Edge Detection Algorithm Used in a 3D Sketching & Photogrammetry System

Daniel He

Abstract

A clear 3D representation is vital to prototyping in design. 3D modeling is a popular approach to designing structures in 3D, resulting in accurate models that can be viewed from multiple perspectives. However, 3D modeling is too costly and rigid for designers because it has a steep learning curve. Existing heavy-weight 3D software packages are usually not suitable for ideation. Therefore, we consider using sketches to convey ideas in 3D a more natural way for prototyping. Even if 2D sketches are loose, people can still identify 3D structures from them. However, most sketching tools only support drawing in 2D, motivating us to find a solution to bridging 2D sketches and 3D structures.

To fill the gap, we propose a 3D sketching system that facilitates sketch creation in 3D. Users can create a sketch in 3D and view it from multiple perspectives, without having to create repetitive sketches. This application also supports reconstructing a point cloud from images taken from existing structures. We derived an innovative edge detection algorithm derived from Hough transform to extract borders as strokes from a point cloud reconstructed based on real-world images. This includes fitting multiple planes to the point cloud and creating strokes by projecting points to the planes.

We also present real-world use cases to evaluate our system. Our application allows designers to prototype more rapidly, opening a gate to more user-friendly ideation processes.

Keywords: 3D sketching, photogrammetry, design

Introduction

Despite the accuracy of 3D modeling (or computer-aided design, CAD), designers continue to use traditional sketching in the ideation process [1]. While CAD tools are convenient in production processes, the quantitative requirements and rigid manipulation of models divert attention from design. Actions including creating primitive shapes, extrusion, and combining shapes in popular 3D modeling software (for example, Maya, 3dsMax, Rhino, etc.) are much less flexible compared with sketches using pencil and paper or their virtual counterparts.

On the other hand, 2D sketches have their flaws: when expressing a 3D structure in the ideation process, the designer often draws several sketches from different angles of perspective to accurately describe the 3D structure of the prototype. Moreover, when designing the prototype based on existing 3D scenes, drawing previous structures costs extra time.

To solve these problems, we propose a 3D sketching application. This approach uses strokes as the main design primitive and allows users to draw on different planes. The system also uses an edge detection algorithm derived from Hough transform to extract borders as strokes from a point cloud reconstructed based on real-world images.

This paper is organized as follows: First, we review related applications and research projects in the field of sketching. Then, we focus on the stroke extraction algorithm and show some prototyping cases using our system. Finally, we discuss the limitation and future work of this application.

Related work

Currently, most sketching software of daily and industry use focuses on smooth user experience, providing functions such as brushes, layers, masks, and more. However, none of them supports sketching in 3D. However, many researchers proposed different approaches and methods for composing and rendering sketches in 3D space.

Tolba et al. [2] mainly focus on sketching with projective 2D strokes. They derived a perspective drawing system that projects hand-drawn 2D sketches onto specific planes and surfaces, also leaving out the need for actually 3D modelling. By first drawing on a 2D surface and projecting the strokes onto a unit sphere simulating the overall space in the real world, they provide a "visual experience of being immersed in the design space". Their system also incorporates an automatic algorithm to calculate the shadows of these strokes under an infinite light source and a walkthrough simulation in the 3D scene. Without having to create a 3D model, their system supports the changing of viewpoint. In our approach, we directly draw in 3D based on planes.

Dorsey et al. [3] also present a prototyping tool based on surfaces. They pushed the free motion of viewpoint to a new height compared to the previous researches. As a system based on drawing 2D strokes on different 3D surfaces, their program supports 3D viewing easily. They also allow users to load an image onto their 3D planes as drawing references, a trial of quickening designing processes based on real-world data. Our approach pushes this aspect even further: we directly reconstruct point clouds based on real-world images to support our sketches.

Paczkowski et al. [4] propose an architectural design system that integrates real-world context is also proposed, aiming to solve relatively large-scaled designing and sketching. The system described models the site topography using local measurements and GPS coordinates and align site features with multiple image pop-ups. A sketching system is then incorporated and users can draw on top of the images imported. However, this system is aiming for large-scale designs, and they cannot be easily used in small prototyping processes.

Methodology

In this part, we will focus on the algorithm used to extract strokes from point clouds. Prerequisite data are pictures from different angles of an existing 3D structure.

The first step is to reconstruct a 3D point cloud based on these images. The features in the images are extracted and matched, and a 3D point cloud is created using *COLMAP*, an open-source library that supports image-based 3D reconstruction [5]. The library first extracts and matches features from images taken from multiple angles, and reconstructs a sparse point cloud based on matched features. One of our use cases gives a concrete example of the procedure of photogrammetry using *COLMAP*.

The second step is to fit the planes from the point cloud. Here, we use a variation of the Hough transform algorithm [6]. In the 2D Hough algorithm, lines are expressed by the perpendicular distance from the line to the origin d and the angle of this perpendicular with the horizontal θ:

$$d = x\cos\theta + y\sin\theta$$

For each point (x, y) in the 2D space, we enumerate the value of θ and calculate the value of d. Then, the value at point (d, θ) in the voting space is incremented. The final result is determined by finding maximum values at certain (d, θ) points. The lines corresponding to those pairs of (d, θ) are the lines being fit.

Here, we use a variation of the Hough transformation in 3D space. We choose to express a plane using three parameters: φ, the angle between the normal vector and the x-axis; θ, the angle between the normal and the z-axis; and d, the distance from the origin to the plane, as shown in Figure 1.

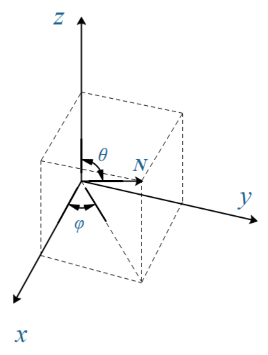

Figure 1. A normal vector $\boldsymbol{N}$ *and corresponding angles* θ *and* φ.

Hence, a normalized normal vector of the plane can be expressed as

$$\boldsymbol{N}(\sin\theta\cos\varphi\,,\sin\theta\sin\varphi\,,\cos\theta)$$

To simplify our representation, the coordinates of $\boldsymbol{N}$ will be represented using (x_n, y_n, z_n). As shown in Figure 2, for each point $\boldsymbol{P}(x, y, z)$ in our point cloud, we could derive the function of the plane according to the normal vector $\boldsymbol{N}$:

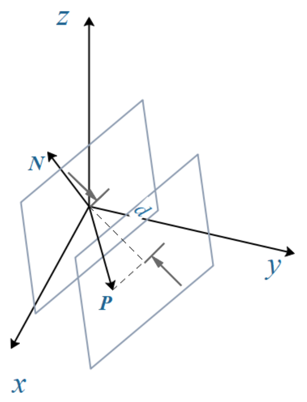

Figure 2. Variables used in determining the equation of the plane.

$$(\boldsymbol{N}\cdot d+\boldsymbol{P})\cdot\boldsymbol{N}=0$$

$$(dx_n+x, dy_n+y, dz_n+z)\cdot(x_n, y_n, z_n)=0$$

$$d(x_n^2+y_n^2+z_n^2)=-(x\cdot x_n+y\cdot y_n+z\cdot z_n)$$

The algorithm used in 2D line detection is called *Hough Transform*. The algorithm searches for lines on a surface through a voting process. Lines are parameterized as $d = x\cos\theta + y\sin\theta$, where d is the perpendicular distance from the line to the origin, and θ the the angle of this perpendicular with the horizontal axis (Figure 3).

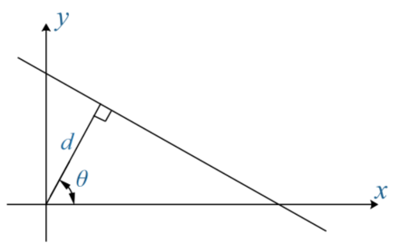

Figure 3. Parametrizing a line in the Hough Transform *Algorithm.*

For each point, the values of θ are being enumerated. A corresponding d is calculated based on the current θ and the coordinates of the point (x_i, y_i), and the vote for the pair (d, θ) is incremented. The final extracted lines are chosen by selecting the (d, θ) pairs that receive more votes than the threshold predefined.

In our algorithm, the first step is to enumerate (θ, φ) for every point in the point cloud and calculate the corresponding d according to our parametrized plane equation above. Then we could select a threshold and find points in the space (d, θ, φ) that achieved the desired amount of votes.

After determining the parameters for the planes, we could derive a transformation matrix for each plane that transforms the plane to the x-y plane. The transformation can be achieved by translating the plane by $-d$ towards the origin, rotate φ around the z-axis, and rotate θ around the y-axis, as shown in Figure 4. Determining the matrix of transformation can be found in [7].

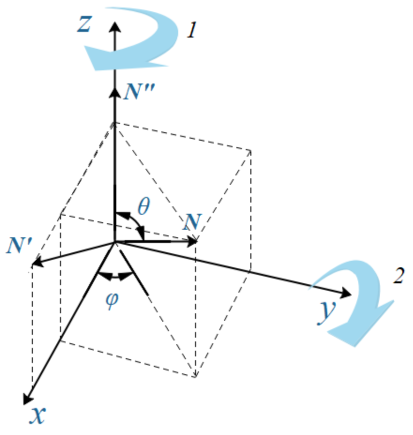

Figure 4. The process of rotating $\boldsymbol{N}$ *to* $\boldsymbol{N}'$*, and to* $\boldsymbol{N}''(0,0,1)$.

This matrix can also be calculated by determining the quaternion of rotation. The matrix rotates the normal $\boldsymbol{N}$ to the vector $(0,0,1)$.

$$q.xyz = \boldsymbol{N} \times (0,0,1)$$

$$q.w = \sqrt{|\boldsymbol{N}|^2 + |(0,0,1)|^2} + \boldsymbol{N} \cdot (0,0,1)$$

The final rotation quaternion can be calculated by normalizing q [8].

The points that voted for this plane will then be transformed based on this matrix and projected on the x-y plane. Each of these points will be rendered as a black circle on the plane; the Hough transformation will then be applied again to the rendered image. The extracted lines will be displayed as strokes in our system.

Use Cases

In this section, we present and analyze two use cases of our system (avoid passive voice). These cases are originated from makerspace projects, each with their real-world application scenario. The creator is familiar with 3D modeling software (Fusion 360, Maya, etc.) and draws engineering drafts frequently.

The first case is a sensor rig aiming to collect light intensity data from different angles. By drawing on multiple planes in 3D space, the draft can be viewed from multiple perspectives. Below are the 3D drafts of a sensor rig (Figure 5), along with a similar real-world counterpart (Figure 6). This example demonstrates sketching based on surfaces.

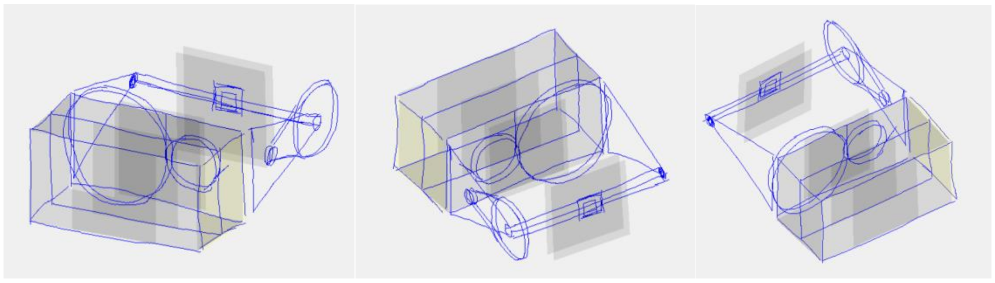

Figure 5. Multi-perspective views of a sensor rig.

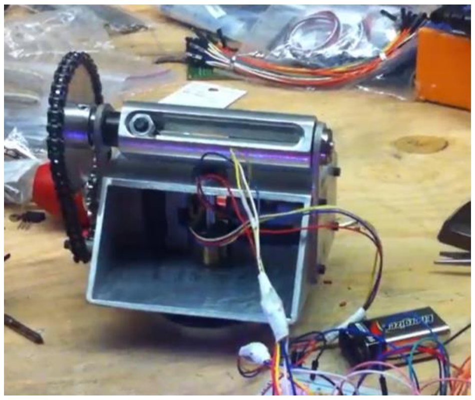

Figure 6. A similar real-world counterpart of the sensor rig.

The second example is a shelf inside a makerspace. We will rebuild this shelf in our makerspace on the new campus. Below shows the raw images, features extracted (Figure 7), feature matched (Figure 8), point cloud result (Figure 9), and result in our system (Figure 10).

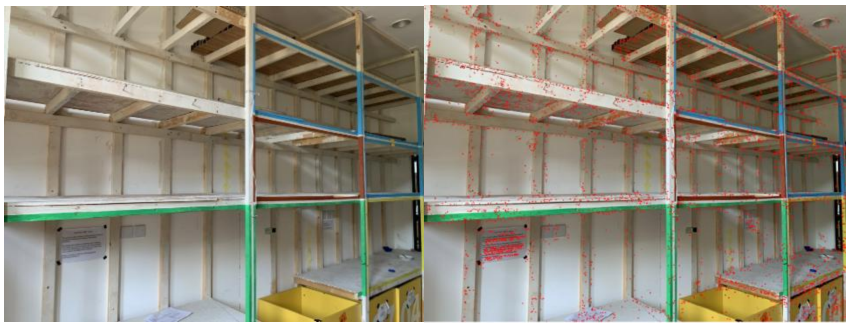

Figure 7. One original image of the shelf along with the features extracted (red dots).

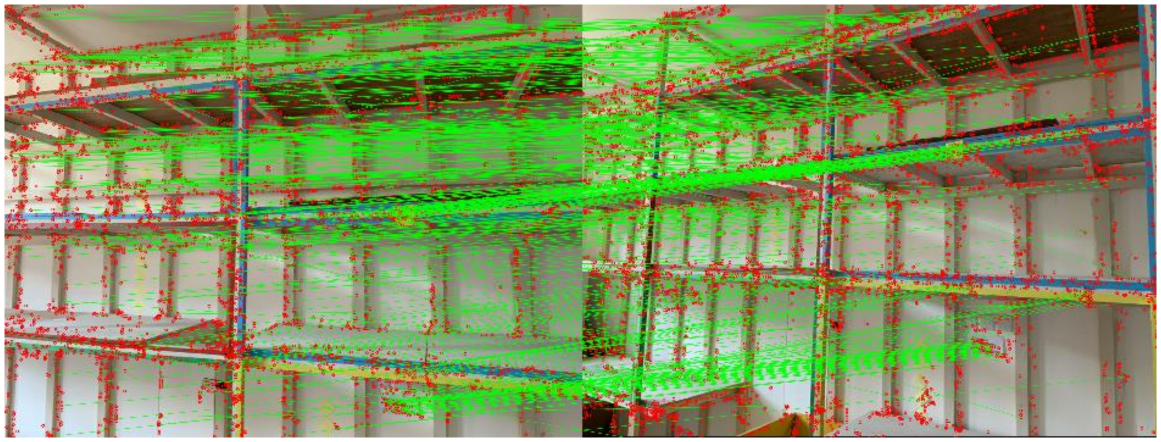

Figure 8. Features matched (green lines indicate successful matches).

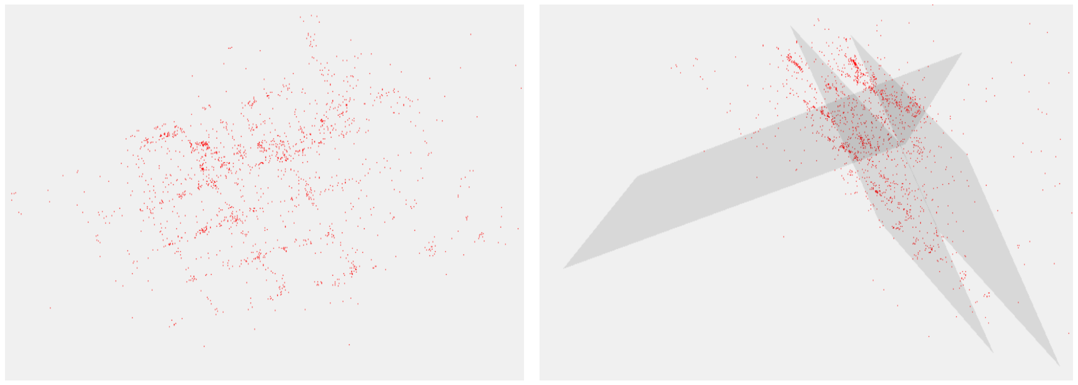

Figure 9. Point cloud result of 3D reconstruction along with the plane detection result.

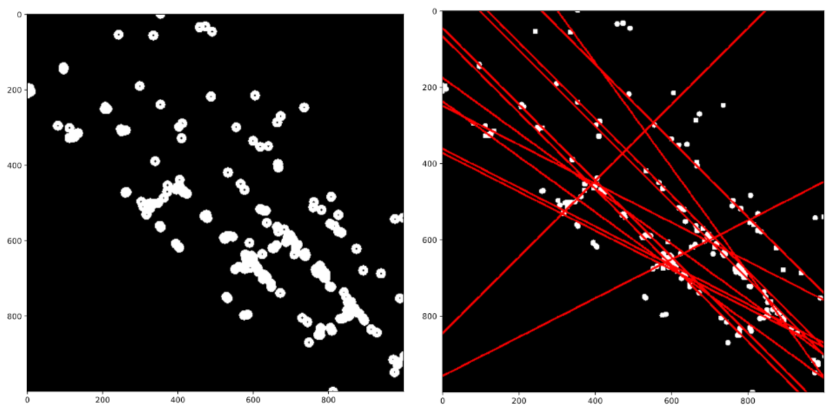

Figure 10. Left: Projection of near points to a detected plane. Right: Stroke extraction with a small threshold.

Our system successfully extracts distinct planes using our algorithm. Users can draw on the provided planes and add context details based on the result of the analysis. However, the automatic stroke extraction process is affected by noise: about 30% of strokes are wrong because of reconstruction errors. Users can set the scanning threshold to a bigger value, and draw the remaining strokes by hand on the calculated planes.

Conclusion

The ideas of traditional sketching, in-context design, and multi-perspective viewing are valuable but people usually overlook their connections. In this project, we explore the synergy between these three aspects, aiming to develop a designing tool without the heaviness of CAD or the tediousness of multi-perspective drafting. Without consideration of precise dimensions and effort in reconstructing the context, designers can work more fluidly in the ideation process.

However, many aspects can be furthered in this application. The reconstruction process based on images is rough and sensitive to noise; the algorithm derived can only extract lines; dozens of photos and a medium amount of time are required to complete the process. If the library used fails to reconstruct a dense point cloud, the thresholding process is likely to fail. Currently, devices that directly capture scene depth information may provide a better user experience.

The functions provided by this system has limitations. Features like making comments, multi-color drawing, drawing in layers, and generate views can be explored in the future. Our system offers an opportunity to design from a different perspective, and saves designers a lot of time in ideation.

Acknowledgments

I would like to thank Jiayu Lu for his help in ideation, debugging, and experimenting with the system in this paper. Thanks to Jaguar Kristeller for his real-world project in Figure 6. Thanks to the makerspace of Moonshot Academy for supporting the projects mentioned in the examples.

References

[1] Dorsey, Julie, and Leonard McMillan. "Computer graphics and architecture: state of the art and outlook for the future." *ACM SIGGRAPH Computer Graphics* 32.1 (1998): 45-48.

[2] Tolba, Osama, Julie Dorsey, and Leonard McMillan. "A projective drawing system." *Proceedings of the 2001 symposium on Interactive 3D graphics*. 2001.

[3] Dorsey, Julie, et al. "The mental canvas: A tool for conceptual architectural design and analysis." *15th Pacific Conference on Computer Graphics and Applications* (PG'07). IEEE, 2007.

[4] Paczkowski, Patrick, et al. "Insitu: sketching architectural designs in context." *ACM Trans. Graph*. 30.6 (2011): 182.

[5] Schonberger, Johannes L., and Jan-Michael Frahm. "Structure-from-motion revisited." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2016.

[6] Hart, Peter E., and R. O. Duda. "Use of the Hough transformation to detect lines and curves in pictures." *Communications of the ACM* 15.1 (1972): 11-15.

[7] Hearn, Donald, M. Pauline Baker, and Warren R. Carithers. *Computer graphics with OpenGL*. Upper Saddle River, NJ: Pearson Prentice Hall, 2014.

[8] Polaris878. "Finding quaternion representing the rotation from one vector to another." https://stackoverflow.com/questions/1171849/. Accessed on 9/13/2020.