# Blender Automation Script Documentation

## Overview

This script is designed to automate the process of rendering 3D models in Blender. It processes multiple 3D mesh files (.obj, .stl, .glb) located in subfolders, applies lighting and camera settings, and generates images and stereoscopic turntable animations from various angles. The rendered outputs are saved in a specified directory, making this script ideal for batch processing and visualisation tasks.

### Key Features:

- **Batch Processing:** The script processes multiple 3D models stored in subfolders.
- **HDRI Lighting Setup:** Utilises an HDRI file to create realistic environment lighting.
- **Mesh Manipulation:** Includes functions to fit the mesh within a bounding box, correct orientation, and apply smooth shading.
- **Camera and Lighting:** Sets up camera positions, adjusts distance dynamically, and applies various types of lighting (area, sun, and point lights).
- **Rendering:** Produces single-frame renders from multiple angles and creates stereoscopic turntable animations for 3D models.
- **Output Management:** Automatically organises rendered images and videos into corresponding output directories.

## Workflow

1. **Directory Setup:**

   The script begins by defining the main folder path containing the 3D models and the output path where the rendered images and videos will be saved.

   It ensures the output directory exists and creates it if necessary.

2. **HDRI Lighting Setup:**

   An HDRI file is loaded and applied to the Blender scene to simulate realistic environmental lighting.

3. **Colour Management:**

   The script configures the colour management settings to control exposure and gamma, ensuring the rendered images maintain a consistent look.

**4. Mesh Import and Processing:**

The script iterates through each subfolder, identifying and importing 3D mesh files (.obj, .stl, .glb).

Once imported, the meshes are combined into a single object, centered, and scaled to fit within a predefined bounding box.

The orientation of the meshes is corrected to ensure they are upright.

**5. Camera and Lighting Setup:**

A camera is added to the scene, and its position is adjusted dynamically based on the size of the mesh.

Various light sources, including area, sun, and point lights, are added to the scene to ensure the mesh is well-lit from all angles.

**6. Rendering:**

The script generates still images of the mesh from multiple predefined camera positions.

It also creates stereoscopic turntable animations (360-degree views) with different interocular distances (IODs) for a 3D effect.

**7. Cleanup:**

After processing each mesh, the script ensures that the scene is cleared of objects to avoid interference with subsequent meshes.

The processed meshes are deleted from the Blender scene to maintain a clean working environment.

## Detailed Function Descriptions

### 1. setup_hdri_lighting(hdri_path)

· **Purpose:** Sets up HDRI (High Dynamic Range Image) environment lighting for the scene.

· **Functionality:** This function clears any existing world nodes in the scene and creates a new environment texture node using the provided HDRI file. It also sets up mapping and background nodes to ensure proper lighting.

### 2. setup_color_management()

· **Purpose:** Configures the colour management settings for the Blender scene.

- **Functionality:** Adjusts the view transform, exposure, and gamma settings to maintain a neutral colour profile, preventing any unwanted colour corrections during rendering.

### 3. combine_objects()

- **Purpose:** Combines all selected mesh objects into a single object.
- **Functionality:** Selects all mesh objects in the scene and joins them into one, which simplifies further processing and rendering.

### 4. make_mesh_unique(mesh_object)

- **Purpose:** Ensures the mesh object has unique data.
- **Functionality:** Copies the mesh data so that the object is independent and does not share data with other objects, allowing for safe transformations.

### 5. render_frame(mesh_name, position_name, position, output_path)

- **Purpose:** Renders a single frame from a specified camera position.
- **Functionality:** Moves the camera to the specified position, sets the frame, and renders an image, saving it with a name that includes the mesh and position identifiers.

### 6. fit_mesh_to_bounding_box(mesh_object, target_size)

- **Purpose:** Scales the mesh to fit within a defined bounding box.
- **Functionality:** Adjusts the mesh's origin and scale to ensure it fits neatly within the target size, adding a small padding for consistency.

### 7. correct_mesh_orientation(mesh_object)

- **Purpose:** Corrects the orientation of the mesh to ensure it is upright.
- **Functionality:** Checks if the mesh is upside down and rotates it 180 degrees around the X-axis if necessary.

### 8. adjust_camera_distance(mesh_object, base_distance=10, padding_factor=1.5)

- **Purpose:** Adjusts the camera's distance based on the mesh's bounding box size.
- **Functionality:** Calculates the optimal distance for the camera to ensure the entire mesh is visible with appropriate padding.

### 9. center_mesh_in_camera_view(camera, mesh_object)

- **Purpose:** Centres the mesh within the camera's view.

- **Functionality:** Aligns the camera to track the mesh and centres it in the view, ensuring the mesh is correctly positioned for rendering.

### 10. setup_camera_for_rendering(camera, mesh_object)

- **Purpose:** Sets up the camera's position and distance for rendering the mesh.
- **Functionality:** Adjusts the camera's location based on the mesh's size and centres the mesh within the camera's view.

### 11. rotate_camera_around_mesh(camera, mesh_object, frame_count, radius, eye_offset=0)

- **Purpose:** Animates the camera rotation around the mesh for a turntable effect.
- **Functionality:** Rotates the camera in a circular path around the mesh, keyframing its position for each frame.

### 12. render_stereoscopic_turntable(subfolder_name, mesh_name, output_path, frame_count, radius, eye_distance)

- **Purpose:** Renders stereoscopic turntable animations from two slightly different perspectives (left and right eye).
- **Functionality:** Produces two videos, one for each eye, by rotating the camera around the mesh with an offset based on the specified interocular distance (IOD).

### 13. generate_camera_positions(n, distance)

- **Purpose:** Generates a set of camera positions around the mesh for rendering.
- **Functionality:** Distributes n camera positions evenly around a circle of specified radius and returns them as vectors.

### 14. render_flexible_frames(subfolder_name, mesh_name, output_path, num_positions, distance)

- **Purpose:** Renders multiple frames from various camera positions around the mesh.
- **Functionality:** Uses the generated camera positions to render images from different angles and save them with descriptive filenames.

## Configuration and Customization

- **HDRI Path:** Customise the hdri_path variable to use a different HDRI file for environment lighting.
- **Output Path:** Change the output_path variable to save the renders in a different directory.

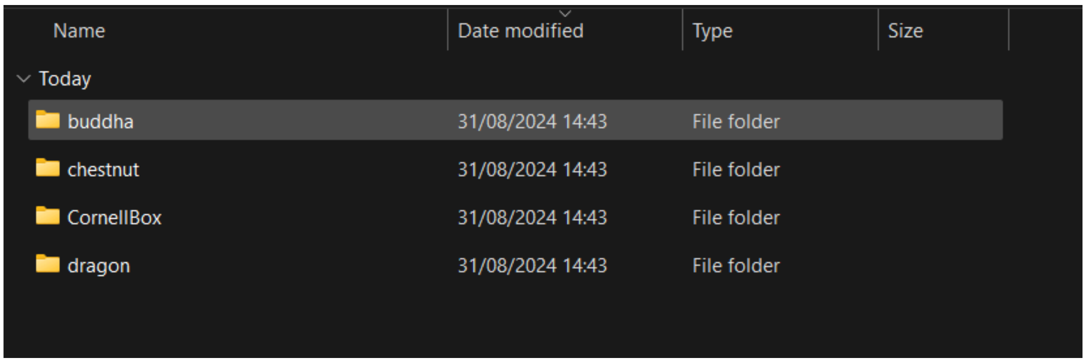
- **Frame Rate and Resolution:** Adjust frame_rate and resolution settings (resolution_x and resolution_y) based on the desired output quality and performance.
- **Render Engine:** The script defaults to Blender's Cycles engine, but you can switch to Eevee by uncommenting the appropriate lines.

## Instructions for Running the Blender Automation Script

To effectively run the Blender automation script and process your 3D models, follow these steps:

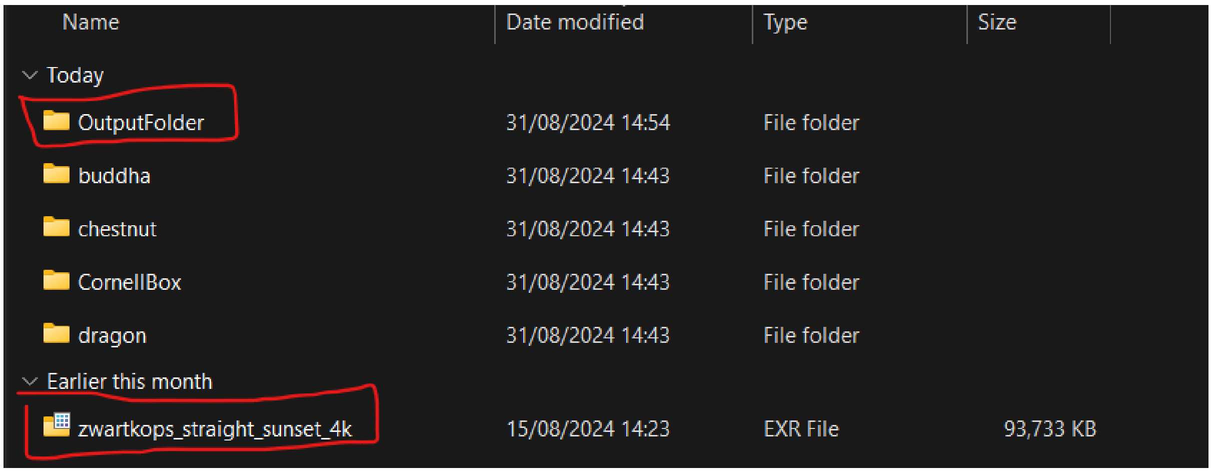
### Step 1: Prepare Your Folders

1. **Input Folder Structure**:
   - Create a main folder called input in your desired location.
   - Inside the input folder, create subfolders for each set of 3D models you wish to process. For example, you might have subfolders like buddha, dragon, chestnut, etc.
   - Place the corresponding 3D mesh files (.obj, .stl, .glb) inside these subfolders.



| Name | Date modified | Type | Size |
|---|---|---|---|
| Today | | | |
| buddha | 31/08/2024 14:43 | File folder | |
| chestnut | 31/08/2024 14:43 | File folder | |
| CornellBox | 31/08/2024 14:43 | File folder | |
| dragon | 31/08/2024 14:43 | File folder | |

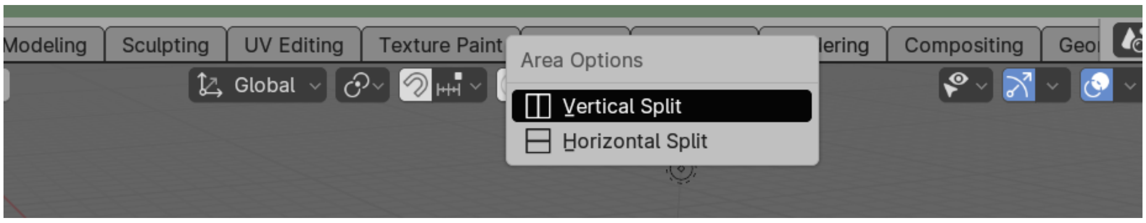
2. **HDRI File**:
   - Place your HDRI map file inside the input folder. This file will be used to create realistic environmental lighting in Blender.
3. **Output Folder**:
   - Create an output folder in your desired location. The script will automatically organise the rendered images and videos into subfolders within this directory, matching the structure of your input folder.



| Name | Date modified | Type | Size |
|---|---|---|---|
| Today | | | |
| OutputFolder | 31/08/2024 14:54 | File folder | |
| buddha | 31/08/2024 14:43 | File folder | |
| chestnut | 31/08/2024 14:43 | File folder | |
| CornellBox | 31/08/2024 14:43 | File folder | |
| dragon | 31/08/2024 14:43 | File folder | |
| Earlier this month | | | |
| zwartkops_straight_sunset_4k | 15/08/2024 14:23 | EXR File | 93,733 KB |

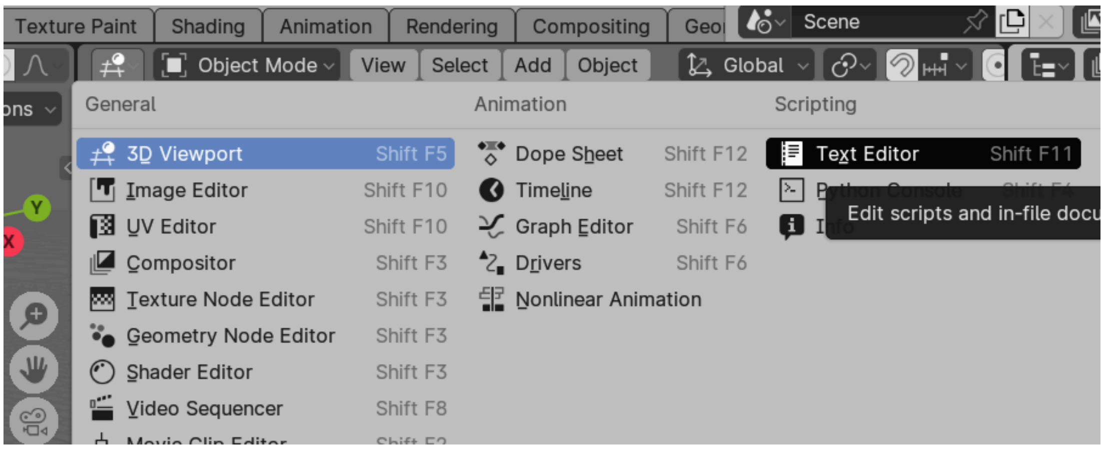
### Step 2: Open Blender and Set Up the Workspace

1. **Open Blender**:
   - Launch Blender on your computer.
2. **Split the Screen**:
   - **Right-click** on the horizontal line at the top of the Blender window.
   - Select **Vertical Split** to create a split-screen setup in Blender. This will allow you to have two panels open simultaneously: one for the text editor and one for viewing the scene.



### Step 3: Load the Script

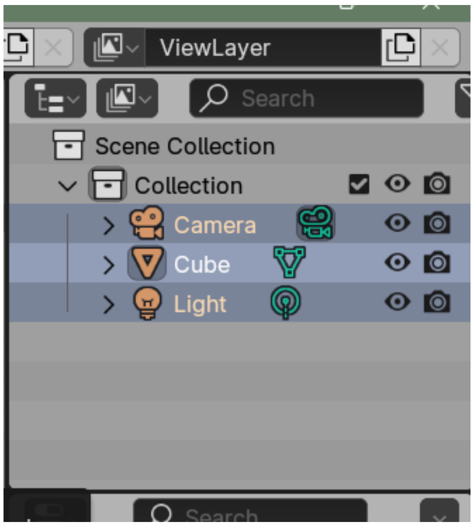
1. **Text Editor Panel**:
   - In one of the panels, switch to the **Text Editor** by clicking the editor type dropdown (top-left corner of the panel) and selecting **Text Editor**.



2. **Copy and Paste the Script**:
   - Copy the entire content of the script you intend to run.
   - Paste it into the text editor panel in Blender.

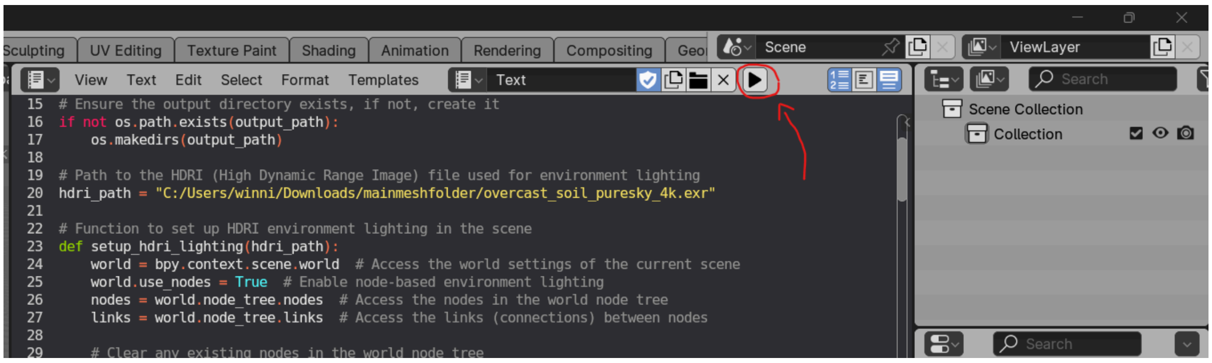
**Step 4: Clear the Scene**

1. **Clear Existing Objects**:
   - In the other panel (3D Viewport), select all the existing objects in the scene, including lights, cameras, and any default objects.
   - Press X to delete them, ensuring that the scene is empty and ready for the script to load the new meshes and set up the environment.



**Step 5: Run the Script**

1. **Execute the Script**:
   - Go back to the text editor panel.
   - At the top of the panel, click on the **Run Script** button. This action will start the automated process.



```
# Ensure the output directory exists, if not, create it
if not os.path.exists(output_path):
    os.makedirs(output_path)

# Path to the HDRI (High Dynamic Range Image) file used for environment lighting
hdri_path = "C:/Users/winni/Downloads/mainmeshfolder/overcast_soil_puresky_4k.exr"

# Function to set up HDRI environment lighting in the scene
def setup_hdri_lighting(hdri_path):
    world = bpy.context.scene.world  # Access the world settings of the current scene
    world.use_nodes = True  # Enable node-based environment lighting
    nodes = world.node_tree.nodes  # Access the nodes in the world node tree
    links = world.node_tree.links  # Access the links (connections) between nodes

    # Clear any existing nodes in the world node tree
```

**Step 6: Review the Outputs**

1. **Check the Output Folder**:

    - Once the script has finished running, navigate to your output folder.
    - You will find subfolders containing the rendered images and animations, organised according to the subfolders in your input directory.

## Conclusion

This script automates the rendering process in Blender, making it easier to manage large collections of 3D models. By customising the script, you can adapt it to different projects, ensuring consistent and high-quality renders with minimal manual intervention. This documentation provides a clear understanding of how each part of the script contributes to the overall workflow, enabling efficient and effective use of Blender for batch rendering tasks.