Skip to content

RenderEngine(bpy_struct)

Simple Render Engine

```
import bpy
import array


class CustomRenderEngine(bpy.types.RenderEngine):
    # These three members are used by blender to set up the
    # RenderEngine; define its internal name, visible name and capabilities.
    bl_idname = "CUSTOM"
    bl_label = "Custom"
    bl_use_preview = True

    # Init is called whenever a new render engine instance is created. Multiple
    # instances may exist at the same time, for example for a viewport and final
    # render.
    # Note the generic arguments signature, and the call to the parent class
    # `__init__` methods, which are required for Blender to create the underlying
    # `RenderEngine` data.
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.scene_data = None
        self.draw_data = None

    # When the render engine instance is destroy, this is called. Clean up any
    # render engine data here, for example stopping running render threads.
    def __del__(self):
        # Own delete code...
        super().__del__()

    # This is the method called by Blender for both final renders (F12) and
    # small preview for materials, world and lights.
    def render(self, depsgraph):
        scene = depsgraph.scene
        scale = scene.render.resolution_percentage / 100.0
        self.size_x = int(scene.render.resolution_x * scale)
        self.size_y = int(scene.render.resolution_y * scale)

        # Fill the render result with a flat color. The frame-buffer is
        # defined as a list of pixels, each pixel itself being a list of
        # R,G,B,A values.
        if self.is_preview:
            color = [0.1, 0.2, 0.1, 1.0]
        else:
            color = [0.2, 0.1, 0.1, 1.0]

        pixel_count = self.size_x * self.size_y
        rect = [color] * pixel_count

        # Here we write the pixel values to the RenderResult
        result = self.begin_result(0, 0, self.size_x, self.size_y)
        layer = result.layers[0].passes["Combined"]
        layer.rect = rect
        self.end_result(result)

    # For viewport renders, this method gets called once at the start and
    # whenever the scene or 3D viewport changes. This method is where data
    # should be read from Blender in the same thread. Typically a render
    # thread will be started to do the work while keeping Blender responsive.
    def view_update(self, context, depsgraph):
        region = context.region
        view3d = context.space_data
        scene = depsgraph.scene

        # Get viewport dimensions
        dimensions = region.width, region.height

        if not self.scene_data:
            # First time initialization
            self.scene_data = []
            first_time = True

            # Loop over all datablocks used in the scene.
            for datablock in depsgraph.ids:
                pass
        else:
            first_time = False

            # Test which datablocks changed
            for update in depsgraph.updates:
                print("Datablock updated: ", update.id.name)

            # Test if any material was added, removed or changed.
            if depsgraph.id_type_updated('MATERIAL'):
                print("Materials updated")

        # Loop over all object instances in the scene.
        if first_time or depsgraph.id_type_updated('OBJECT'):
            for instance in depsgraph.object_instances:
                pass

    # For viewport renders, this method is called whenever Blender redraws
    # the 3D viewport. The renderer is expected to quickly draw the render
    # with OpenGL, and not perform other expensive work.
    # Blender will draw overlays for selection and editing on top of the
    # rendered image automatically.
    def view_draw(self, context, depsgraph):
        # Lazily import GPU module, so that the render engine works in
        # background mode where the GPU module can't be imported by default.
        import gpu

        region = context.region
        scene = depsgraph.scene

        # Get viewport dimensions
        dimensions = region.width, region.height

        # Bind shader that converts from scene linear to display space,
        gpu.state.blend_set('ALPHA_PREMULT')
        self.bind_display_space_shader(scene)

        if not self.draw_data or self.draw_data.dimensions != dimensions:
            self.draw_data = CustomDrawData(dimensions)

        self.draw_data.draw()

        self.unbind_display_space_shader()
        gpu.state.blend_set('NONE')


class CustomDrawData:
    def __init__(self, dimensions):
        import gpu

        # Generate dummy float image buffer
        self.dimensions = dimensions
        width, height = dimensions

        pixels = width * height * array.array('f', [0.1, 0.2, 0.1, 1.0])
        pixels = gpu.types.Buffer('FLOAT', width * height * 4, pixels)

        # Generate texture
        self.texture = gpu.types.GPUTexture((width, height), format='RGBA16F', data=pixels

        # Note: This is just a didactic example.
        # In this case it would be more convenient to fill the texture with:
        # self.texture.clear('FLOAT', value=[0.1, 0.2, 0.1, 1.0])

    def __del__(self):
        del self.texture

    def draw(self):
        from gpu_extras.presets import draw_texture_2d
        draw_texture_2d(self.texture, (0, 0), self.texture.width, self.texture.height)


# RenderEngines also need to tell UI Panels that they are compatible with.
# We recommend to enable all panels marked as BLENDER_RENDER, and then
# exclude any panels that are replaced by custom panels registered by the
# render engine, or that are not supported.
def get_panels():
    exclude_panels = {
        'VIEWLAYER_PT_filter',
        'VIEWLAYER_PT_layer_passes',
    }

    panels = []
    for panel in bpy.types.Panel.__subclasses__():
        if hasattr(panel, 'COMPAT_ENGINES') and 'BLENDER_RENDER' in panel.COMPAT_ENGINES:
            if panel.__name__ not in exclude_panels:
                panels.append(panel)

    return panels


def register():
    # Register the RenderEngine
    bpy.utils.register_class(CustomRenderEngine)

    for panel in get_panels():
        panel.COMPAT_ENGINES.add('CUSTOM')


def unregister():
    bpy.utils.unregister_class(CustomRenderEngine)

    for panel in get_panels():
        if 'CUSTOM' in panel.COMPAT_ENGINES:
            panel.COMPAT_ENGINES.remove('CUSTOM')


if __name__ == "__main__":
    register()
```

GPU Render Engine

```
import bpy


class CustomGPURenderEngine(bpy.types.RenderEngine):
    bl_idname = "CUSTOM_GPU"
    bl_label = "Custom GPU"

    # Request a GPU context to be created and activated for the render method.
    # This may be used either to perform the rendering itself, or to allocate
    # and fill a texture for more efficient drawing.
    bl_use_gpu_context = True

    def render(self, depsgraph):
        # Lazily import GPU module, since GPU context is only created on demand
        # for rendering and does not exist on register.
        import gpu

        # Perform rendering task.
        pass


def register():
    bpy.utils.register_class(CustomGPURenderEngine)


def unregister():
    bpy.utils.unregister_class(CustomGPURenderEngine)
```

```
if __name__ == "__main__":
    register()
```

base class — `bpy_struct`

subclasses — `HydraRenderEngine`

class bpy.types.**RenderEngine(bpy_struct)**

Render engine

bl_idname

TYPE:
string, default “”, (never None)

bl_label

TYPE:
string, default “”, (never None)

bl_use_alembic_procedural

Support loading Alembic data at render time

TYPE:
boolean, default False

bl_use_custom_freestyle

Handles freestyle rendering on its own, instead of delegating it to EEVEE

TYPE:
boolean, default False

bl_use_eevee_viewport

Uses EEVEE for viewport shading in Material Preview shading mode

TYPE:
boolean, default False

bl_use_gpu_context

Enable OpenGL context for the render method, for engines that render using OpenGL

TYPE:
boolean, default False

bl_use_image_save

Save images/movie to disk while rendering an animation. Disabling image saving is only supported when bl_use_postprocess is also disabled.

TYPE:
boolean, default True

bl_use_materialx

Use MaterialX for exporting materials to Hydra

TYPE:
boolean, default False

bl_use_postprocess

Apply compositing on render results

TYPE:

boolean, default False

bl_use_preview

Render engine supports being used for rendering previews of materials, lights and worlds

TYPE:

boolean, default False

bl_use_shading_nodes_custom

Don’t expose Cycles and EEVEE shading nodes in the node editor user interface, so separate nodes can be used instead

TYPE:

boolean, default True

bl_use_spherical_stereo

Support spherical stereo camera models

TYPE:

boolean, default False

bl_use_stereo_viewport

Support rendering stereo 3D viewport

TYPE:

boolean, default False

camera_override

TYPE:

`Object`, (readonly)

is_animation

TYPE:

boolean, default False

is_preview

TYPE:

boolean, default False

layer_override

TYPE:

boolean array of 20 items, default (False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False)

render

TYPE:

`RenderSettings`, (readonly)

resolution_x

TYPE:

int in [-inf, inf], default 0, (readonly)

resolution_y

TYPE:

int in [-inf, inf], default 0, (readonly) int in [-inf, inf], default 0, (readonly)

temporary_directory

TYPE:
string, default “”, (readonly, never None)

use_highlight_tiles

TYPE:
boolean, default False

update(*, data=None, depsgraph=None)

Export scene data for render

render(depsgraph)

Render scene into an image

render_frame_finish()

Perform finishing operations after all view layers in a frame were rendered

draw(context, depsgraph)

Draw render image

bake(depsgraph, object, pass_type, pass_filter, width, height)

Bake passes

PARAMETERS:

- **pass_type** (enum in Bake Pass Type Items) – Pass, Pass to bake
- **pass_filter** (*int in [0, inf]*) – Pass Filter, Filter to combined, diffuse, glossy and transmission passes
- **width** (*int in [0, inf]*) – Width, Image width
- **height** (*int in [0, inf]*) – Height, Image height

view_update(context, depsgraph)

Update on data changes for viewport render

view_draw(context, depsgraph)

Draw viewport render

update_script_node(*, node=None)

Compile shader script node

update_render_passes(*, scene=None, renderlayer=None)

Update the render passes that will be generated

tag_redraw()

Request redraw for viewport rendering

tag_update()

Request update call for viewport rendering

begin_result(x, y, w, h, *, layer='', view='')

Create render result to write linear floating-point render layers and passes

PARAMETERS:

- **x** (*int in [0, inf]*) – X
- **y** (*int in [0, inf]*) – Y

- **w** (*int in [0, inf]*) – Width
- **h** (*int in [0, inf]*) – Height
- **layer** (*string, (optional, never None)*) – Layer, Single layer to get render result for
- **view** (*string, (optional, never None)*) – View, Single view to get render result for

RETURNS:

Result

RETURN TYPE:

`RenderResult`

update_result(result)

Signal that pixels have been updated and can be redrawn in the user interface

PARAMETERS:

result (`RenderResult`) – Result

end_result(result, *, cancel=False, highlight=False, do_merge_results=False)

All pixels in the render result have been set and are final

PARAMETERS:

- **result** (`RenderResult`) – Result
- **cancel** (*boolean, (optional)*) – Cancel, Don't mark tile as done, don't merge results unless forced
- **highlight** (*boolean, (optional)*) – Highlight, Don't mark tile as done yet
- **do_merge_results** (*boolean, (optional)*) – Merge Results, Merge results even if cancel=true

add_pass(name, channels, chan_id, *, layer='')

Add a pass to the render layer

PARAMETERS:

- **name** (*string, (never None)*) – Name, Name of the Pass, without view or channel tag
- **channels** (*int in [0, inf]*) – Channels
- **chan_id** (*string, (never None)*) – Channel IDs, Channel names, one character per channel
- **layer** (*string, (optional, never None)*) – Layer, Single layer to add render pass to

get_result()

Get final result for non-pixel operations

RETURNS:

Result

RETURN TYPE:

`RenderResult`

test_break()

Test if the render operation should been canceled, this is a fast call that should be used regularly for responsiveness

RETURNS:

Break

RETURN TYPE:

boolean

pass_by_index_get(layer, index)

pass_by_index_get

PARAMETERS:

- **layer** (*string, (never None)*) – Layer, Name of render layer to get pass for

- **layer** (*string, (never None)*) – Layer, Name of render layer to get pass for
- **index** (*int in [0, inf]*) – Index, Index of pass to get

RETURNS:

Index, Index of pass to get

RETURN TYPE:

`RenderPass`

active_view_get()

active_view_get

RETURNS:

View, Single view active

RETURN TYPE:

string, (never None)

active_view_set(view)

active_view_set

PARAMETERS:

view (*string, (never None)*) – View, Single view to set as active

camera_shift_x(camera, *, use_spherical_stereo=False)

camera_shift_x

PARAMETERS:

use_spherical_stereo (*boolean, (optional)*) – Spherical Stereo

RETURNS:

Shift X

RETURN TYPE:

float in [0, inf]

camera_model_matrix(camera, *, use_spherical_stereo=False)

camera_model_matrix

PARAMETERS:

use_spherical_stereo (*boolean, (optional)*) – Spherical Stereo

RETURNS:

Model Matrix, Normalized camera model matrix

RETURN TYPE:

`mathutils.Matrix` of 4 * 4 items in [-inf, inf]

use_spherical_stereo(camera)

use_spherical_stereo

RETURNS:

Spherical Stereo

RETURN TYPE:

boolean

update_stats(stats, info)

Update and signal to redraw render status text

PARAMETERS:

- **stats** (*string, (never None)*) – Stats
- **info** (*string, (never None)*) – Info

frame_set(frame, subframe)

Evaluate scene at a different frame (for motion blur)

PARAMETERS:

- **frame** (*int in [-inf, inf]*) – Frame
- **subframe** (*float in [0, 1]*) – Subframe

update_progress(progress)

Update progress percentage of render

PARAMETERS:

progress (*float in [0, 1]*) – Percentage of render that's done

update_memory_stats(*, memory_used=0.0, memory_peak=0.0)

Update memory usage statistics

PARAMETERS:

- **memory_used** (*float in [0, inf], (optional)*) – Current memory usage in megabytes
- **memory_peak** (*float in [0, inf], (optional)*) – Peak memory usage in megabytes

report(type, message)

Report info, warning or error messages

PARAMETERS:

- **type** (enum set in Wm Report Items) – Type
- **message** (*string, (never None)*) – Report Message

error_set(message)

Set error message displaying after the render is finished

PARAMETERS:

message (*string, (never None)*) – Report Message

bind_display_space_shader(scene)

Bind GLSL fragment shader that converts linear colors to display space colors using scene color management settings

unbind_display_space_shader()

Unbind GLSL display space shader, must always be called after binding the shader

support_display_space_shader(scene)

Test if GLSL display space shader is supported for the combination of graphics card and scene settings

RETURNS:

Supported

RETURN TYPE:

boolean

get_preview_pixel_size(scene)

Get the pixel size that should be used for preview rendering

RETURNS:

Pixel Size

RETURN TYPE:

int in [1, 8]

free_blender_memory()

Free Blender side memory of render engine

tile_highlight_set(x, y, width, height, highlight)

Set highlighted state of the given tile

PARAMETERS:

- **x** (*int in [0, inf]*) – X
- **y** (*int in [0, inf]*) – Y
- **width** (*int in [0, inf]*) – Width
- **height** (*int in [0, inf]*) – Height
- **highlight** (*boolean*) – Highlight

tile_highlight_clear_all()

The temp directory used by Blender

register_pass(scene, view_layer, name, channels, chanid, type)

Register a render pass that will be part of the render with the current settings

PARAMETERS:

- **name** (*string, (never None)*) – Name
- **channels** (*int in [1, 8]*) – Channels
- **chanid** (*string, (never None)*) – Channel IDs
- **type** (*enum in ['VALUE', 'VECTOR', 'COLOR']*) – Type

classmethod bl_rna_get_subclass(id, default=None)

PARAMETERS:

id (*str*) – The RNA type identifier.

RETURNS:

The RNA type or default when not found.

RETURN TYPE:

`bpy.types.Struct` subclass

classmethod bl_rna_get_subclass_py(id, default=None)

PARAMETERS:

id (*str*) – The RNA type identifier.

RETURNS:

The class or default when not found.

RETURN TYPE:

type

Inherited Properties

- `bpy_struct.id_data`

Inherited Functions

- `bpy_struct.as_pointer`
- `bpy_struct.items`

- bpy_struct.as_pointer
- bpy_struct.driver_add
- bpy_struct.driver_remove
- bpy_struct.get
- bpy_struct.id_properties_clear
- bpy_struct.id_properties_ensure
- bpy_struct.id_properties_ui
- bpy_struct.is_property_hidden
- bpy_struct.is_property_overridable_library
- bpy_struct.is_property_readonly
- bpy_struct.is_property_set
- bpy_struct.items
- bpy_struct.keyframe_delete
- bpy_struct.keyframe_insert
- bpy_struct.keys
- bpy_struct.path_from_id
- bpy_struct.path_resolve
- bpy_struct.pop
- bpy_struct.property_overridable_library_set
- bpy_struct.property_unset
- bpy_struct.type_recast
- bpy_struct.values

Previous
RemeshModifier(Modifier)

Ne
RenderLayer(bpy_stru

Report issue on this page