CSG-Based ML-Supported 3D Translation of Sketches into Game Assets for Game Designers

Yiming Chen[1][0009−0006−9016−6646], Yihang Liu[1][0009−0009−4147−1101], and Gizem Kayar-Ceylan[1][0000−0002−7811−9357]

New York University, New York, NY 10012, USA
https://www.nyu.edu/
yc4346@nyu.edu
yl7838@nyu.edu
gk2409@nyu.edu

Abstract. This research project aims to develop a new technique for translating 2D sketches into 3D assets like building models using machine learning and Constructive Solid Geometry (CSG) algorithms. The ability to quickly create 3D building models from simple freehand sketches has widespread applications in fields like architecture, construction, and interactive design. The key innovation is the use of CSG principles to understand geometric elements from sketches, extrude them into 3D, and combine simple shapes to create complex structures.
This allows the creation of building models with walls, windows, doors, and other features that closely match the original sketch, without requiring extensive 3D modeling expertise. Specialized sketch recognition methods including machine learning techniques for edge detection and point decimation are used to identify lines, arcs, and intersections, and transform them into the necessary CSG primitives. We plan to develop a user-friendly web interface where users can sketch freehand, and see a live 3D model preview generated in real-time by reconstructing surfaces from the identified geometric features. This integrated sketching and 3D modeling environment provides an intuitive workflow, harnessing CSG techniques to generate complex forms from sparse 2D input. Overall, this research aims to improve sketch-based 3D modeling methods and enable more users to create 3D content with ease. The researchers believe this work has the potential to significantly enhance productivity in construction planning, enable faster design iterations, and increase architectural creativity.

Keywords: 3D Translation of Sketches · Constructive Solid Geometry · Game Engine Integration.

1 Introduction

The ability to quickly and intuitively translate 2D sketches into 3D assets, e.g. building models, has widespread applications in fields such as architecture, construction, video game design, and interactive media. Architects, designers,

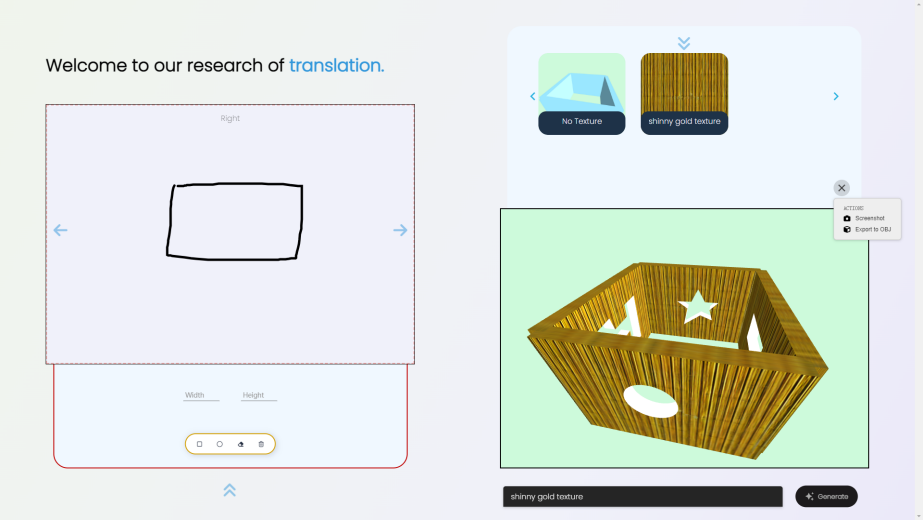

Fig. 1. 3D Reconstruction Pipeline

and artists often begin the creative process by hand-drawing rough conceptual sketches, which could be tremendously valuable if they could be instantly transformed into 3D representations. This could aid in the visualization and iteration of building designs, help translate abstract blueprints into physical visualizations, and empower those without extensive 3D modeling expertise to create virtual environments and assets.

However, the technical challenge of reliably reconstructing 3D geometry from simple 2D sketches remains an open problem. While humans can easily interpret the meaning and structure of basic sketch elements like walls, windows, and doors, computer vision and geometry processing algorithms continue to struggle with this task. Existing sketch-to-3D techniques often rely on standardized geometric drawings, large training datasets, or can only handle simple shapes, limiting their real-world applicability.

As shown in Figure 1, the user interface for our application includes a sketch area on the left side for users to input 2D sketches, coupled with a real-time 3D preview window that displays the transformation into 3D models. It features a tool palette for direct sketch manipulation like rectangle or circle drawers. Essential action buttons for screenshot-ting and exporting are present, along with navigation controls to explore the model with different selection of textures. This setup aims to facilitate intuitive 3D content creation from simple sketches, enhancing accessibility and user engagement.

It is worth noting that many of the commercially available Large Language Models (LLMs) have begun to support the generation of 3D images and videos. However, most of them only produce 2D images or videos with 3D concepts. There is no model that works well for generating 3D objects that can be applied

to games or other 3D designs. For sketches, there are models that can generate images from sketches; however, most of the models are not sensitive to the vertex position information in the sketches. As a result, the generated images are not good enough for the design needs.

This research aims to develop a new approach that can robustly translate freehand sketches of any style into 3D building models maintaining significant vertex or edge features. By leveraging specialized sketch recognition methods and a unified sketching-modeling environment, the proposed system seeks to enable intuitive 3D content creation from sparse 2D input. The key innovations include the use of Constructive Solid Geometry (CSG) to automatically extrude and combine basic shapes into complex structures, and the integration of this 3D reconstruction pipeline with a user-friendly web interface for real-time sketch-based modeling. Overall, this work has the potential to significantly enhance productivity, enable faster iteration, and increase accessibility in a variety of 3D design applications.

2 Related Work

Sketch Analysis. Most of the sketches contained in existing datasets of hand-drawn sketches are single-object[1, 2], while datasets with sketches covering multiple objects are mostly generated from photos [3, 4]. Training deep neural networks (DNNs) to recognize sketches with multiple objects is challenging due to the unique characteristics of sketches, such as their abstract nature and lack of color and texture information. Recent studies on object detection in computer vision have shown that DNNs based on bounding boxes, like those implemented in YOLO, can effectively distinguish multiple objects in photos [5, 6]. However, these DNNs designed for photos do not perform well on sketches [7]. Unlike photographs, sketches are sparse and abstract, which complicates object recognition [7–9].

Sketches can be represented in multiple ways: as static pixel images, dynamic stroke sequences, and geometric graphs [9]. The state-of-the-art model for sketch recognition is Sketch-a-Net, a DNN-based model that achieves a 74.90% accuracy on the TU-Berlin dataset [9]. Despite its success, there has been relatively little research on multi-object sketch analysis in recent years, likely due to the lack of new datasets. Most existing models can only study single-target sketches and output a single result without providing positional information.

For shape recognition from sketches, vertex-based polygon analysis combines both efficiency and accuracy and also returns positional information based on vertices. This approach addresses the limitations of current DNN-based models by offering a more detailed understanding of the sketch structure.

Constructive Solid Geometry. Constructive Solid Geometry (CSG) is a modeling technique used in computer graphics that builds complex surfaces or objects by using Boolean operations to combine simpler ones [10]. CSG is advantageous because of its high scalability, intuitive modeling process, and efficient data man-

agement of simple primitive models, resulting in fast rendering and easy retrieval of complex shapes. CSG models can achieve high accuracy and are highly scalable, accurately representing complex geometries with simple primitives such as cubes, spheres, and cylinders. The intuitive process of creating models with CSG - using operations such as union, intersection, and difference - makes it accessible to users without extensive 3D modeling expertise. Moreover, CSG's use of simple primitive models allows for efficient data management and quick rendering, particularly useful in applications requiring real-time performance, such as video game engines [10].

In video game development, CSG is integrated into game engines to generate complex 3D environments and assets, enhancing visual quality and interactivity. Recent research has focused on improving CSG techniques and integrating them with modern machine-learning approaches. For example, combining CSG with machine learning techniques for edge detection and point decimation improves the identification and transformation of geometric elements from sketches to 3D models [11]. Implementing CSG in real-time systems, such as web-based interfaces, enables users to create and visualize 3D models from sketches instantly, leveraging the efficiency of CSG operations to maintain performance while providing high-quality visual output [12].

Texture Generation. Building on the foundations laid by previous research in generative models, the "Denoising Diffusion Probabilistic Models" by Ho et al [13]. presents a comprehensive framework for image synthesis that draws from both diffusion probabilistic models and denoising score matching techniques. This approach, rooted in the principles of nonequilibrium thermodynamics, innovatively utilizes variational inference to reverse a diffusion process characterized by incremental Gaussian noise applications, thereby generating high-fidelity images.

Significant progress in high-resolution image synthesis has been achieved through various approaches. Generative Adversarial Networks (GANs), introduced by Goodfellow et al., have been pivotal in generating high-quality images, though they often encounter stability challenges. Variational Autoencoders (VAEs) and flow-based models, as developed by Kingma and Welling, provide robust frameworks but sometimes at the expense of image quality. Recently, diffusion models, particularly those advanced by Ho et al. and Dhariwal and Nichol [14], have shown promise in balancing image quality with training stability by utilizing reweighted variational objectives. These methods underpin our research, where we extend the capability of Latent Diffusion Models (LDMs) to efficiently produce detailed high-resolution images in a reduced computational space, advancing the state-of-the-art in the field.

2D-To-3D Generation Based on Sketches. Recent advancements in deep learning and computer vision have led to significant progress in 2D-to-3D generation based on sketches. One of the primary approaches in this domain is image-to-image translation models, such as pix2pix3D, which focus on maintaining 3D

realism when viewing objects from different angles [15]. These models typically utilize Generative Adversarial Networks (GANs) to generate high-quality 3D representations from 2D inputs [15].

Another direction is to generate 3D models in the form of point clouds like the work by Jiayun Wang et al [16]. This method exploits the inherent spatial information in point clouds to generate more accurate 3D shapes [16]. Despite these advances, a significant challenge remains the lack of comprehensive training datasets, which limits the ability of these models to generalize to different sketch styles and complexities. In addition, many existing models are limited to handling simple shapes and often struggle with more complex, multi-object sketches.

Recent research has sought to address these limitations by integrating multi-view learning and shape-from-shading techniques. For instance, multi-view learning models synthesize multiple 2D views of a sketch to infer the 3D structure, improving the robustness and accuracy of the generated models. Shape-from-shading techniques, on the other hand, utilize shading information in sketches to better understand the depth and curvature of objects, leading to more realistic 3D reconstructions [17]. Furthermore, hybrid models combining CSG principles with deep learning have shown promise in generating more precise and geometrically accurate 3D models from sketches [11]. These advancements are crucial for developing robust 2D-to-3D generation systems that can handle a wide range of sketch inputs and produce high-quality 3D models suitable for various applications. Overall, not limited to sketches as input, 2D-to-3D is still a problem without well-established generative model solutions.

3 Method

3.1 Sketch Recognition and Geometric Analysis

The foundation of the sketch-to-3D reconstruction pipeline is the robust recognition and analysis of the input sketch geometry. To achieve this, we employed a multi-step computer vision approach leveraging both traditional image processing techniques and machine learning-based shape detection.

The process begins by loading the input sketch image and applying several preprocessing steps to enhance the sketch features and prepare the data for analysis. First, a Gaussian blur filter is used to smooth the image and reduce noise, helping to clean up the hand-drawn sketch lines. The Gaussian blur filter works by averaging the pixel values in a neighborhood around each pixel in the image. The averaging process is weighted by a Gaussian function, which is a bell-shaped curve that decreases in value as you move away from the center pixel. This results in pixels closer to the center of the neighborhood having a greater influence on the average than pixels farther away, creating a smoothing effect that preserves edges while reducing high-frequency noise. By applying the Gaussian blur, fine details and minor imperfections in the sketch are smoothed out, making the primary structures more prominent and easier to detect. This

preprocessing step is crucial for subsequent stages, as it enhances the clarity of the sketch, allowing for more accurate feature extraction and geometric analysis. The degree of blurring can be controlled by adjusting the standard deviation of the Gaussian function, which determines the width of the bell curve. A larger standard deviation results in more extensive blurring, while a smaller standard deviation focuses on reducing only minor noise without significantly altering the overall structure of the sketch.

Next, Canny edge detection is performed to identify the key edges and contours in the sketch. The Canny edge detection algorithm is a multi-stage process that involves several steps. These include noise reduction using a Gaussian filter, gradient calculation, non-maximum suppression, and edge tracking by hysteresis.

1. Gaussian filter: It reduces noise in a manner analogous to the initial Gaussian blur step. This prepares the image for edge detection by smoothing out minor variations that could otherwise lead to false edge detection.
2. Image intensity gradient: It is calculated using finite difference approximations. This step involves computing the gradient of the image intensity at each pixel, identifying areas of rapid intensity change, which correspond to edges. The gradients are calculated in both horizontal and vertical directions to determine the strength and direction of the edges.
3. Non-maximum suppression: It is then applied to the edges to refine the edge map. This technique involves examining the gradient magnitude of each pixel and retaining only the local maxima, where the gradient magnitude is the highest. By doing this, non-maximum suppression helps in eliminating extraneous details and reducing the thickness of the edges, resulting in a cleaner, more precise edge representation.
4. Edge tracking by hysteresis: It employs two thresholds to identify strong and weak edges. Strong edges are those with gradient magnitudes above the higher threshold and are considered definitive edges. Weak edges, which have gradient magnitudes between the two thresholds, are included in the edge map only if they are connected to strong edges. This connectivity check helps in preserving the continuity of edges and discarding isolated weak edges that are likely to be noise.

Therefore, Canny edge detection algorithm produces a binary image that highlights the significant edges and contours in the sketch, which are crucial for further processing and 3D reconstruction. This method ensures that the detected edges are both accurate and meaningful, providing a robust foundation for subsequent modeling tasks.

A morphological closing operation is applied to refine the edge information further to close any gaps in the detected edges. With the preprocessed sketch data, the next step is to detect and classify the various geometric shapes present in the drawing. This is accomplished by iterating through the detected contours and approximating each contour with a polygon using the `cv2.approxPolyDP()` function. This function takes a contour as input and approximates it with a polygon. The function works by iteratively reducing the number of vertices in the polygon, while ensuring that the maximum distance between the original

contour and the approximated polygon is less than a specified epsilon value. This allows the function to find the minimal number of vertices required to accurately represent the original contour shape. Specifically, the algorithm checks the following criteria to identify different shape types:

- Triangles: Identified by 3 vertices forming a convex polygon
- Rectangles: Identified by 4 vertices forming a convex polygon with specific edge angle properties
- Circles: Identified by closed contours that satisfy circularity criteria based on the bounding box and area

For each detected shape, the vertex coordinates are extracted and stored, along with metadata like the bounding box dimensions. This information provides the essential geometric data needed for the subsequent 3D reconstruction stage. Additionally, the algorithm attempts to identify custom, irregularly shaped objects that don't fit the predefined shape categories. For these cases, the vertex coordinates of the approximated polygon are still captured and stored, enabling more complex geometry to be incorporated into the final 3D model.

An important aspect of the sketch recognition process is the classification of doors and windows. Any detected shapes with their bottom edge below 5% of the image height are classified as potential doors and labeled accordingly in the output. This contextual information is critical for properly interpreting the architectural intent behind the sketch.

The extracted geometric information from the sketch analysis is written to a specified output text file. The format of the output is designed to be structured and easily parsable for the subsequent 3D reconstruction steps. The first line of the output contains the width and height of the canvas, which provides the necessary context for interpreting the coordinate values. The subsequent lines are grouped by shape type, starting with a single-character identifier (C for circles, T for triangles, R for rectangles, U for custom/irregular shapes). For each detected shape, the relevant vertex coordinates and other metadata (e.g., radius for circles, bounding box dimensions for rectangles) are listed on a new line. By processing the input sketches one by one and appending the output to the same text file, the system is able to generate a comprehensive representation of the entire building's geometry, with each wall's elements clearly labeled and organized. This structured output format ensures that the downstream 3D reconstruction module can reliably interpret the extracted sketch data and generate an accurate 3D model.

Algorithm 1: Sketch Reading and Shape Detection

```
Data: Input sketch image
Result: Textual data of detected shape
Load the image and preprocess;
image ← load_image();
blurred_image ← apply_gaussian_blur(image);
gray_image ← convert_to_grayscale(blurred_image);
edges ← apply_canny_edge_detection(gray_image);
closed_edges ← apply_morphological_closing(edges);
contours ← find_contours(closed_edges);
Initialize lists for shape info;
circle_info, triangle_info, rectangle_info, custom_object_info
 ← [], [], [], [];
for each contour in contours do
    Get the polygonized contour;
    approx ← approximate_contour(contour);
    Classify the shape;
    if is_triangle(approx) then
        store_triangle_info(triangle_info, approx);
    else
        if is_rectangle(approx) then
            store_rectangle_info(rectangle_info, contour);
        else
            if is_circle(contour) then
                store_circle_info(circle_info, contour);
            else
                store_custom_object_info(custom_object_info,
                 approx);
Write output and save image;
write_output_text_file(width, height, circle_info, triangle_info,
 rectangle_info, custom_object_info);
save_output_image(image);
```

We may notice that Algorithm 1 processes an input sketch image to detect and classify various shapes, producing textual data as output. It begins by loading and preprocessing the image, including applying Gaussian blur to reduce noise, converting it to grayscale, and performing Canny edge detection to identify edges. Morphological closing is then applied to close gaps in the edges, followed by contour detection.

For each detected contour, the algorithm approximates the contour to a polygon and classifies it as a triangle, rectangle, circle, or custom object based on its geometric properties. The classified shape information is stored in corresponding lists. Finally, the algorithm writes the detected shape information to a text file and saves the processed image, completing the shape detection process.

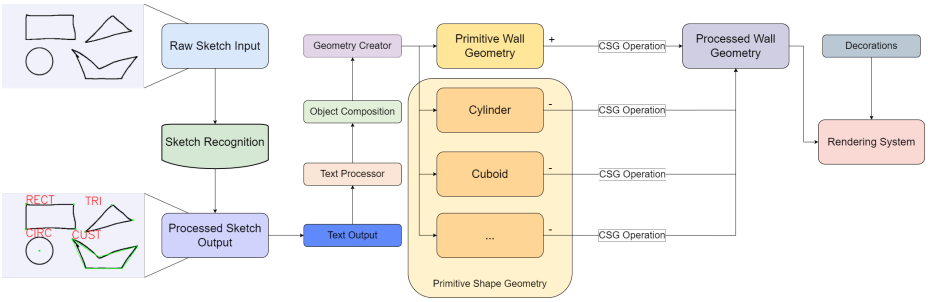

Fig. 2. 3D Reconstruction Pipeline

3.2 3D Reconstruction with CSG

The key to converting the 2D sketches into 3D models is the use of Constructive Solid Geometry (CSG) operations as shown in Figure 2. To begin the 3D reconstruction from the text output from the sketch analysis, we have a text processor. The text processor parses the text, recognizes the shape type of each object, and stores each object's data in the object composition as an array. The object composition is an array that contains a nested composite data type for each wall. Each nested composite data type contains arrays of wall sizes and object data. This design allows quick access to the data for each wall and each object.

The geometry creator is a function that turns walls and objects into geometry objects. It first creates a wall and then the shape objects on this wall. The geometry for walls derives from the geometric transformations of a cube. These are the primitive wall geometry objects. For each shape object on the walls, the geometry creator uses the shape data, such as vertices, to draft a planar shape and then make a column by panning. We call these geometry objects from the shape objects Primitive Shape Geometry (PSG) objects. PSG objects are simple, which is favorable for the following CSG operation. Also, the simplicity of PSG helps achieve the quick construction necessary for real-time reconstruction.

For each PSG object, a CSG operation is conducted to subtract the PSG object from the corresponding primitive wall geometry object. After finishing all CSG operations, we have the processed wall geometry objects with voids corresponding to the shapes in the sketches. Through the data stored by the object composition, further windows or other decoration objects can be generated and settled on the matching voids. For instance, we can use a series of CSG operations with primitive 3D geometry objects to create decoration objects.

The processed wall geometry objects are then transferred to the rendering system with decorations, including the generated textures and decoration objects. All geometry objects are turned into mesh objects in the rendering system. According to the workflow we have presented, most of the 3D object operations are conducted on geometry objects. It is also an important feature aiding the efficiency needed for real-time reconstruction.

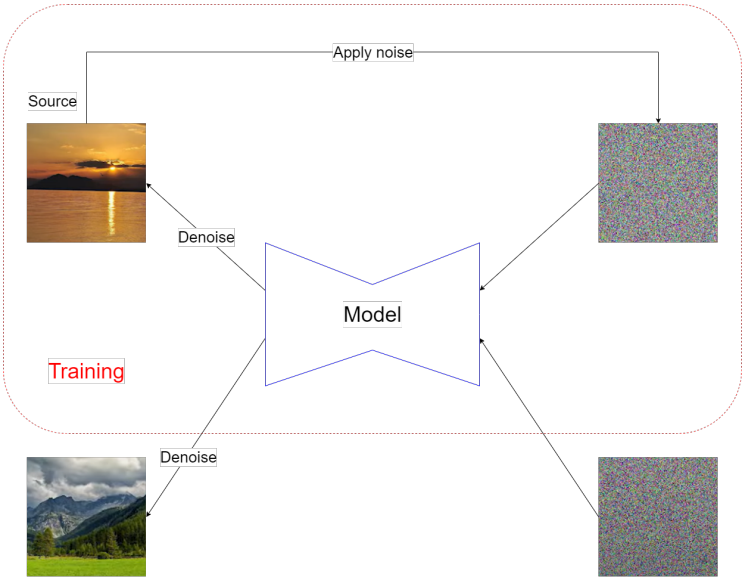

Fig. 3. Training and Application of a Denoising Diffusion Probabilistic Model: A Visual Representation of Noise Addition and Reduction in Image Processing

3.3 Diffusion-Based Texture Generation for 3D Models

We have integrated a diffusion-based texture generation approach to automatically create textures for the 3D building models reconstructed from the sketch input. This technique leverages the impressive capabilities of diffusion models, a class of deep learning models that have shown remarkable success in generating high-quality, diverse images from textual descriptions. As shown in Figure 3, the core of the texture generation component is the Stable Diffusion pipeline, which is a pre-trained denoising diffusion probabilistic model made available by Hugging Face [14].

Diffusion models work by learning to gradually transform random noise into realistic images that match a given textual prompt. This is achieved through a process of iterative refinement, where the model learns to predict the step-by-step changes required to transform the initial noise into the desired output. In the context of this sketch-to-3D reconstruction research, the texture generation process is invoked whenever a 3D building model needs to be textured. The user provides a textual prompt that describes the desired style or theme for the texture, such as "modern office texture" or "medieval wooden texture". This prompt is then fed into the Stable Diffusion pipeline, which generates a high-quality, photorealistic image that matches the given description. The architecture of the Stable Diffusion model can be broadly divided into two main components:

– Encoder: The encoder network takes the input text prompt and encodes it into a compact, semantic representation. This representation captures the

high-level features and attributes of the desired texture, such as the materials, colors, and architectural elements.
– Denoiser: The denoiser network is responsible for progressively refining a random noise input into the final, high-quality texture image. It does this by learning to predict the step-by-step transformations required to gradually remove the noise and shape the image to match the encoded text prompt.

The diffusion process works by iteratively applying a series of learned noise-adding and noise-removing operations to the input. In the initial stages, the input is heavily corrupted with noise, and the model learns to predict how to gradually remove this noise to reveal the underlying image structure. As the process continues, the model becomes increasingly capable of generating sharper, more detailed textures that closely align with the given prompt.

One of the key advantages of using a diffusion-based approach for texture generation is its flexibility and generalization capability. Since the model is trained on a large and diverse dataset of images and text pairs, it can generate textures for a wide range of architectural styles and materials, beyond what might be feasible with traditional texture synthesis or manual creation methods. In the context of the sketch-to-3D reconstruction pipeline, the generated textures are seamlessly integrated with the 3D building models, providing a visually appealing and realistic final result. This diffusion-based texture generation component significantly enhances the overall quality and fidelity of the 3D models, making them more suitable for a variety of applications, such as architectural visualization, interactive design, and video game asset creation.

3.4 Web-Deployed Sketch-Based 3D Modeling System

Web Graphics Library (WebGL) and Three.js We used WebGL and the Three.js library to handle the 3D rendering and visualization components of the web-deployed sketch-to-3D modeling system. This approach leverages the power and flexibility of WebGL, a low-level graphics API that allows for hardware-accelerated 3D rendering directly within web browsers, and the comprehensive Three.js library, which provides a high-level, scene graph-based API for building complex 3D scenes and animations.

The core of the 3D rendering system is the Three.js scene, which is initialized with a perspective camera, a WebGL renderer, and various lighting components. The scene is then populated with the 3D building models that are reconstructed from the user's sketch input. The system first reads a text file that contains the geometric information extracted from the user's sketch, including the locations and dimensions of various shapes (circles, rectangles, triangles, and custom polygons) that make up the building's walls. Using the information read from the text file, the system constructs the corresponding 3D geometries for each shape and extruded polygons for triangles and custom shapes. Then we employs a CSG library (Three-BVH-CSG) to perform boolean operations on the 3D geometries, effectively "subtracting" the shapes that represent doors, windows, and other architectural features from the base wall geometry. This allows the creation of

complex building models with accurate openings and structural elements. We position and orient each wall object model in the 3D scene based on the information provided in the text file, ensuring that the complete building structure is accurately represented. Then the system applies a default material (a Phong material with a specified wall color) to the building models. Additionally, the system integrates the ability to load and apply custom textures to the walls, which are generated using a diffusion-based approach and made available through a separate texture management system.

The resulting 3D scene is then rendered using the Three.js WebGL renderer, which takes advantage of the GPU hardware acceleration provided by the user's web browser. The scene can be interactively explored and manipulated using the built-in OrbitControls component of Three.js, allowing users to pan, zoom, and rotate the 3D building model. To enable a seamless user experience, the system also includes various user interface (UI) elements and interaction mechanisms. This includes a button for triggering the texture generation process, a loading progress indicator, and a drawer interface that displays a history of the generated textures, allowing users to quickly switch between different material styles.

Flask We implemented the sketch-to-3D modeling system as a web-based application using the Flask framework. This approach ensures cross-platform accessibility via a web browser. The system processes user sketches, extracts geometric information, and generates high-quality textures for 3D models.

Our web application includes error handling mechanisms to handle exceptions that may occur during the sketch processing and texture generation steps. Appropriate error messages are returned to the client, providing feedback on the success or failure of the requested operations. The web application also manages the necessary directory structure and file operations for handling the input sketches, output models, and generated textures. Additionally, the web application is designed with unit testing in mind. It supports the use of environment variables to enable unit testing mode, which allows for the specific configuration of input and output paths, as well as the generation of test images for validation purposes.

4 Results

Figure 4 showcases the results of our 2D sketch-to-3D model translation system, demonstrating the transformation of various hand-drawn sketches into fully realized 3D models, complete with texturing and export to a Unity environment.

1. Sketch Input: The leftmost column displays the initial 2D sketches used as input. These sketches vary in complexity and include basic geometric shapes, as well as more intricate designs.
2. Basic 3D Model: The second column illustrates the initial 3D models generated from the 2D sketches using our constructive solid geometry (CSG) based system. These models accurately reflect the structure and layout of the sketches.

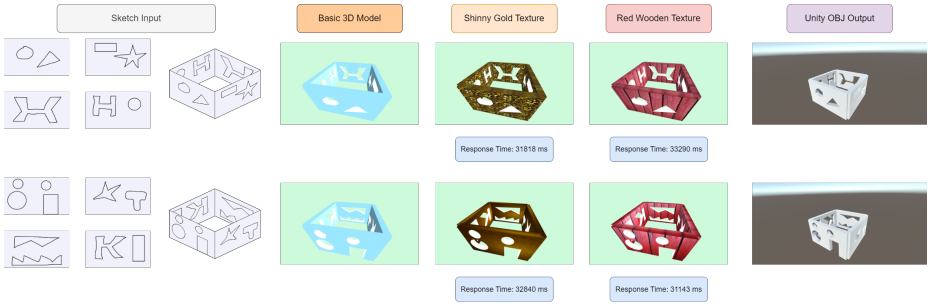

Fig. 4. Results of 2D Sketch-to-3D Model Translation with Texturing and Unity Output

3. Texturing: The third and fourth columns show the application of different textures to the 3D models. Two types of textures are applied: shiny gold and red wooden textures. These textured models demonstrate the versatility of our system in applying various surface materials to the generated models, enhancing their visual appeal and realism.
4. Response Time: Below each textured model, the response time for generating and texturing the model is displayed. This metric provides insight into the efficiency of our system, with times ranging around 31,000 to 33,000 milliseconds.
5. Unity OBJ Output: The rightmost column presents the final 3D models as rendered in the Unity game engine. These models have been exported in the OBJ format, showcasing the system's capability to integrate seamlessly with popular game development platforms. The Unity-rendered models retain the structural integrity and details of the original sketches, demonstrating the system's effectiveness in maintaining accuracy through the transformation process.

5 Future Work

Further goals of our project are currently to complete the generation of wall decorations and to finish the analysis of sketches of indoor objects with the generation of objects. At present, there are no lightweight 3D generative models suitable for web-based real-time rendering. A suggested way to use 3D generative models is to prepare considerable 3D objects by generative models and store them in the database with labels. Another machine learning model can relate user's input, including sketched shapes and themes, and stored 3D objects. However, customized window/door shapes on walls can be problematic. Another way is to use CSG to generate windows and doors. It's possible to design a special set of 3D geometric combinations with CSG operations to deal with customized shapes.

We have noted that Blender is now trending in 3D modeling. Many 3D generative models use Blender as the output format. Generating a Blender model

will be challenging for a web-based project targeting real-time generation. But we can consider a method to output 3D objects in the Blender format.

6 Conclusion

In this research, we have developed an innovative method for translating 2D sketches into 3D assets using machine learning and Constructive Solid Geometry (CSG) techniques. It is able to output 3D objects that can be utilized in game engines including Unity. Our approach enables users without extensive 3D modeling experience to intuitively create complex structures by addressing the long-standing challenge of translating freehand sketches into 3D models.

We have constructed efficient sketch recognition methods through vertex and edge analysis, which supports customized shapes and maintains precise vertex and position information. Using CSG operations to construct complex 3D geometries from simple primitives also helps maintain the fidelity of the original sketches, and it achieves the computational efficiency that allows real-time rendering. We also incorporated diffusion-based models for generating realistic textures, enhancing the visual quality of the 3D models. The final results are combined in a web-based interface with real-time rendering capabilities for immediate feedback and iterative design.

Our research is seminal to the problem of translating 2D sketches into 3D objects, and, moreover, the further 2D to 3D translation. It has strong extensibility and potential. By continuing to refine and extend it, we can further enhance its utility and impact in various 3D design applications. Also, the implications of this research are far-reaching. It offers significant benefits in fields such as architecture, construction, video game design, and interactive media. By streamlining the process of creating 3D models from sketches, our system enables a wider range of users to engage in 3D design, potentially increasing productivity and fostering creativity.

7 Acknowledgment

The research receives Dean's Undergraduate Research Fund (DURF) from the College of Arts and Science (CAS) at New York University (NYU).

References

1. Google.: Quick, Draw!. The Data. https://quickdraw.withgoogle.com/data last accessed 2024/4/20.
2. Eitz, M., Hays, J., Alexa M.: How do humans sketch objects?. ACM Trans. Graph. 31, 4 (2012). https://doi.org/10.1145/2185520.2185540
3. Sangkloy, P., Burnell, N., Ham, C., Hays, J.: The sketchy database: learning to retrieve badly drawn bunnies. ACM Trans. Graph. 35, 4, Article 119 (2016). https://doi.org/10.1145/2897824.2925954

4. Li, M. Lin, Z., Mech, R., Yumer, E., Ramanan, D.: Photo-Sketching: Inferring Contour Drawings from Images. In: WACV (2019). https://arxiv.org/abs/1901.00542
5. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. Proceedings of the Advances in Neural Information Processing Systems (2015)
6. Luo, Y., Yu, T., Zheng, J., Ding, Y.: Design of engineering drawing recognition system based on Yolo V4. 2022 IEEE 6th Information Technology and Mechatronics Engineering Conference 2022, pp. 1221-1225. doi: 10.1109/ITOEC53115.2022.9734453.
7. Yu, Q., Yang, Y., Liu, F., et al. Sketch-a-Net: A Deep Neural Network that Beats Humans. Int J Comput Vis 122, 411–425 (2017). https://doi.org/10.1007/s11263-016-0932-3
8. Muhammad, U.R., Yang, Y., Song, Y., Xiang, T., Hospedales, T.M.: Learning Deep Sketch Abstraction. Computer Vision and Pattern Recognition (CVPR) (2018).
9. Xu, P., Hospedales, T. M., Yin, Q., Song, Y.-Z., Xiang, T., Wang, L.: Deep Learning for Free-Hand Sketch: A Survey. IEEE TPAMI (2022).
10. Fayolle, P.A., Friedrich, M.: A survey of methods for converting unstructured data to CSG Models. Computer-Aided Design (2024), 168, p. 103655. doi:10.1016/j.cad.2023.103655.
11. Wang, J., Lin, J., Yu, Q., Liu, R., Chen, Y., Yu, S.X.: 3D Shape Reconstruction from Free-Hand Sketches. In: Karlinsky, L., Michaeli, T., Nishino, K. (eds) Computer Vision – ECCV 2022 Workshops. ECCV 2022. Lecture Notes in Computer Science, vol 13808. Springer, Cham. doi:10.1007/978-3-031-25085-9_11.
12. Deng, K., Yang, G., Ramanan, D., Zhu, J.-Y.:. 3D-aware Conditional Image Synthesis. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2023).
13. Ho, J., Jain, A., and Abbeel, P.: Denoising Diffusion Probabilistic Models. arXiv:2006.11239 [cs.LG] (2020).
14. Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B.: High-Resolution Image Synthesis With Latent Diffusion Models. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 10684–10695. June 2022.
15. Deng, K., Yang, G., Ramanan, D., Zhu, J.Y.: 3D-aware Conditional Image Synthesis. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2023).
16. Wang, J., Lin, J., Yu, Q., Liu, R., Chen, Y., Yu, S.X.: 3D Shape Reconstruction from Free-Hand Sketches. In: Karlinsky, L., Michaeli, T., Nishino, K. (eds) Computer Vision – ECCV 2022 Workshops. ECCV 2022. Lecture Notes in Computer Science, vol 13808. Springer, Cham. https://doi.org/10.1007/978-3-031-25085-9_11
17. Barron, J. T., and Malik, J.: Shape, Illumination, and Reflectance from Shading. IEEE Transactions on Pattern Analysis and Machine Intelligence (2015), 37(8), 1670-1687.