties:
- _**SunSize:** Controls the size of the sun.
- _**SunIntensity:** Determines how bright the sun appears.

- **_SunCol:** Sets the color of the sun.
- **_SunDirectionWS:** Defines the direction of the sun in world space.
- **_ScatteringIntensity:** Controls the intensity of light scattering around the sun.

Sky Textures:
- **_StarTex:** Texture for the stars in the sky.
- **_MilkyWayTex:** Texture for the Milky Way.
- **_MilkyWayNoise:** Noise texture for the Milky Way, adding variation.
- **_MilkyWayCol1:** The first color for the Milky Way.
- **_MilkyWayCol2:** The second color for the Milky Way.
- **_MilkywayIntensity:** Controls the brightness of the Milky Way.
- **_FlowSpeed:** Speed at which the Milky Way noise moves.

Moon Properties:
- **_MoonCol:** Sets the color of the moon.
- **_MoonIntensity:** Determines the brightness of the moon.
- **_MoonDirectionWS:** Defines the direction of the moon in world space.

Star Intensity:
- **_StarIntensity:** Controls the brightness of the stars.

Shader Structure

1. CBUFFER Blocks
These define the properties that are sent to the GPU. The shader uses various properties for the sun, moon, and Milky Way.

2. Vertex Shader (`vert` function)
This function transforms vertices from object space to screen space and computes additional data like the position of the moon and Milky Way.

3. Mie Phase Function (`getMiePhase`)
This function simulates the Mie scattering, a type of scattering of light, which is important for rendering realistic skies.

4. Sun Attenuation Calculation (`calcSunAttenuation`)
This function calculates the sun's attenuation based on the light's position and the direction of rays.

5. Voronoi Noise (`VoronoiNoise`)
This generates a procedural noise pattern that can be used to create a more organic look in the sky, such as star distribution.

6. Soft Light Blending (`softLight`)
A blending mode that adjusts the colors in a non-linear way, making highlights softer and shadows less harsh.

7. Fragment Shader (`frag` function)
This is where the final color of each pixel is determined. The function blends the sky color, sun, moon, stars, and Milky Way textures to produce the final output.

Final Output (`finCol`)
- **skyColor:** Base color from the sky gradient.
- **sun:** Sun color and scattering.
- **star:** Star texture.
- **moon:** Moon texture and scattering.
- **milkyCol:** Milky Way color, including noise and star effects.

Summary

This shader dynamically generates a skybox with the sun, moon, stars, and the Milky Way. The use of procedural textures, Voronoi noise, and phase functions helps to create a realistic sky with natural lighting effects. The shader is highly customizable, allowing for the adjustment of various parameters such as color, intensity, and texture movement.

Script Explanations

Sky Time Data Copier

This script, `**SkyTimeDataCopier**`, is a Unity editor utility designed to copy a folder named **"SkyTimeData"** from its original location to a folder selected by the user within the Unity project. It utilizes the Unity Editor API to interact with the project's assets and file system.

Explanation:

1. **Menu Integration:**
   - The script integrates with the Unity Editor's menu, specifically under `**Assets > Create > Skybox URP > SkyTimeData Library**`. Selecting this menu item will trigger the folder copying process.

2. **Main Method (CopySkyTimeData):**
   - **Finding the "SkyTimeData" Folder:** The script searches the entire Unity project for folders named **"SkyTimeData"** using Unity's asset database. It retrieves an array of unique identifiers (GUIDs) for each matching folder.
   - **Checking for Folder Existence:** The script checks if any **"SkyTimeData"** folders were found. If no such folder exists, an error message is logged, and the operation stops.
   - **Converting GUID to Folder Path:** The script converts the first found folder's GUID to a path string, representing the folder's location within the project.
   - **Determining the Destination Folder:** The script then determines the destination folder by calling a helper method that retrieves the path of the currently selected folder in the Unity project window.
   - **Validating Paths:** Before proceeding, the script ensures that both the source and destination folders exist. If either is missing, it logs an error and exits.
   - **Copying the Folder:** The script constructs the full destination path by combining the selected folder path with the name of the source folder. It then copies the entire contents of the source folder to this destination.
   - **Refreshing the Asset Database:** After copying, the script refreshes the asset database to ensure that Unity recognizes the newly copied assets.
   - **Logging the Operation:** Finally, a success message is logged to inform the user that the folder has been copied and where it was placed.

3. **Helper Method (GetSelectedFolderPath):**
   - **Default Path:** If no folder is selected in the Unity project, the script defaults to the **"Assets"** folder.
   - **Iterating through Selected Objects:** The script checks the currently selected objects in the Unity project window. If a folder is selected, it uses that folder's path. If a file is selected, it uses the path of the directory containing the file.
   - **Returning the Path:** The method returns the path of the selected folder or defaults to **"Assets"** if no valid folder was selected.

Summary:
The `**SkyTimeDataCopier**` script simplifies the process of duplicating a specific folder structure within a Unity project. By integrating with the Unity Editor, it allows developers to quickly and easily copy a **"SkyTimeData"** folder to a chosen destination, without the need to manually navigate through the project's file system. This can be particularly useful for managing skybox data or other assets that need to be duplicated across different parts of a project.

Sky Controller Create

This script, **`SkyControllerCreate`**, is a Unity Editor tool that allows users to create and manage specific game objects related to skyboxes directly from the Unity Editor's GameObject menu. The script adds three custom menu items: **"Sky Controller," "SkyController Trigger,"** and **"Screenshot Skybox,"** each corresponding to a different functionality. Here's a detailed explanation:

Overview

- **Sky Controller:** Instantiates a prefab named **"Sky Controller"** in the scene, optionally as a child of the currently selected game object.
- **SkyController Trigger:** Creates a new game object named **"SkyController Trigger"** with a **`BoxCollider`** set as a trigger and a **`SkyControllerTrigger`** component.
- **Screenshot Skybox:** Creates a new game object named **"Screenshot Skybox"** and attaches multiple cameras to it, oriented in different directions for taking screenshots of a skybox.

1. Sky Controller

Functionality:
- **Searches for the "Sky Controller" Prefab:** The script looks through the project for a prefab named **"Sky Controller"** using the Unity Editor's asset database.
- **Validates the Prefab:** If no such prefab is found, it logs an error and exits. If found, it loads the prefab from the specified path.
- **Instantiates the Prefab:** The prefab is instantiated in the scene. If a game object is selected, the new object is created as a child of the selected object; otherwise, it is created at the root level of the hierarchy.
- **Renames and Unpacks:** The new game object is renamed to **"Sky Controller,"** and the prefab is unpacked, meaning it is converted from an instance of a prefab into a regular game object.
- **Undo Registration:** The action is registered with Unity's undo system, allowing the user to undo the creation of the object if needed.

2. SkyController Trigger

Functionality:
- **Creates a New GameObject:** The script checks if a game object named **"SkyController Trigger"** already exists in the scene. If it does, it appends a number to the name to ensure uniqueness.
- **Parenting:** If a game object is currently selected in the hierarchy, the new **"SkyController Trigger"** object is made a child of that object.
- **Adds Components:** A **`BoxCollider`** is added and set as a trigger, and a **`SkyControllerTrigger`** component is also added to the object.
- **Undo Registration:** The creation of the object and any added components are registered with the undo system, allowing the user to undo these actions.

3. Screenshot Skybox

Functionality:
- **Creates a New GameObject:** Similar to the previous method, this creates a new game object named **"Screenshot Skybox,"** ensuring the name is unique.
- **Parenting:** The new object is optionally parented to the currently selected game object in the hierarchy.
- **Adds a `ScreenshotSkybox` Component:** A custom component, **`ScreenshotSkybox`**, is added to handle skybox screenshot functionalities.
- **Adds Cameras:** Six cameras are created and added as child objects to the **"Screenshot Skybox"** object, each oriented to capture a different face of the skybox (front, left, back, right, up, down).
- **Configures Cameras:** Each camera is configured with a 90-degree field of view, a near clip plane of 0.01, and a far clip plane of 1.0. The cameras are rotated to face different directions to cover all sides of a cube, ideal for capturing a full skybox.
- **Assigns Cameras to `ScreenshotSkybox`:** The array of cameras is assigned to the **`cameras`** property of the **`ScreenshotSkybox`** component.
- **Selects the Object in the Editor:** The newly created **"Screenshot Skybox"** object is selected in the Unity Editor.
- **Undo Registration:** The creation of the object and the addition of the cameras are registered with the undo system.

Summary

The **`SkyControllerCreate`** script provides three useful tools for managing skyboxes and related objects in Unity:

- **Sky Controller:** Facilitates the instantiation of a predefined skybox controller prefab, making it easier to manage skyboxes in the scene.
- **SkyController Trigger:** Allows users to create triggers associated with skybox controllers, useful for dynamic skybox changes in-game.
- **Screenshot Skybox:** Automates the setup of a game object with multiple cameras for capturing skybox screenshots, streamlining the process of generating skybox textures.

Each tool is designed to integrate seamlessly into the Unity Editor, enhancing productivity and providing a more efficient workflow for managing skybox-related assets.

Skybox Materials Creator

This script defines a utility for creating skybox materials within Unity's editor, focusing on convenience and workflow efficiency for users developing scenes with Unity's Universal Render Pipeline (URP). Here's a breakdown of its functionality:

Overview

The script facilitates the quick creation of skybox materials by adding options directly into the Unity Editor's **"Assets"** menu. By selecting **"Assets/Create/Skybox URP/Material/Skybox"** or **"Skybox Static,"** users can instantly generate new skybox materials based on predefined templates. This approach ensures each material is correctly named and placed in the right folder, streamlining material management.

Key Components and Flow

1. Menu Items for Skybox Creation

Two options are added to Unity's **"Assets/Create"** menu:
- **Skybox**: Creates a skybox material based on a **"Skybox Base"** template.
- **Skybox Static**: Uses a **"Static Skybox Base"** template.

These options are designed to speed up the workflow by eliminating the need for manual duplication of base materials.

2. Core Function for Material Duplication

The method **`CreateSkyboxMaterial()`** manages the core process:
- **Base Material Search**: Using **`AssetDatabase.FindAssets`**, the script looks for a material in the project that matches the specified base name (e.g., **"Skybox Base"** or **"Static Skybox Base"**).
- **Error Handling**: If no matching material is found, it logs an error, preventing the process from continuing with an invalid reference.
- **Material Duplication**: Once the base material is located, the script creates a duplicate.
- **Unique Naming and Storage**: The duplicate is given a unique name to prevent overwriting and saved in the current project folder.

3. Automatic Selection and Renaming

After the new material is created:
- The material is **automatically selected in the Editor** and the user is prompted to rename it, making the process of managing and organizing materials simpler.
- The script uses Unity's **`delayCall`** to allow the Editor time to apply changes before prompting the rename.
- An F2 key event is simulated, which is the standard shortcut for renaming in Unity's Project view. This minimizes interruptions by letting the user rename the new material immediately after creation.

4. Folder Path Handling

The **`GetCurrentFolderPath`** method determines the active folder in the Unity project, ensuring that the new material is saved in the expected location:

- If the user has a specific folder selected in the Project view, the script will use that location.
- If a file is selected instead of a folder, it defaults to the file's directory.
- If no specific selection is made, the default location is the `Assets` root folder.

Summary of Workflow

1. User selects either **"Skybox"** or **"Skybox Static"** from the **"Assets/Create/Skybox URP/Material"** submenu.
2. The script searches for the corresponding base material and, if found, duplicates it.
3. The new material is saved with a unique name in the current folder.
4. The material is automatically selected, and the user is prompted to rename it.
5. Unity displays a confirmation in the console upon successful creation.

This script simplifies the creation of skybox materials, centralizes material organization, and reduces manual handling. The automated naming and selection steps make it highly efficient for large projects with multiple skybox materials.

Sky Time Data

The **`SkyTimeData`** script is a **`ScriptableObject`** in Unity that holds various properties related to the visual appearance and environmental settings of a skybox, particularly in a Universal Render Pipeline (URP) context. This script is designed to store data for customizing the appearance of the sky at different times of the day or night, including factors such as sky color, sun intensity, scattering, and celestial objects like stars and the Milky Way.

Here's a detailed explanation of the script:

Overview

- **Purpose:** The **`SkyTimeData`** class is a data container that stores sky-related settings that can be used to dynamically alter the appearance of the sky in a game or application. By using a **`ScriptableObject`**, this data can be easily created, stored, and reused across different scenes or projects.

- **Usage:** This **`ScriptableObject`** can be created through the Unity Editor and used by other scripts or systems that handle sky rendering or day-night cycles.

1. ScriptableObject Definition

- **`ScriptableObject`:** The class inherits from **`ScriptableObject`**, making it a type of object that can be saved as an asset in Unity. Unlike regular **`MonoBehaviour`** scripts, **`ScriptableObject`** instances are not attached to GameObjects; they exist as independent data containers.

- **`[CreateAssetMenu]`:** This attribute adds a menu option in the Unity Editor to create instances of this **`ScriptableObject`**.
  - **`menuName = "Skybox URP/SkyTimeData"`**: Specifies the path in the Unity Editor where the asset creation option will appear (e.g., **"Create -> Skybox URP -> SkyTimeData"**).
  - This makes it easy for developers to create new **`SkyTimeData`** assets directly from the Unity Editor.

2. Public Fields

The class contains several public fields, each representing different properties of the sky environment. Since these fields are public, they can be edited directly in the Unity Editor when the **`SkyTimeData`** asset is selected.

- **`[Header("SkyTimeData Settings")]`:** This attribute adds a header above the following fields in the Unity Inspector. It helps to organize and label the settings when editing the asset.

- **`public Gradient skyColorGradient;`**
  - **Description:** This field holds a **`Gradient`**, which is a color gradient used to define how the sky's color changes over time (e.g., from day to night).
  - **Use Case:** Useful for creating smooth transitions in sky color based on time or other factors.

- **`public float sunIntensity;`**
  - **Description:** A `float` value that determines the intensity of the sun's light in the skybox.
  - **Use Case:** Can be adjusted to simulate different times of the day (e.g., bright midday sun or dim evening light).

- **`public float scatteringIntensity;`**
  - **Description:** A **`float`** value that controls the intensity of atmospheric scattering, which affects how light diffuses through the atmosphere.
  - **Use Case:** Scattering can create effects like the blueness of the sky or the reddening of the sun during sunset.

- **`public float starIntensity;`**
  - **Description:** A **`float`** value that specifies the brightness of stars in the sky.
  - **Use Case:** Can be used to adjust the visibility of stars, particularly during nighttime.

- **`public float milkywayIntensity;`**
  - **Description:** A **`float`** value that controls the brightness of the Milky Way galaxy in the sky.
  - **Use Case:** Enhances the visibility of the Milky Way during clear nights.

3. Optional Texture Field

- **`[Space(20)]`:** Adds 20 pixels of vertical spacing in the Unity Inspector, which helps separate different sections of settings for better readability.

- **`public Texture2D skyColorGradientTex;`**
  - **Description:** A `Texture2D` that can store a texture representation of the sky color gradient.
  - **Use Case:** This texture can be used in shaders or other systems that need a precomputed gradient texture, allowing for more complex or detailed control over the sky's appearance.

Summary

The **`SkyTimeData`** class is a highly customizable **`ScriptableObject`** that stores data related to the appearance of the sky in a Unity project. It includes settings for sky color gradients, sun intensity, atmospheric scattering, and celestial objects like stars and the Milky Way. By storing this data in a **`ScriptableObject`**, Unity developers can easily create, modify, and reuse sky settings across different scenes or projects, allowing for a dynamic and visually appealing skybox experience in their games or applications.

Sky Time Data Controller

The **`SkyTimeDataController`** script is designed to interpolate between different instances of **`SkyTimeData`** based on the time of day in a Unity project. This allows for smooth transitions in the sky's appearance, such as color changes, lighting adjustments, and the visibility of celestial objects, providing a dynamic day-night cycle or other time-based effects.

Here's a detailed explanation of the script:

1. Class Overview

- **`SkyTimeDataController`:** This class manages the interpolation between different **`SkyTimeData`** instances based on the time of day. It is intended to be attached to a GameObject in a Unity scene and provides real-time updates to the sky and environment lighting as time progresses.

- **Attributes:**
  - **`[ExecuteInEditMode]`:** This attribute allows the script to execute not only during play mode but also in the Unity Editor, making it possible to preview the effects without entering play mode.
  - **`[AddComponentMenu("Skybox URP/Sky Time Data Controller")]`:** Adds this script to the Unity Editor's component menu under **"Skybox URP,"** making it easier to find and add to GameObjects.

2. Public Fields

- **`[Header("Required Items")]`:** This attribute creates a labeled section in the Unity Inspector to organize the required fields.

- **`public SkyTimeDataCollection skyTimeDataCollection = new();`:**
  - **Description:** This field stores a collection of **`SkyTimeData`** instances, each representing the sky settings for a specific time of day (e.g., morning, noon, evening).
  - **Use Case:** This collection is used to determine the start and end points for interpolation based on the current time.

- **`[HideInInspector] public bool updateEnvironmentLighting;`:**
  - **Description:** This boolean flag indicates whether the environment lighting should be updated. It is hidden in the Inspector to prevent direct modification by users.

- **`[HideInInspector] public Color defaultColorEnvironmentLighting;`:**
  - **Description:** This color is used as the default environment lighting color when `updateEnvironmentLighting` is false. It is hidden in the Inspector.

- **`[HideInInspector] public bool inInternalSpace;`:**
  - **Description:** This flag indicates whether the player is in an internal space (e.g., indoors) where different lighting settings might be applied. It is hidden in the Inspector.

- **`[HideInInspector] public Color colorEnvironmentLighting;`:**
  - **Description:** This color is used for environment lighting when the player is in an internal space. It is hidden in the Inspector.

3. Private Fields

- **`private SkyTimeData newData;`:**
  - **Description:** A temporary **`SkyTimeData`** instance used to store the interpolated sky settings. This is where the blended values from the start and end **`SkyTimeData`** instances are stored.

4. Lifecycle Methods

- **`private void OnEnable()`:**
  - **Purpose:** This method is called when the script is enabled (e.g., when the GameObject is activated or the scene is loaded).
  - **Functionality:** It creates a new instance of **`SkyTimeData`** to be used for storing the interpolated values.

5. Interpolation Method

- **`public SkyTimeData GetSkyTimeData(float time)`:**
  - **Purpose:** This method returns an interpolated **`SkyTimeData`** instance based on the provided time of day.
  - **Process:**
    1. **Determine Start and End Data:**
       - The method selects the appropriate **`SkyTimeData`** instances from **`skyTimeDataCollection`** based on the given **`time`** (which is expected to be between 0 and 24, representing hours in a day).
       - The time is divided into 3-hour segments, with each segment corresponding to a specific start and end **`SkyTimeData`**.

    2. **Calculate Lerp Value:**
       - A linear interpolation value (**`lerpValue`**) is calculated based on how far the current time is within the 3-hour segment.

    3. **Generate Sky Gradient Texture:**
       - The method generates a new sky gradient texture by interpolating between the gradients of the start and end **`SkyTimeData`** instances using **`GenerateSkyGradientColorTex`**.

4. **Update Environment Lighting:**
   - The environment lighting is updated based on the interpolated **`SkyTimeData`** and certain conditions like whether the lighting update is enabled or if the player is in internal space.

5. **Interpolate Other Properties:**
   - The method interpolates other sky-related properties such as **`starIntensity`**, **`milkywayIntensity`**, **`sunIntensity`**, and **`scatteringIntensity`** between the start and end **`SkyTimeData`** instances.

6. **Return Interpolated Data:**
   - Finally, the method returns the **`newData`** instance containing the interpolated values.

6. Environment Lighting Update

- **`private void UpdateEnvironmentLighting(SkyTimeData start, SkyTimeData end, float lerpValue)`:**
  - **Purpose:** This method updates the ambient lighting settings in the scene based on the interpolated **`SkyTimeData`**.
  - **Process:**
    1. **Calculate Ambient Colors:**
       - The method interpolates the colors for the sky, equator, and ground ambient lighting using the gradients from the start and end **`SkyTimeData`** instances.

    2. **Check Flags:**
       - If **`updateEnvironmentLighting`** is false, the ambient colors are set to the **`defaultColorEnvironmentLighting`**.
       - If **`inInternalSpace`** is true, the ambient colors are set to **`colorEnvironmentLighting`**.

    3. **Apply Ambient Colors:**
       - The method updates **`RenderSettings`** in Unity with the calculated ambient colors.

7. Gradient Texture Generation

- **`public Texture2D GenerateSkyGradientColorTex(Gradient startGradient, Gradient endGradient, int resolution, float lerpValue)`:**
  - **Purpose:** This method generates a new **`Texture2D`** representing a gradient that is a blend of two input gradients.
  - **Process:**
    1. **Create Texture:**
       - A new texture with the specified resolution is created, and its filter mode and wrap mode are set for smooth rendering.

    2. **Calculate Pixel Colors:**
       - The method loops through each pixel in the texture and calculates its color by linearly interpolating between the corresponding colors in the start and end gradients.

    3. **Apply Texture:**
       - The calculated colors are applied to the texture, which is then returned for use in sky rendering.

8. SkyTimeDataCollection Class

- **`[System.Serializable] public class SkyTimeDataCollection`:**
  - **Purpose:** This nested class is a container for multiple **`SkyTimeData`** instances, each representing a specific time of day.
  - **Fields:**
    - **`public SkyTimeData time0;` through `public SkyTimeData time21;`** each represent **`SkyTimeData`** instances for times like midnight, 3 AM, 6 AM, etc.
  - **Use Case:** This collection is used by the **`SkyTimeDataController`** to interpolate between different sky settings throughout the day.

Summary

The **`SkyTimeDataController`** script dynamically adjusts the sky and environment lighting in a Unity scene based on the time of day. It interpolates between predefined **`SkyTimeData`** instances to create smooth transitions in sky color, sun intensity, star visibility, and other atmospheric effects. The script also handles environment lighting updates and can generate a texture representing the sky gradient for use in rendering. This approach allows for a rich and dynamic visual experience, particularly useful in games or simulations with day-night cycles or other time-based environmental changes.

Sky Controller

This script is a Unity component that manages the skybox, time progression, and environmental lighting in a scene. It's designed for both runtime and editor mode, allowing dynamic control over day-night cycles and environmental settings.

Script Breakdown:

1. Class Declaration and Attributes

- **`[ExecuteInEditMode]`: Enables the script to execute its `Update()`** method even when the game isn't running.
- **`[RequireComponent(typeof(SkyTimeDataController))]`**: Ensures that any GameObject with this script also has a **`SkyTimeDataController`** component.
- **`[AddComponentMenu("Skybox URP/Sky Controller")]`**: Adds the script to Unity's component menu under **"Skybox URP."**

2. Public Variables

- **Time Settings:**
  - **`executeInEditMode`**: Boolean to determine if **`Update()`** should run in edit mode.
  - **`time`**: A float representing the current time of day (0 to 24).
  - **`timeGoes`**: Enum controlling how time progresses (e.g., disabled, automatic, etc.).
  - **`reverseTime`**: Boolean to reverse the direction of time progression.
  - **`TimeSpeed`**: Speed of time progression.
  - **`TimeOfDay`**: The length of a full day cycle in seconds.
- **Necessary Components:**
  - **`SkyboxMaterial`**: Material for the skybox.
  - **`LightSun`** & **`LightMoon`**: Light sources representing the sun and moon.
  - **`sunTransform`**, **`moonTransform`**, **`milkyWayTransform`**: Transforms for the sun, moon, and milky way.
  - **`milkyWayRotation`**: Vector3 for setting the milky way's rotation.
- **Environment Lighting Settings:**
  - **`inInternalSpace`**: Boolean indicating if the scene is internal (affecting lighting).
  - **`update`**: Controls if the environment lighting should be updated.
  - **`defaultColor`**: Default color for environment lighting.
  - **`maxIntensity`**: Maximum intensity for sun and moon.
- **Automatic Day and Night Information:**
  - **`updateGITime`**: Time counter for updating Global Illumination.
  - **`dayOrNightChanging`**: Boolean indicating if a day-night transition is happening.
  - **`totalTimeByTime`**: Tracks cumulative time progression.

3. Private Variables

- **`skyTimeDataController`**: Reference to **`SkyTimeDataController`**, which manages time data.
- **`currentSkyTimeData`**: Stores the current interpolated **`SkyTimeData`**.

4. Enums

- **`TimeGoes` Enum**: Defines the possible modes for time progression.
  - **`Disabled`**: Time doesn't progress.
  - **`OnlyOnStart`**: Time progression occurs only at the start of the scene.
  - **`Enable`**: Time progresses automatically.
  - **`TimeBySpeed`**: Time progresses at a set speed.
  - **`TimeByTime`**: Time progresses based on total time of day.

5. OnEnable() Method
- Initializes the **`skyTimeDataController`** by getting the **`SkyTimeDataController`** component from the GameObject.

6. Update() Method
- **Edit Mode Check:** The script only runs in edit mode if **`executeInEditMode`** is true.
- **Time Progression:** Adjusts the **`time`** variable based on the selected **`timeGoes`** mode.
- **Day-Night Transition:**
  - Checks if it's time to start the day or night and triggers the respective coroutine (**`ChangeToDay`** or **`ChangeToNight`**).
- **Global Illumination Update:** Calls **`DynamicGI.UpdateEnvironment()`** if **`updateGITime`** exceeds 0.5 seconds.
- **Sky Time Data:** Fetches the interpolated **`SkyTimeData`** based on the current time.
- **Lighting and Material Updates:**
  - Updates environmental lighting and toggles keywords for internal space.
  - Updates the positions of the sun, moon, and milky way and sets relevant properties in the skybox material.

7. ControllerSunAndMoonTransform() Method
- Calculates and sets the positions and rotations of the sun and moon based on the current time.
- Updates the directional vectors for the sun and moon in the skybox material.

8. CalculateIntensity() Method
- Calculates the intensity of the sun or moon using a cosine function based on the time of day.

9. SetProperties() Method
- Sets various properties of the skybox material, including textures and lighting intensities, based on the current **`SkyTimeData`**.

10. Coroutines for Day-Night Transitions
- **`ChangeToNight()`** & **`ChangeToDay()`**: These methods gradually fade the sun and moon lights in and out, simulating the transition between day and night.

11. Editor Script (SkyControllerEditor)
- **OnInspectorGUI()**: Custom inspector interface for the script, including a button to save the gradient texture.
- **SaveGradientTexture()**: Method to save the sky gradient texture as a PNG file. Uses a temporary render texture to capture and encode the gradient, saving it to a user-selected folder.

Summary
The **`SkyController`** script is a comprehensive tool for managing the day-night cycle and skybox appearance in a Unity scene. It provides real-time control over lighting, time progression, and skybox settings, making it ideal for dynamic environments. The script also includes editor integration, allowing developers to adjust settings and save textures directly from the Unity editor.

Sky Controller Menu

This script is a Unity C# component named **`SkyControllerMenu`**, which provides a user interface (UI) for controlling various settings of a **`SkyController`** component in a Unity project. The script manages UI elements like sliders, dropdowns, toggles, and input fields, allowing users to interactively modify the sky's appearance and behavior.

Components and Variables

- **SkyController (`skyController`)**: A reference to the **`SkyController`** component that this script interacts with. This component likely controls the sky's appearance, such as time progression, lighting, and possibly weather effects.

- **UI Elements:**

- **Slider (`time`)**: A slider to control the time of day within the **`SkyController`**.
- **Text (`hour`)**: A UI text element that displays the current time value.
- **Dropdown (`timeGoes`)**: A dropdown menu for selecting how time progresses (e.g., by speed or by a set time).
- **Toggle (`reverseTime`)**: A toggle switch to reverse the flow of time.
- **InputField (`timeSpeed`)**: An input field to set the speed at which time progresses.
- **InputField (`timeOfDay`)**: An input field to specify the length of a full day cycle.

Unity Methods

- **OnEnable()**: This method is called when the script is enabled. It initializes the UI elements based on the current settings of the **`SkyController`**, sets up listeners to handle UI interactions, and configures input validation for numeric input fields.

- **OnDisable()**: This method is called when the script is disabled. It removes the listeners added in **`OnEnable()`** and stops validating input for the numeric fields, ensuring no further interactions occur while the script is inactive.

Update Methods

- **UpdateTime(float value)**: Updates the **`SkyController`**'s time based on the slider value. It also updates the displayed time percentage.

- **UpdatePercentage(float value)**: Converts the time value to an integer and updates the **`hour`** text to reflect this value in the UI.

- **UpdateTimeGoes(int value)**: Converts the dropdown value to a corresponding **`SkyController.TimeGoes`** enumeration value and updates the **`SkyController`** accordingly. It also updates the interactivity of the time-related UI elements based on the selected time progression mode.

- **UpdateReverseTime(bool value)**: Updates the **`SkyController`**'s reverse time setting based on the toggle's state.

- **UpdateTimeSpeed(string value)**: Tries to parse the string value from the **`timeSpeed`** input field as a floating-point number. If successful, it updates the **`SkyController`**'s time speed.

- **UpdateTimeOfDay(string value)**: Similar to **`UpdateTimeSpeed`**, but this method updates the length of a full day in the **`SkyController`**.

- **UpdateTimeGoesInteractable(SkyController.TimeGoes timeGoesValue)**: This method enables or disables the interactivity of the **`timeSpeed`** and **`timeOfDay`** input fields based on the selected time progression mode. It allows only the relevant field to be interactable at any given time.

Input Validation

- **ValidateNumericInput(string text, int charIndex, char addedChar)**: This method ensures that only numeric input (digits or a single decimal point) is allowed in the input fields for **`timeSpeed`** and **`timeOfDay`**. If the input is invalid (e.g., a second decimal point or a non-numeric character), the method returns a null character (`\0`) to prevent it from being added to the input field.

Usage

The **`SkyControllerMenu`** script provides an intuitive interface for adjusting the sky settings in a Unity project. It allows users to interact with the time of day, time progression speed, and other related settings through a combination of sliders, dropdowns, and text input fields. This script is typically used in projects where dynamic environmental changes are essential, such as in open-world games or simulations that simulate day/night cycles and other time-based environmental effects.

Sky Controller Trigger

This script, **`SkyControllerTrigger`**, is designed to interact with a **`SkyController`** component in a Unity project. It triggers a change in the environment's **"internal space"** setting when a specified object (typically the player) enters a designated trigger zone. This can be useful for creating dynamic environmental changes based on player location, such as transitioning between different skyboxes or lighting setups.

Components and Variables

- **`inInternalSpace`**: A boolean that determines whether the **`SkyController`** should be in **"internal space."** This could control various environmental settings, such as lighting, skybox appearance, or weather effects, that change when transitioning from one area to another (e.g., indoors vs. outdoors).

- **`playerTag`**: A string that specifies the tag of the GameObject representing the player. The script uses this tag to identify the object that should trigger the change in the **`SkyController`**. By default, this is set to **"Player,"** but it can be customized to match the tag used in your project.

- **`skyController`**: A reference to the **`SkyController`** component in the scene. This script assumes that there is only one **`SkyController`** in the scene, and it finds this component when the script is enabled.

Unity Methods

- **`OnEnable()`**: This method is called when the script is enabled. It finds the **`SkyController`** component in the scene using **`FindObjectOfType<SkyController>()`**. This ensures that the script can interact with the **`SkyController`** as soon as it becomes active.

- **`OnTriggerEnter(Collider other)`**: This method is triggered when another collider enters the trigger zone attached to the GameObject that this script is attached to. The method performs the following actions:
  - **Tag Check**: It first checks if the collider that entered the trigger has the same tag as the **`playerTag`** variable.
  - **SkyController Check**: If the tag matches, it checks whether the **`skyController`** component was successfully found in the **`OnEnable`** method.
  - **Internal Space Update**: If both checks pass, the script updates the **`inInternalSpace`** property of the **`SkyController`** to the value specified in the **`inInternalSpace`** variable. This likely triggers a change in the environment, such as switching to an indoor skybox when the player enters a building.
  - **Debug Logging**: The script logs a message to the console to confirm the change. If no **`SkyController`** is found, it logs a warning instead.

Additional Features

- **`[RequireComponent(typeof(Collider))]`**: This attribute ensures that a Collider component is attached to the same GameObject as this script. Unity automatically adds a Collider if one is not already present. This is crucial because the script relies on trigger events, which require a Collider component to function.

- **`[AddComponentMenu("Skybox URP/Sky Controller Trigger")]`**: This attribute adds the script to the Unity editor's **"Add Component"** menu under the specified path. It organizes the script under **"Skybox URP"** to make it easier to find when adding it to a GameObject.

Usage

This script is typically used in scenarios where the environment needs to change dynamically based on the player's location. For example, it could be used to switch between different lighting or skybox settings when the player enters or exits a building, moves between different areas of a map, or crosses specific environmental boundaries. The **`inInternalSpace`** variable allows you to control whether the **`SkyController`** should simulate an indoor or outdoor environment, and the **`playerTag`** ensures that only the designated player object triggers this change.

Screenshot Skybox

This script, **`ScreenshotSkybox`**, is designed to capture a 360-degree screenshot of the skybox in Unity using six cameras. These cameras are positioned and rotated to capture each face of a cube, which can be later stitched together to form a complete skybox image. The script also includes a custom editor to facilitate taking screenshots and opening the folder where screenshots are saved directly from the Unity Inspector.

Key Components and Variables

- **`pixelSize`**: This integer determines the resolution of each side of the skybox screenshot. A value of **`2048`** means each face of the skybox will be 2048x2048 pixels.

- **`cameras`**: This array holds six **`Camera`** objects, each responsible for capturing one face of the skybox. The script ensures that all six cameras are present, as each one corresponds to a specific direction (front, back, left, right, top, and bottom).

Methods

1. `TakeScreenshot()`

This is the main method used to capture the skybox screenshot.

- **Camera Validation**: It checks if exactly six cameras are assigned to the **`cameras`** array. If not, an error message is logged, and the screenshot process is aborted.

- **Apply Camera Positions and Rotations**: The **`ApplyCameraPositionsAndRotations()`** method is called to ensure that each camera is positioned at the origin and rotated to face one of the six directions needed for a 360-degree capture.

- **Screenshot Folder**: The script checks if a folder named **"Screenshots"** exists within the project's Assets directory. If it doesn't, the folder is created.

- **Rendering and Saving**:
  - For each camera, a **`RenderTexture`** is created, and the camera renders its view into this texture.
  - The texture is then converted into a **`Texture2D`** object, and the pixel data is read into this object.
  - The **`Texture2D`** is then encoded into a PNG file.
  - The screenshot is saved in the **"Screenshots"** folder with a filename that includes the camera name and the current date and time.

- **Cleanup**: After the screenshot is taken, the **`RenderTexture`** is destroyed to free up memory.

2. `ApplyCameraPositionsAndRotations()`

This method sets each camera's position and rotation to capture the skybox correctly:

- Each camera is positioned at the origin (**`Vector3.zero`**) and rotated to face one of the six directions needed for a full 360-degree capture:
  - Camera 1: Front (default)
  - Camera 2: Right (rotated 90° around the Y-axis)
  - Camera 3: Back (rotated 180° around the Y-axis)
  - Camera 4: Left (rotated -90° around the Y-axis)
  - Camera 5: Top (rotated 90° around the X-axis)
  - Camera 6: Bottom (rotated -90° around the X-axis)

3. `OpenScreenshotFolder()`

This method opens the "Screenshots" folder in the file explorer so the user can easily access the captured screenshots.

Custom Editor: `ScreenshotSkyboxEditor`

This custom editor script enhances the Unity Inspector for the **`ScreenshotSkybox`** component, adding buttons to take screenshots and open the screenshot folder directly from the Inspector.

- **Buttons**: The custom editor adds two buttons:
  - **Take Screenshot**: Calls the **`TakeScreenshot()`** method when clicked.
  - **Open Folder**: Calls the **`OpenScreenshotFolder()`** method when clicked.

- **Inspector Layout**: After these buttons, the default Inspector for the component is drawn, showing the **`pixelSize`** and **`cameras`** array.

Usage

This script is useful for capturing high-quality 360-degree images of a skybox, which can be used for creating panoramic views or custom skyboxes in Unity. The **`ScreenshotSkybox`** component should be attached to a GameObject in the scene, and six cameras must be assigned to the **`cameras`** array. By using the custom editor, you can easily capture screenshots without writing additional code or navigating through complex menus.

Considerations

- **Performance**: Capturing high-resolution screenshots can be resource-intensive, so it's best to use this tool when the game is not running or during a specific setup phase.
- **Camera Configuration**: Ensure that all six cameras are properly configured with the correct field of view and are free from any visual obstructions that might appear in the screenshots.

This script provides a powerful tool for creating custom skyboxes or panoramic images in Unity, making it easier for developers and artists to capture the exact visual environment they need for their projects.