Documentation of Python Code for BatteryCT Blender UI and Cell Generation

F. Bisinger, E. Grenz, I. Schopf

2024-07-03

1 Introduction

This document provides a detailed explanation of the Python code used for the BatteryCT Blender UI and cell generation. The code is designed to create and export 3D models of battery cells in Blender, with various parameters and settings configurable through a custom user interface (UI).

2 Code Description

2.1 Imports and Initialization

The script begins by importing necessary libraries and defining a class `Modeling` which contains all the properties and methods for battery modeling.

```python
import bpy
import json
import numpy as np
import os
from datetime import datetime
import bmesh
```

2.2 Modeling Class

The `Modeling` class initializes various parameters related to the battery components such as anode, cathode, and their coatings. It also sets up the export paths and creates a timestamp for file naming.

```python
class Modeling:
    def __init__(self):
        self.iterations = 1
        self.seperator_height = 0.001
        self.anode_overhang_bool = True
        self.export_inner_battery_bool = True
        self.export_housing_bool = True
        self.cut_housing_zy_bool = True
        self.cut_housing_zx_bool = True
        self.bending_bool = True
        self.max_angle = 15.0
        self.min_angle = -15.0
        self.x_variation = 1e-3
        self.y_variation = 1e-3
        self.current_datetime = datetime.now().strftime("%Y%m%d_%H%
            M%S")
        self.export_folder = os.path.normpath("Model_" + self.
            current_datetime)
        self.export_path = os.path.join(r"C:/", self.export_folder)

        self.anode = {
            "length": 0.1015,
            "width": 0.050,
            "height": 0.001,
            "length_tol": 0.000,
            "max_overhang": 0.0068,
            "min_overhang": 0.0023,
            "width_tol": 0.000,
            "height_tol": 0.000,
            "amount": 10,
            "color": (1, 0, 0, 1),
        }
        # More initializations...
        print("PARAMS␣INIT␣DONE")
```

2.3 Inner Battery Creation and Export

The create_and_export_inner_battery method generates the inner components of the battery and exports them as .stl files. It handles the positioning, scaling, and configuration of the battery components within Blender.

```
def create_and_export_inner_battery(self, j, parameters, name):
    for i in range(parameters["amount"]):
        deviations = self.generate_deviations(parameters, self.
            x_variation, self.y_variation)
        if parameters in [self.lower_anode_coating, self.
            upper_anode_coating]:
            x_position = self.data["anode"]["anode_position"]["x"][
                i]
            y_position = self.data["anode"]["anode_position"]["y"][
                i]
        else:
            x_position = deviations["x_position"]
            y_position = deviations["y_position"]
        z_position = parameters["position"]["z_position"] + i * (
            parameters["position"]["z_distance"])
        dimensions = self.generate_dimensions(name, parameters,
            deviations)
        bpy.ops.mesh.primitive_cube_add(size=1, enter_editmode=
            False, align='WORLD', location=(x_position, y_position,
             z_position))
        obj = bpy.context.object
        obj.name = f"{name}_{i}"
        obj.scale = (dimensions["length"], dimensions["width"],
            dimensions["height"])
        self.bend_object(obj.name, parameters, dimensions, name, i)
        obj.active_material = bpy.data.materials.new(name=f"Color_{
            j}_{i}")
        obj.active_material.diffuse_color = parameters["color"]
        bpy.ops.object.origin_set(type='ORIGIN_GEOMETRY', center='
            BOUNDS')
    self.export_inner_battery(name, i, j)
```

2.4 Detailed Description of Cell Generation

The core process of generating the cell involves multiple steps. Each step ensures that the created model is both accurate to the specified parameters and varied enough to simulate real-world production variations.

2.4.1 Generating Deviations

The `generate_deviations` method is crucial for simulating real-world variances in battery cell production. This method creates random deviations for the dimensions and positions of the battery components. It uses a uniform distribution within a specified range to add randomness to the x and y positions of each component.

```
def generate_deviations(self, parameters, x_var, y_var):
    deviations = {
        "x_position": parameters["position"]["x_position"] + np.
            random.uniform(-x_var, x_var),
        "y_position": parameters["position"]["y_position"] + np.
            random.uniform(-y_var, y_var)
    }
    return deviations
```

2.4.2 Generating Dimensions

The `generate_dimensions` method calculates the actual dimensions of the battery components. This method applies tolerances to the base dimensions of each component to simulate slight differences that occur during manufacturing.

```
def generate_dimensions(self, name, parameters, deviations):
    dimensions = {
        "length": parameters["length"] + np.random.uniform(-
            parameters["length_tol"], parameters["length_tol"]),
        "width": parameters["width"] + np.random.uniform(-
            parameters["width_tol"], parameters["width_tol"]),
        "height": parameters["height"] + np.random.uniform(-
            parameters["height_tol"], parameters["height_tol"])
    }
    return dimensions
```

2.4.3 Adding Components to Blender

In the `create_and_export_inner_battery` method, after calculating the deviations and dimensions, the battery components are added to Blender. The components are positioned based on the calculated deviations and scaled to the generated dimensions.

```
bpy.ops.mesh.primitive_cube_add(size=1, enter_editmode=False, align
    ='WORLD', location=(x_position, y_position, z_position))
obj = bpy.context.object
obj.name = f"{name}_{i}"
obj.scale = (dimensions["length"], dimensions["width"], dimensions[
    "height"])
```

2.4.4 Applying Bending Transformations

If bending is enabled, the `bend_object` method is called to apply bending transformations to the components. This step ensures that the components reflect real-world deformations that might occur during the manufacturing process.

```
def bend_object(self, obj_name, type, dimensions, name, i):
    if self.bending_bool:
        obj = bpy.context.active_object
        mesh = obj.data
        bm = bmesh.new()
        bm.from_mesh(mesh)
        edges = bm.edges
        bmesh.ops.subdivide_edges(bm, edges=edges, cuts=140,
            use_grid_fill=True)
        bm.to_mesh(mesh)
        bm.free()
        mesh.update()
        vertex_group = obj.vertex_groups.new(name='My␣Vertex␣Group'
            )
        for v in mesh.vertices:
            world_vertex_co = obj.matrix_world @ v.co
            relative_x = (world_vertex_co.x - obj.bound_box[0][0])
                / (obj.bound_box[7][0] - obj.bound_box[0][0])
            if 0.95 <= relative_x <= 1:
                vertex_group.add([v.index], 1.0, 'ADD')
        modifier = obj.modifiers.new(name="Deform", type='
            SIMPLE_DEFORM')
        modifier.deform_method = 'BEND'
        modifier.deform_axis = 'Z'
        modifier.vertex_group = 'My␣Vertex␣Group'
        modifier.angle = bending_x_pos * (np.pi)/180 * 360/45
```

2.4.5 Assigning Materials

Each component is assigned a material to distinguish it within Blender. The materials are created with specific colors defined in the parameters.

```
obj.active_material = bpy.data.materials.new(name=f"Color_{j}_{i}")
obj.active_material.diffuse_color = parameters["color"]
```

2.4.6 Exporting the Components

Finally, the `export_inner_battery` method exports each component as a .stl file. The components are saved with names that include the iteration and component type, ensuring that each file is uniquely identifiable.

```
def export_inner_battery(self, name, i, j):
    if not os.path.exists(self.export_path):
        os.makedirs(self.export_path)
    file_path = os.path.join(self.export_path, f"{name}_{j}_{i}.stl
        ")
    bpy.ops.export_mesh.stl(filepath=file_path, use_selection=True)
```

3 UI Elements

The script defines various UI elements using Blender's `bpy.types` module, including sliders, checkboxes, and file path selectors. These elements allow users to input parameters for the battery models directly within Blender's interface.

```
bpy.types.Scene.num_slider = bpy.props.IntProperty(
    name="Number␣of␣Exported␣Variants",
    description="Adjust␣the␣value␣using␣this␣slider",
    default=1,
    min=1,
    max=1000
)
# Other UI elements...
```

4 Panel and Operators

A custom panel and operators are defined to handle user interactions. The panel displays the UI elements, and operators handle actions like loading configurations, saving configurations, and exporting the battery models.

```
class CustomPanel(bpy.types.Panel):
    bl_label = "BatteryCT␣Configuration"
    bl_idname = "PT_CustomPanel"
    bl_space_type = 'VIEW_3D'
    bl_region_type = 'UI'
    bl_category = 'BatteryCT'
    bl_context = "objectmode"
    def draw(self, context):
        layout = self.layout
        scene = context.scene
        layout.label(text="Load␣Config␣File:")
        col = layout.column(align=True)
        col.prop(scene.path_tool_1, "file", text="")
        layout.operator("object.load_config_button")
        layout.operator("object.save_config_button")
        layout.label(text="Output␣Folder␣Location:")
        col = layout.column(align=True)
        col.prop(scene.path_tool_2, "path", text="")
        layout.prop(scene, "num_slider")
        layout.operator("object.export_button")
```

5 Main Execution

The script's main execution block registers the custom classes and properties with Blender.

```
classes = (
    CustomPanel,
    DataButton,
    PathProperty,
    FileProperty,
    LoadConfig,
    SaveConfig,
    Export
)

def register():
    from bpy.utils import register_class
    for cls in classes:
        register_class(cls)
    bpy.types.Scene.path_tool_1 = bpy.props.PointerProperty(type=
        FileProperty)
    bpy.types.Scene.path_tool_2 = bpy.props.PointerProperty(type=
        PathProperty)

def unregister():
    from bpy.utils import unregister_class
    for cls in reversed(classes):
        unregister_class(cls)
    del bpy.types.Scene.path_tool_1
    del bpy.types.Scene.path_tool_2

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

6 Conclusion

This script provides a robust framework for generating and exporting parametric battery cell models in Blender. The detailed configuration options and UI elements make it user-friendly, while the use of Blender's powerful modeling and scripting capabilities ensures flexibility and scalability.