

An Exploratory Analysis of Blender Scripting Activities and 3D Model Performance

Yoseph Satria Praka^{1,*}, Ade Rahma Yuly¹

¹Digital Multimedia Engineering Study Program, Jakarta State Polytechnic, Indonesia
E-mail: yoseph.satria.praka.tik23@stu.pnj.ac.id *

*Corresponding Author

ABSTRACT

This study explores Blender scripting for 3D modelling through an activity-based analysis of a simple 3D object, complemented by performance testing of the resulting scene. The research employed an exploratory descriptive approach in which individual Blender Python operations were recorded and analysed based on their functions, object types, transformation parameters, and activity outcomes. The resulting scripting log contained 413 recorded activities, with 50 representative records presented for analysis. The findings document a sequence of modelling operations, including scene initialisation, primitive object creation, transformations, and modifier application. Performance testing further characterised the generated scene, which was produced in 0.9778 seconds and comprised 149 objects, 21,464 vertices, 42,352 edges, 21,168 faces, and 42,368 triangles. The scene was configured to use the Cycles rendering engine at a resolution of 1920×1080 pixels. These results demonstrate how activity logging and performance measurements can be combined to describe the modelling process and the characteristics of the resulting 3D object. Rather than proposing a new modelling method or optimisation technique, this study provides an exploratory account of Blender scripting activities and their associated scene characteristics. Future research may extend this approach to more complex objects, compare scripted and manual modelling workflows, and examine performance across different modelling tasks.

This is an open access article under the CC-BY-SA license.



ARTICLE INFO

Article history

Received:

14 August 2026

Revised:

06 October 2026

Accepted:

07 October 2026

Keywords

Blender
Exploratory analysis
performance
Python Scripting
3D Modelling.

1. Introduction

Three-dimensional (3D) modelling has become an important component in various fields, including education [1], architecture [2], digital media [3], visualisation [3], and scientific research [4]. The availability of accessible 3D modelling software enables users to construct digital representations of objects through both conventional and computational approaches. One of the widely used tools for this purpose is Blender, an open-source 3D creation software that provides functionalities for modelling, texturing, lighting, animation, rendering, and scripting. In addition to its graphical interface, Blender provides Python scripting capabilities that allow users to perform modelling operations programmatically [5]. Previous work has demonstrated the use of Blender for creating digital representations of public buildings, where fundamental modelling, texturing, and lighting techniques



were applied to produce detailed visual representations. The study also highlighted Blender's accessibility and flexibility for educational and professional applications [6].

The use of Blender has also expanded into scientific and specialised applications. In [7], Blender was used to construct models of resident space objects and generate synthetic light curves. The study utilised Python scripting and GPU capabilities as part of the modelling process, demonstrating that Blender can support computational workflows beyond conventional visual modelling. Similarly, [8] explored Geometry Nodes and modifier tools for reconstructing historical castle structures. The study focused on parametric modelling to facilitate flexible modification and customisation of complex architectural objects. These studies demonstrate that Blender provides various modelling capabilities that can be adapted to different modelling requirements and application domains.

Blender has also been applied to conventional 3D modelling and digital asset production. Research on the design of a television studio interior used Blender's solid geometry modelling techniques and basic geometric primitives, including cubes, spheres, cylinders, and cones, as the basis for constructing the interior environment [9]. Another study on the production of Minecraft comic strips examined the layout process of 3D assets using Blender as part of the production workflow. The research focused on the arrangement and presentation of 3D assets for visual media production [10]. These studies indicate that Blender can be utilised to construct and organise 3D objects for different purposes, ranging from architectural and interior visualisation to digital content production.

Despite these applications, previous studies have generally focused on the resulting 3D models, their application contexts, or specific modelling techniques. Studies involving scripting and procedural features have also tended to emphasise their application to specialised or relatively complex modelling tasks. As a result, the individual scripting activities performed during the modelling process have received comparatively less attention as an object of analysis. In particular, there is an opportunity to examine basic Blender Python operations as individual activities during the construction of simple 3D objects. Operations such as object creation, transformation, modifier application, and shading can be observed directly from the scripting process and represented as structured activity data [11].

Examining these activities is relevant because a 3D model is not produced through a single operation, but through a sequence of individual modelling activities. Each activity may contain specific attributes, such as the operation performed, object type, location, rotation, scale, parameters, and resulting action. By documenting these activities, the modelling process can be represented not only through its final visual output but also through the sequence of operations used to construct the object. Simple 3D objects provide an appropriate scope for this exploration because their relatively straightforward geometry allows individual scripting activities to be identified and documented more clearly.

To address this research gap, this study does not propose a new modelling method, framework, or optimisation technique. Instead, it focuses on exploring and describing Blender scripting activities during the modelling of simple 3D objects. Each Blender Python operation is treated as an individual activity and recorded based on its observable attributes. The collected activity data are analysed quantitatively to examine the types, frequency, and distribution of modelling operations, including object creation, transformations, and other scripting-related actions. The analysis considers observable attributes such as operation type, object type, location, rotation, scale, parameters, and activity outcome. In addition to the activity-based analysis, this study includes performance testing to characterise the resulting 3D scene. The performance evaluation considers generation time, scene complexity, geometric structure, asset usage, estimated mesh geometry memory, and rendering configuration. By combining

scripting activity records with performance measurements, this study provides a descriptive account of both the operations performed during modelling and the characteristics of the generated scene. Therefore, this study aims to explore Blender scripting for 3D modelling through an activity-based analysis of simple objects, complemented by performance testing of the resulting model.

2. Method

This study employed an exploratory descriptive approach to examine Blender scripting activities in the modelling of simple 3D objects, complemented by performance testing of the resulting scene. Blender was selected as the primary modelling and scripting platform because it supports conventional 3D modelling and Python scripting through its bpy API. Python scripts were used to construct simple 3D objects through operations such as primitive creation, object transformation, modifier application, shading, lighting, and camera setup [12], [13]. The objects were limited to simple forms to facilitate the identification and analysis of individual scripting activities without introducing the complexity of highly detailed models. The main unit of analysis was an individual Blender Python operation, with each operation treated as a measurable activity in the modelling process.

Activity data were collected directly from the Python scripts executed during modelling. Each bpy operation was recorded sequentially, together with its operation type, object or primitive type, location, rotation, scale, relevant parameters, modifier information, and activity outcome. Outcomes were categorised according to the operation performed, such as “Created,” “Applied,” “Modified,” or “Shaded.” The records were compiled into a structured dataset to support quantitative analysis. In parallel, performance data were collected to characterise the resulting scene, including generation time, evaluation time, object counts, geometric statistics, asset usage, estimated mesh geometry memory, and rendering configuration [14], [15], [16]. Geometric statistics comprised the numbers of vertices, edges, faces, and triangles, while asset usage included the numbers of unique meshes and materials. The rendering configuration documented the rendering engine and output resolution.

The research procedure comprised several stages: selecting simple 3D objects, preparing the Blender environment, constructing the objects through Python scripting, recording each scripting operation, organising the collected data, and analysing the results. The activity records were examined to identify patterns in operation types, frequencies, object types, and transformation parameters. Performance testing was conducted separately to summarise the generation time and structural characteristics of the resulting scene. The findings from both analyses were then interpreted to describe the scripting activities and the characteristics of the generated model. The overall research procedure is presented in Fig. 1.

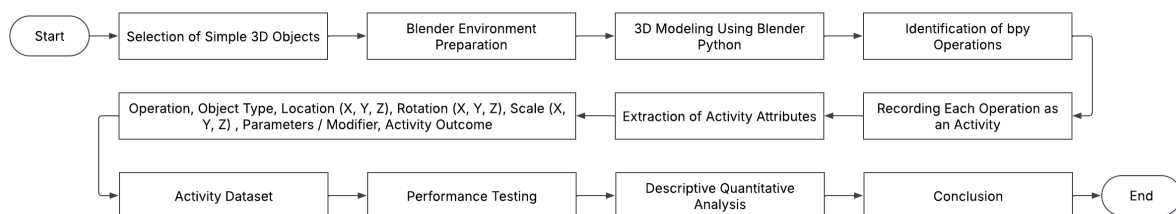


Fig. 1. Research Stages for Activity-Based Analysis of Blender Scripting

The operational variables were defined according to the observable characteristics of the scripting activities and the resulting 3D scene. The primary variable, scripting activity, referred to an individual Blender Python operation. Its attributes included operation type, object type, location coordinates, rotation values, scale values, operation parameters, modifier information, and activity outcome [5], [11]. These attributes enabled each operation to be described without assigning subjective scores to the visual quality of the model.

For performance testing, the variables included generation time, evaluation time, scene complexity, geometric complexity, asset usage, estimated mesh geometry memory, and rendering configuration. Generation time measured the duration required to execute the model-generation process, whereas evaluation time represented the duration reported by the performance-monitoring script for its evaluation operation. Scene complexity was described by the numbers of total objects, mesh objects, cameras, and lights. Geometric complexity was represented by the numbers of vertices, edges, faces, and triangles. Asset usage was measured through the number of unique meshes and materials, while memory usage was represented by the reported mesh geometry memory estimate. The rendering engine and output resolution were recorded to document the scene configuration. These measures were used as descriptive indicators rather than as a comprehensive benchmark of rendering performance.

Data analysis was conducted using descriptive quantitative analysis [17]. For the activity dataset, operation frequencies and object-type distributions were examined to identify the most common modelling actions. Location, rotation, and scale values were analysed to describe the spatial arrangement and transformations of objects. For the performance dataset, the recorded measurements were summarised to characterise generation time, scene complexity, geometric structure, asset usage, estimated memory requirements, and rendering configuration [18], [19]. The analysis was conducted at two levels: the individual-operation level for scripting activities and the scene level for performance measurements. The resulting findings were used to describe observable patterns without testing causal relationships or comparing the superiority of different modelling techniques. Finally, the activity records and performance measurements were considered together to provide a complementary account of the modelling process and its output. The activity dataset documented the sequence and characteristics of the operations used to construct the objects, while the performance dataset summarised measurable properties of the resulting scene. This approach maintains the exploratory and descriptive scope of the study without proposing a new modelling method, framework, or optimisation technique.

3. Results and Discussion

3.1. Results

The Blender scripting process consisted of a series of modelling activities implemented through the bpy API [20]. Each activity represents a specific operation performed during the construction of the simple 3D object. To provide a clear description of how scripting was used throughout the modelling process, the identified activities are summarised in Table 1. The table presents the sequence number, the function or modelling component represented by each activity, and its corresponding script implementation.

Table 1. Identified Blender Scripting Activities for 3D Modelling

No.	Function / Section	Script Implementation
1	Scene preparation	bpy.ops.object.select_all(action='SELECT')
2	Remove existing objects	bpy.ops.object.delete(use_global=False)

No.	Function / Section	Script Implementation
3	Create basic cube object	bpy.ops.mesh.primitive_cube_add(...)
4	Apply object transformation	bpy.ops.object.transform_apply(...)
5	Apply bevel modifier	bpy.ops.object.modifier_apply(modifier="Realistic Bevel")
6	Create additional cube structure	bpy.ops.mesh.primitive_cube_add(...)
7	Apply transformation to structure	bpy.ops.object.transform_apply(...)
8	Apply bevel to structure	bpy.ops.object.modifier_apply(modifier="Realistic Bevel")
9	Create lower/base structure	bpy.ops.mesh.primitive_cube_add(...)
10	Create side structure	bpy.ops.mesh.primitive_cube_add(...)
11	Create opposite side structure	bpy.ops.mesh.primitive_cube_add(...)
12	Create front structure	bpy.ops.mesh.primitive_cube_add(...)
13	Create upper structure	bpy.ops.mesh.primitive_cube_add(...)
14	Create repeated structural elements	bpy.ops.mesh.primitive_cube_add(...)
15	Create additional structural elements	bpy.ops.mesh.primitive_cube_add(...)
16	Create detailed side elements	bpy.ops.mesh.primitive_cube_add(...)
17	Create symmetrical components	bpy.ops.mesh.primitive_cube_add(...)
18	Create circular component	bpy.ops.mesh.primitive_uv_sphere_add(...)
19	Smooth circular component	bpy.ops.object.shade_smooth()
20	Create additional circular component	bpy.ops.mesh.primitive_uv_sphere_add(...)
21	Create cylindrical component	bpy.ops.mesh.primitive_cylinder_add(...)
22	Smooth cylindrical component	bpy.ops.object.shade_smooth()
23	Create torus component	bpy.ops.mesh.primitive_torus_add(...)
24	Create cylinder detail	bpy.ops.mesh.primitive_cylinder_add(...)
25	Create smaller cylinder detail	bpy.ops.mesh.primitive_cylinder_add(...)
26	Create supporting structure	bpy.ops.mesh.primitive_cube_add(...)
27	Create repeated interior/structural components	bpy.ops.mesh.primitive_cube_add(...)
28	Create vertical structural elements	bpy.ops.mesh.primitive_cylinder_add(...)
29	Create horizontal structural element	bpy.ops.mesh.primitive_cylinder_add(...)
30	Create additional detailed structure	bpy.ops.mesh.primitive_cube_add(...)
31	Create lighting	bpy.ops.object.light_add(type='AREA', ...)
32	Create camera	bpy.ops.object.camera_add(...)
33	Execute Blender script	bpy.ops.text.run_script()

Table 1 shows that Blender scripting in the modelling process can be divided into several functional groups. The first group consists of scene preparation activities, represented by object selection and deletion. These operations establish a clean Blender scene before the modelling process begins. The use of `bpy.ops.object.select_all()` followed by `bpy.ops.object.delete()` demonstrates that scripting can be used not only to create geometry but also to control the initial state of the modelling environment. The second and largest group consists of primitive object creation. The script predominantly uses `bpy.ops.mesh.primitive_cube_add()` to construct the main structural components of the model. Cubes are positioned at different coordinates and subsequently modified to form larger structural elements. This indicates that the simple object was constructed through the combination of multiple basic geometric primitives rather than through a single complex mesh operation.

The script also demonstrates the use of transformation parameters as part of each modelling activity. The location parameter determines the position of an object along the X, Y, and Z axes, while scale determines its dimensions relative to the original primitive. Rotation parameters are also used for objects that require a particular orientation [21], [22]. Consequently, each primitive creation command contains spatial information that can be extracted and represented numerically in the activity dataset. Another important activity is the application of modifiers and transformations. The `transform_apply()` operation is used to apply the current transformation values to the object, while `modifier_apply()` is used to apply the bevel operation. The repeated use of the "Realistic Bevel" modifier indicates that beveling was incorporated into the modelling process to modify the sharp edges of primitive objects. From an

activity-based perspective, these commands represent separate scripting activities even when they are performed immediately after object creation.

The script further includes several geometric primitive types, including cubes, UV spheres, cylinders, and toruses. The use of `primitive_uv_sphere_add()` introduces rounded geometry, while `primitive_cylinder_add()` and `primitive_torus_add()` are used to create cylindrical and circular details. The subsequent `shade_smooth()` commands demonstrate another category of activity related to the visual surface representation of the geometry. The final part of the script involves scene presentation activities, specifically the creation of area lights and a camera. These operations are different from the geometry-generation activities because they do not directly construct the modelled object. Instead, they establish the environment required for viewing and presenting the resulting model. This distinction is important for the activity-based analysis because the total scripting process includes both modelling and scene-setup operations.

The scripting process can be visualised as a sequence of activities, beginning with scene preparation, followed by object creation and transformation, geometry modification, surface refinement, and scene setup. This process is illustrated in Fig. 2.

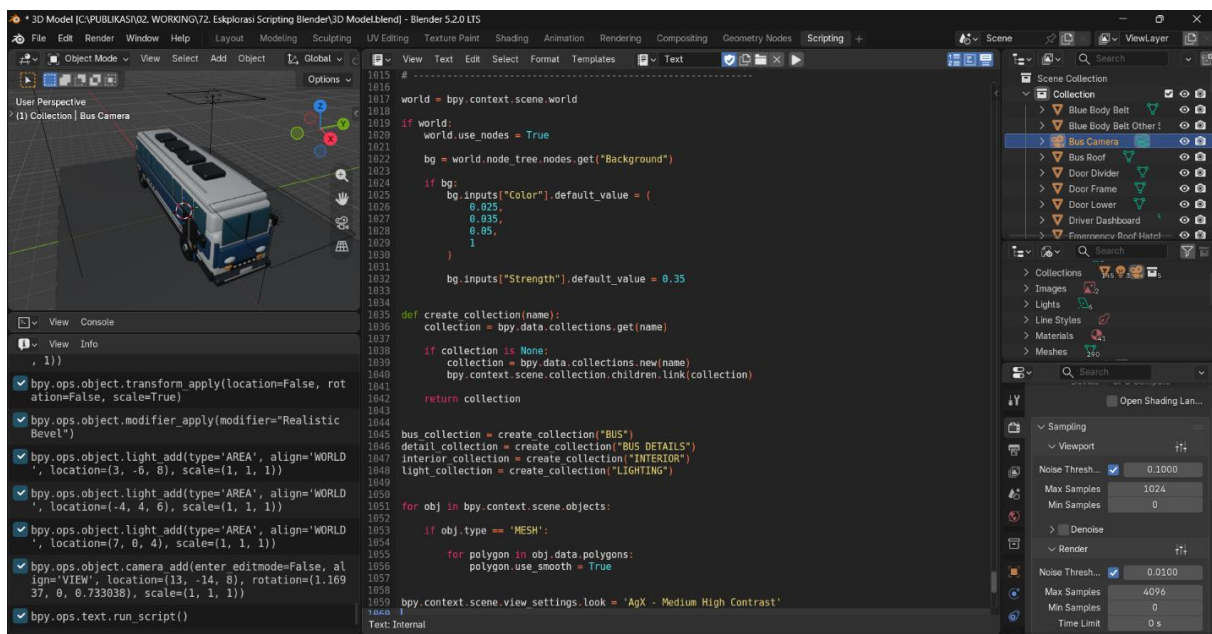


Fig. 2. Blender Scripting Process

The scripting process generated a total of 413 activity records throughout the modelling of the object. Each record represents an individual operation executed through Blender's bpy API, including object creation, transformation application, modifier application, and other modelling-related actions. To maintain readability while still providing a representative overview of the recorded activities, the first 50 records are presented in Table 2. The complete activity log is available at s.pnj.ac.id/LogScriptBlender.

Table 2. Blender Scripting Activity Log

No.	Activity / Operation	Object Type	Location X	Location Y	Location Z	Scale X	Scale Y	Scale Z
1	Object creation	Cube	0.00	0.00	1.65	1.00	1.00	1.00
2	Transform application	Cube	0.00	0.00	1.65	1.00	1.00	1.00
3	Bevel application	Cube	0.00	0.00	1.65	1.00	1.00	1.00
4	Object creation	Cube	0.00	0.00	2.75	1.00	1.00	1.00
5	Transform application	Cube	0.00	0.00	2.75	1.00	1.00	1.00
6	Bevel application	Cube	0.00	0.00	2.75	1.00	1.00	1.00
7	Object creation	Cube	0.00	0.00	0.75	1.00	1.00	1.00
8	Transform application	Cube	0.00	0.00	0.75	1.00	1.00	1.00
9	Bevel application	Cube	0.00	0.00	0.75	1.00	1.00	1.00
10	Object creation	Cube	0.00	-1.385	1.95	1.00	1.00	1.00
11	Transform application	Cube	0.00	-1.385	1.95	1.00	1.00	1.00
12	Bevel application	Cube	0.00	-1.385	1.95	1.00	1.00	1.00
13	Object creation	Cube	0.00	1.385	1.95	1.00	1.00	1.00
14	Transform application	Cube	0.00	1.385	1.95	1.00	1.00	1.00
15	Bevel application	Cube	0.00	1.385	1.95	1.00	1.00	1.00
16	Object creation	Cube	5.72	0.00	1.85	1.00	1.00	1.00
17	Transform application	Cube	5.72	0.00	1.85	1.00	1.00	1.00
18	Bevel application	Cube	5.72	0.00	1.85	1.00	1.00	1.00
19	Object creation	Cube	0.00	0.00	3.55	1.00	1.00	1.00
20	Transform application	Cube	0.00	0.00	3.55	1.00	1.00	1.00
21	Bevel application	Cube	0.00	0.00	3.55	1.00	1.00	1.00
22	Object creation	Cube	-2.80	0.00	3.83	1.00	1.00	1.00
23	Transform application	Cube	-2.80	0.00	3.83	1.00	1.00	1.00
24	Bevel application	Cube	-2.80	0.00	3.83	1.00	1.00	1.00
25	Object creation	Cube	-1.00	0.00	3.83	1.00	1.00	1.00
26	Transform application	Cube	-1.00	0.00	3.83	1.00	1.00	1.00
27	Bevel application	Cube	-1.00	0.00	3.83	1.00	1.00	1.00
28	Object creation	Cube	1.00	0.00	3.83	1.00	1.00	1.00
29	Transform application	Cube	1.00	0.00	3.83	1.00	1.00	1.00
30	Bevel application	Cube	1.00	0.00	3.83	1.00	1.00	1.00
31	Object creation	Cube	2.80	0.00	3.83	1.00	1.00	1.00
32	Transform application	Cube	2.80	0.00	3.83	1.00	1.00	1.00
33	Bevel application	Cube	2.80	0.00	3.83	1.00	1.00	1.00
34	Object creation	Cube	5.78	0.00	2.85	1.00	1.00	1.00
35	Transform application	Cube	5.78	0.00	2.85	1.00	1.00	1.00
36	Bevel application	Cube	5.78	0.00	2.85	1.00	1.00	1.00
37	Object creation	Cube	5.84	0.00	2.85	1.00	1.00	1.00
38	Transform application	Cube	5.84	0.00	2.85	1.00	1.00	1.00
39	Bevel application	Cube	5.84	0.00	2.85	1.00	1.00	1.00
40	Object creation	Cube	5.85	0.00	1.75	1.00	1.00	1.00
41	Transform application	Cube	5.85	0.00	1.75	1.00	1.00	1.00
42	Bevel application	Cube	5.85	0.00	1.75	1.00	1.00	1.00
43	Object creation	Cube	-4.50	-1.375	2.83	1.00	1.00	1.00
44	Transform application	Cube	-4.50	-1.375	2.83	1.00	1.00	1.00
45	Bevel application	Cube	-4.50	-1.375	2.83	1.00	1.00	1.00

No.	Activity / Operation	Object Type	Location X	Location Y	Location Z	Scale X	Scale Y	Scale Z
46	Object creation	Cube	-4.50	1.375	2.83	1.00	1.00	1.00
47	Transform application	Cube	-4.50	1.375	2.83	1.00	1.00	1.00
48	Bevel application	Cube	-4.50	1.375	2.83	1.00	1.00	1.00
49	Object creation	Cube	-3.35	-1.375	2.83	1.00	1.00	1.00
50	Transform application	Cube	-3.35	-1.375	2.83	1.00	1.00	1.00

Based on Table 2, the activity log records each operation sequentially, allowing the modelling process to be analysed at the individual-action level. The recorded parameters include the object type, its X, Y, and Z location, scale values, and the status of the activity. This structure makes it possible to trace how each primitive object was positioned and processed during the modelling process. The initial records show a repeated pattern in which a primitive object is created, followed by transformation application and bevel modification. For example, the creation of a cube is followed by `transform_apply()` and the application of the bevel modifier. This pattern is repeated for several objects with different spatial coordinates, indicating that the model was constructed incrementally from multiple geometric components.

The location values in the log also provide quantitative information about the spatial arrangement of the model. Differences in the X, Y, and Z coordinates indicate the positioning of individual components along the three axes. Meanwhile, the scale values record the dimensional configuration applied to each object. Therefore, the log does not only documents the sequence of scripting activities but also provides numerical data that can be used to examine the spatial characteristics of the modelling process. The 413 recorded activities further demonstrate that a seemingly simple 3D object can involve a considerable number of scripting operations. Rather than evaluating the final model alone, this activity-based log enables the modelling process to be examined in terms of the number, sequence, type, and parameters of individual scripting operations. This supports the main focus of the study, which is to explore Blender scripting activities in the modelling of simple objects rather than to propose a new modelling method or technique.

The performance evaluation was conducted to examine the computational characteristics of the 3D modelling process and the resulting scene in Blender. The evaluation covered generation time, scene complexity, geometric structure, asset usage, memory estimation, and rendering configuration [16], [19], [23]. The main performance results are summarised in Table 3.

Table 3. Blender Scripting Activity Log

No.	Evaluation Aspect	Parameter	Result
1	Generation	Creation Time	0.9778 sec
2	Generation	Evaluation Time	0.051 ms
3	Scene Complexity	Total Objects	149
4	Scene Complexity	Mesh Objects	145
5	Scene Complexity	Cameras	1
6	Scene Complexity	Lights	3
7	Geometry	Vertices	21,464
8	Geometry	Edges	42,352
9	Geometry	Faces	21,168
10	Geometry	Triangles	42,368

No.	Evaluation Aspect	Parameter	Result
11	Assets	Unique Meshes	145
12	Assets	Materials	57
13	Memory Estimate	Mesh Geometry	1.215 MB
14	Rendering	Render Engine	Cycles
15	Rendering	Resolution	1920 × 1080
16	Environment	Blender Version	5.2.2 LTS

Based on Table 3, the model was generated in 0.9778 seconds, with an evaluation time of 0.051 milliseconds. These measurements provide an initial indication of the time required for the scripting process and its reported evaluation operation. However, the evaluation time should be interpreted according to the specific measurement implemented in the performance-monitoring script. The resulting scene contained 149 objects, comprising 145 mesh objects, one camera, and three lights. The geometric structure consisted of 21,464 vertices, 42,352 edges, 21,168 faces, and 42,368 triangles. These values describe the geometric complexity of the generated model and provide quantitative indicators that can be used to characterise the resulting scene. Regarding asset usage, the scene contained 145 unique meshes and 57 materials, with an estimated mesh geometry memory footprint of 1.215 MB. The scene was configured to use the Cycles rendering engine at a resolution of 1920 × 1080 pixels. These results document the model's asset composition and rendering configuration, although they do not independently establish rendering speed or overall system performance.

The final rendering of the modelled object is presented in Fig. 3. The rendering demonstrates the resulting 3D model after the scripting-based modelling activities, including object construction, transformation, modifier application, material assignment, lighting, and camera configuration [24], [25]. The output was rendered in a horizontal 16:9 format with a resolution of 1920 × 1080 pixels. The resulting image has a file size of 6.70 MB (7,030,326 bytes), with both horizontal and vertical resolution specified at 72 DPI and a 64-bit depth. These properties indicate that the scripting process successfully produced a high-resolution rendered output suitable for visual inspection and documentation of the resulting model.



Fig. 3. Rendered Result of the 3D Model

3.2. Discussion

The findings of this study show that Blender scripting can be explored as a practical mechanism for constructing simple 3D objects through a sequence of individual activities. This is consistent with [6], [9], which demonstrate Blender's ability to support 3D modelling through basic modelling operations, while [7] also shows that Python scripting can be integrated into Blender for more specialised modelling and visualisation activities. However, unlike these studies, the present study focuses on documenting and analysing individual scripting activities rather than architectural reconstruction, scientific visualisation, or the development of specialised modelling techniques. The modelling process generated 413 recorded activities, each associated with observable attributes and parameters. These records provide a basis for examining the sequence and characteristics of scripting operations involved in constructing a simple 3D object.

The activity-based results also show that the modelling process consists of repeated operations such as primitive object creation, transformation, modifier application, and scene configuration. This complements the findings of [8], which highlights the flexibility of Blender's scripting and modifier-based workflow, although the present study does not propose a parametric or Geometry Nodes-based method. Similarly, [10] examines Blender within a structured production workflow, whereas this study emphasises the documentation and quantitative analysis of scripting activities. Beyond the activity log, performance testing was conducted to characterise the resulting 3D scene. The model was generated in 0.9778 seconds and consisted of 149 objects, including 145 mesh objects. Its geometric structure comprised 21,464 vertices, 42,352 edges, 21,168 faces, and 42,368 triangles. The performance results also documented asset usage, estimated mesh geometry memory, and rendering configuration. These measurements complement the activity records by describing both the modeling operations and the structural characteristics of the generated scene.

The contribution of this study is exploratory and descriptive. By combining activity-based analysis with performance testing, it provides an account of how Blender scripting is used to construct a simple 3D object and characterises the resulting scene through measurable indicators. However, the reported performance measurements describe a single modelling process and its resulting scene; they do not establish comparative efficiency or demonstrate superior performance relative to other modelling approaches. The study therefore contributes an activity-level perspective on Blender scripting without introducing a new modelling method or claiming optimisation over existing approaches.

4. Conclusion

This study concludes that Blender scripting provides a practical means of constructing simple 3D objects while enabling the modelling process to be documented and analysed at the activity level. The modelling process was represented as a sequence of individual operations executed through Blender's Python scripting interface. These operations included clearing the initial scene, creating primitive cube objects, defining their locations and scales, applying transformations, and applying the Realistic Bevel modifier. The resulting activity log contained 413 recorded rows, of which 50 representative rows were presented in the study. These records demonstrate how scripting activities and their associated parameters can be documented to provide a traceable account of the modelling process. In addition to activity-based analysis, performance testing characterised the resulting 3D scene in terms of generation time, scene complexity, geometric structure, asset usage, estimated mesh geometry memory, and rendering configuration. The model was generated in 0.9778 seconds and consisted of 149 objects, including 145 mesh objects, with 21,464 vertices, 42,352 edges, 21,168 faces, and 42,368 triangles.

These findings show that activity logging and performance testing can complement each other in describing both the modelling operations and the characteristics of the resulting scene. Given the exploratory and descriptive nature of this study, the findings are not intended to establish a new modelling technique, optimise an existing method, or demonstrate superiority over manual modelling. Instead, they provide an activity-based account of Blender scripting and a quantitative description of the generated 3D model.

References

- [1] M. Tejera, S. Galiç, and Z. Lavicza, “3D Modelling and Printing in Teacher Education: A Systematic Literature Review,” *J. STEM Educ. Res.*, vol. 9, no. 1, pp. 1–32, Jan. 2026, doi: 10.1007/s41979-025-00147-2.
- [2] A. Hajirasouli and S. Banihashemi, “Augmented reality in architecture and construction education: state of the field and opportunities,” *Int. J. Educ. Technol. High. Educ.*, vol. 19, no. 1, p. 39, Dec. 2022, doi: 10.1186/s41239-022-00343-9.
- [3] Ö. Uzunel and A. Topaloğlu Uzunel, “From Social Media to Digital Memory: Opportunities and Limitations in the 3D Representation of Cultural Heritage,” *Akdeniz İnsani Bilim. Derg.*, vol. 16, no. 1, pp. 157–177, Jun. 2026, doi: 10.13114/mjh.1892073.
- [4] S. Muenster, “Digital 3D Technologies for Humanities Research and Education: An Overview,” *Appl. Sci.*, vol. 12, no. 5, p. 2426, Feb. 2022, doi: 10.3390/app12052426.
- [5] I. Lupiani, *Blender Scripting with Python*. Berkeley, CA: Apress, 2025. doi: 10.1007/979-8-8688-1127-2.
- [6] D. Petrov and G. Hristov, “Using Blender for 3D Modeling of Public Buildings,” *Sci. Eng. Educ.*, vol. 11, no. 1, pp. 11–14, May 2026, doi: 10.59957/see.v11.i1.2026.2.
- [7] V. Kudak, V. Perig, V. Dzhumelia, and O. Kryoka, “Modeling of Resident Space Object Light Curves with Blender Software,” *Artif. Satell.*, vol. 59, no. 2, pp. 42–54, Jun. 2024, doi: 10.2478/arsa-2024-0003.
- [8] E. Sommer, M. Koehl, and P. Grussenmeyer, “Parametric Modelling Techniques for Rhine Castle Models in Blender,” *Heritage*, vol. 8, no. 1, p. 31, Jan. 2025, doi: 10.3390/heritage8010031.
- [9] N. N. Sakbani, U. Hayati, and A. R. Dikananda, “PERANCANGAN DESAIN INTERIOR GEDUNG STUDIO TV MENGGUNAKAN SOFTWARE BLENDER,” *Narada J. Desain dan Seni*, vol. 9, no. 3, Dec. 2022, doi: 10.22441/narada.2022.v9.i3.005.
- [10] M. Maria and N. Hidayat, “Perancangan Layout 3D Asset Dalam Pembuatan Komik Strip Minecraft Menggunakan Blender,” *J. Appl. Multimed. Netw.*, vol. 8, no. 1, pp. 9–16, Aug. 2024, doi: 10.30871/jamn.v8i1.7203.
- [11] P. Acampora, *Python Scripting in Blender: Extend the power of Blender using Python to create objects, animations, and effective add-ons*. Packt Publishing Limited, 2024.
- [12] I. Lupiani, “Getting Started on Blender Scripting,” in *Blender Scripting with Python*, Berkeley, CA: Apress, 2025, pp. 1–32. doi: 10.1007/979-8-8688-1127-2_1.
- [13] M. D. Shah, “Introduction to Scripting in Blender3D: Computational Geometry Algorithms,” in *ACM SIGGRAPH 2024 Courses*, New York, NY, USA: ACM, Jul. 2024, pp. 1–83. doi: 10.1145/3664475.3664525.
- [14] O. Raheem, B. Omijeh, and C. Ibeachu, “An In-depth Analysis of Rendered Models using Blender : A Research Result,” *JITCE (Journal Inf. Technol. Comput. Eng.)*, vol. 8, no. 2, pp. 67–72, Dec. 2024, doi: 10.25077/jitce.8.2.67-72.2024.
- [15] M. A. Hamid, S. A. Rahman, I. A. Darmawan, M. Fatkhurrohman, and M. Nurtanto, “Performance efficiency of virtual laboratory based on Unity 3D and Blender during the Covid-19 pandemic,” *J. Phys. Conf. Ser.*, vol. 2111, no. 1, p. 012054, Nov. 2021, doi: 10.1088/1742-6596/2111/1/012054.
- [16] I. A. Astuti, I. H. Purwanto, T. Hidayat, D. A. Satria, Haryoko, and R. Purnama, “Comparison

- of Time, Size and Quality of 3D Object Rendering Using Render Engine Eevee and Cycles in Blender,” in *2022 5th International Conference of Computer and Informatics Engineering (IC2IE)*, IEEE, Sep. 2022, pp. 54–59. doi: 10.1109/IC2IE56416.2022.9970186.
- [17] Y. Ouyang *et al.*, “Integration of machine learning XGBoost and SHAP models for NBA game outcome prediction and quantitative analysis methodology,” *PLoS One*, vol. 19, no. 7, p. e0307478, Jul. 2024, doi: 10.1371/journal.pone.0307478.
- [18] G. F. Tua Sinaga, E. Prakasa, M. Y. Rezaldi, and A. Abdurrazzaq, “Vertex Reduction Optimization on Drone Photo-Based 3D Orthomosaic for Smart Parking Virtual Reality Content in the KST Samaun Samadikun Area,” in *2026 13th International Conference on ICT for Smart Society (ICISS)*, IEEE, Aug. 2026, pp. 1–6. doi: 10.1109/ICISS71466.2026.11715469.
- [19] M. Rosalina, G. P. D. Aji, and I. Sonjaya, “Design and Optimization of 3D Asset for Multimedia Representation of Satriamandala Museum,” in *2025 9th International Conference on Information Technology, Information Systems and Electrical Engineering (ICITISEE)*, IEEE, Dec. 2025, pp. 415–420. doi: 10.1109/ICITISEE68184.2025.11355044.
- [20] R. K. Luu and M. J. Buehler, “Bioinspired123D: generative 3D modeling system for bioinspired structures,” *AI Sci.*, vol. 2, no. 2, p. 025004, Jun. 2026, doi: 10.1088/3050-287X/ae61d1.
- [21] D. Apriliyanti, J. N. Fadila, and F. Nugroho, “3D ANIMATION DESIGN ‘SCIENCE, LANTERNS TO HEAVEN’ USING THE POSE-TO-POSE METHOD,” *JITK (Jurnal Ilmu Pengetah. dan Teknol. Komputer)*, vol. 8, no. 1, pp. 46–55, Jan. 2023, doi: 10.33480/jitk.v8i1.2231.
- [22] A. F. Leonard, G. Gjonaj, M. Rahman, and H. E. Durand, “Virtual Test Beds for Image-Based Control Simulations Using Blender,” *Processes*, vol. 12, no. 2, p. 279, Jan. 2024, doi: 10.3390/pr12020279.
- [23] I. Biškup, M. Grašić, and A. Bernik, “Procedural Versus Baked Textures in Path-Traced Rendering: A Controlled Comparison of Render Time and Memory Footprint in Blender Cycles,” *J. Imaging*, vol. 12, no. 10, p. 471, Sep. 2026, doi: 10.3390/jimaging12100471.
- [24] L. Soni, A. Kaur, and A. Sharma, “A Review on Different Versions and Interfaces of Blender Software,” in *2023 7th International Conference on Trends in Electronics and Informatics (ICOEI)*, IEEE, Apr. 2023, pp. 882–887. doi: 10.1109/ICOEI56765.2023.10125672.
- [25] Y. S. Praka *et al.*, “Implementation of a 3D Animation Production Pipeline for a Psychological Well-Being Trailer: Visualizing Self-Acceptance Using Autodesk Maya,” *Intechno J. (Information Technol. Journal)*, vol. 7, no. 2, pp. 65–76, Dec. 2025, doi: 10.24076/intechnojournal.2025v7i2.2485.