An Application of Steganography in Point Set Models

Ji Wang

Abstract

This report discusses an implementation of an algorithm that hides and extracts information in point set models. Combined with personal interest, I explored the possibilities of doing steganography in geometrical input, specifically in point set models. I tried implementing an algorithm that embeds information into point set models and extracts information from it. This algorithm uses an interval structure proposed by Chung-Ming Wang and Peng-Cheng Wang in 2006[1], which turns a three-point structure into a two-state object. Through testing in some models, the program works as expected while having visible vulnerabilities, which will be discussed along with several possible improvements in the future.

1. Introduction

Advancements in digital communication have made it challenging to send secure messages to intruders. Cryptography converts messages into incomprehensible forms but can create suspicion. Meanwhile, information hiding hides secret information in an innocuous file, making it practical and secure [2].

Watermarking and steganography are two methods of information hiding. Steganography is better for unsecured communication channels and embeds secret information inside a host file to create a stego file that is perceptually similar to the host file. The quality of a steganography algorithm depends on the hidden message's imperceptibility, the approach's robustness, and the capacity to carry a message of at least a quarter of the size of the host file [2].

The inspiration for this project stems from a previous endeavor in my early exploration of computer science. I developed an algorithm that hides information in images by subtly altering pixel values. Each pixel possessed a vast range of 256 possible values, allowing for ample storage capacity while remaining imperceptible to the human eye. This experience also sparked my curiosity about the potential to conceal information in geometric models while studying computational geometry.

When analyzing various input models for data processing, I initially believed that the point set model only consisted of one data format - points. As I am relatively new to this subject, I began brainstorming ways to utilize this seemingly simplistic input model effectively.

A paper introduced an interval structure that allows point structures to represent two-state objects, including bit sequences. I incorporated this interval design into my point set and added a secret key for randomization to ensure security. With this, I developed a program to hide and extract text information in point set models with added security measures. The output stego model is designed to have minimal visual differences to avoid detection. Additionally, my implementation allows for recovering the original point set when extracting information.

This report will delve into related research, the algorithm design, and my implementation process. I will also present the results of my testing, discuss potential vulnerabilities and suggested improvements, and conclude with the significance of my work.

2. Related Works

Since I have limited experience hiding information in geometric-based data, I started reading some papers about some interesting techniques that are useful for this project.

A technique for hiding data on 3D polygonal models was presented by Cayre and Macq [3]. It is a blind spatial domain-based approach that involves selecting admissible triangles from a list of triangles in the model to embed a bit. Each triangle is considered a geometrical object with two states. Though this technique is robust against translation, rotation, and scaling operations, it is not resistant to other mesh-related operations.

In a separate publication, researchers presented a pair of inventive techniques for hiding data within point-sampled geometry via principal component analysis[1]. These methods have the capacity to embed single or multiple bits within each macro embedding primitive while maintaining large data capacity, robustness, and minimal visual distortion. Additionally, this paper proposes an innovative approach to enable point structures to represent states, allowing points to harbor pets. This structure forms the basis of the implementation of this project.

3. Algorithm

The algorithm comprises an embedder and an extractor. The embedder assimilates the payload text into the designated point set model, while the extractor retrieves the embedded information from the stego model and restores the original point set. A secret key is required to determine the random locations for embedding the information.

3.1 Embedding

The embedding process involves taking a payload in text format, a point set model (a collection of points), and a secret key generated by the user for randomization. The output of this process is a stego model that contains embedded information and extra bits, which are bit sequences necessary for recovering the original model.

This implementation accepts point set models in .txt or .xyz files and uses a text file for the payload, although any data that can be converted into bit-representations can be used. A 32-bit long key in a text file generated by openssl with base64 specification is required for security. The stego model is the modified model after the embedding process, and ideally, it should have minimal visual differences compared to the original model. The decision to include or exclude the extra bits depends on user expectations regarding the recovery of the original model, but it does not affect the extraction of payload information.

To handle the input point set model, we first separate each point's x, y, and z coordinate values and create a sorted list for each axis.

A crucial structure in this project is the interval structure, which Chung-Ming Wang and Peng-Cheng Wang first proposed in a paper[1]. In this project, we use an interval structure to handle the point set in the following way:

1) An interval consists of three consecutive values from the same axis, which are all x, y, or z coordinate values of points. They are arranged in ascending order since we keep the three lists sorted. We treat the first value as the start, the second as the middle, and the last as the end.

2) Using the start and end, we can calculate the symmetry axis by taking their average. Then, we determine the relative position of the middle value to the symmetry axis. If it falls to the left of the symmetry axis, we define the interval as having the state "0"; otherwise, it has the state "1".

3) If the bit we want to embed in the interval has the same value as the interval's original state, then we do not change the interval. However, if they differ, we flip the middle value across the symmetry axis. This modification changes the interval's state, representing the bit's value.

With this definition, an interval of three values can be viewed as a two-state object, and encoding a bit in the interval can be accomplished by changing its state as per the desired bit. Moreover, it is crucial to maintain the original start and end values of the interval to preserve the symmetry axis and position of the central value. The consistency of all symmetry axes is pivotal in ensuring a successful recovery. By ensuring that an interval's start and end values remain unchanged, neighboring intervals can share a start or end to maximize capacity. Thus, we can obtain (N-1)/2 intervals (rounded down) for a list of length N. So, for a 3D point set with N points, we could obtain approximately 1.5N intervals, enabling us to embed 1.5N bits in it.

When embedding a bit into an interval, the resulting interval only reflects the intended bit from the payload, regardless of whether a change happened. It is necessary to keep track of any state modifications during the embedding process to restore the initial state of an interval. We record a bit "1" if a change happens; otherwise, we record a bit "0". This recording provides additional bits of equal length to the payload, which will be kept confidential and treated as a security measure.

Remember that we have been provided with a secret key as input. Now that we know how to form intervals with the given point set, we must decide where to embed each bit randomly. This randomness is essential to ensure security. This implementation addresses this by utilizing the key as a seed and randomly assigning each interval for embedding. The assigned intervals do not repeat to ensure the embedded bits remain secure.

3.2 Extraction

In order to successfully extract information from the stego model, it is essential to input both the secret key and extra bits. Firstly, we utilize the extra bit to identify any state changes during embedding. By reading the length of the extra bits, we can also determine the number of bits embedded in the model. Next, we employ the secret key to reconstruct the indices of the intervals where each bit was embedded. Through this process, we can recover the original state of each interval and the specific bit embedded within it using the extra bit. We can successfully retrieve the embedded payload and the original model by applying this approach to all intervals.

4. Testing Results

I utilized the CGAL library to implement the algorithm in C++ programming language and successfully conducted tests on a personal computer equipped with an Apple M2 chip running macOS 14.1.2.

Fig. 1 demonstrates testing results before and after the embedding process for some models. The point set models, consisting of approximately ten to twenty thousand points each, were sourced from the CGAL library examples. To enhance visual comparison effects, I distinguished altered points during embedding as red and unaltered points as blue. The payload text utilized for testing was a paragraph in English, converted to 4064 bits. In my program, I generated a secret key using openssl with the base64 standard and a length of 32 bits.

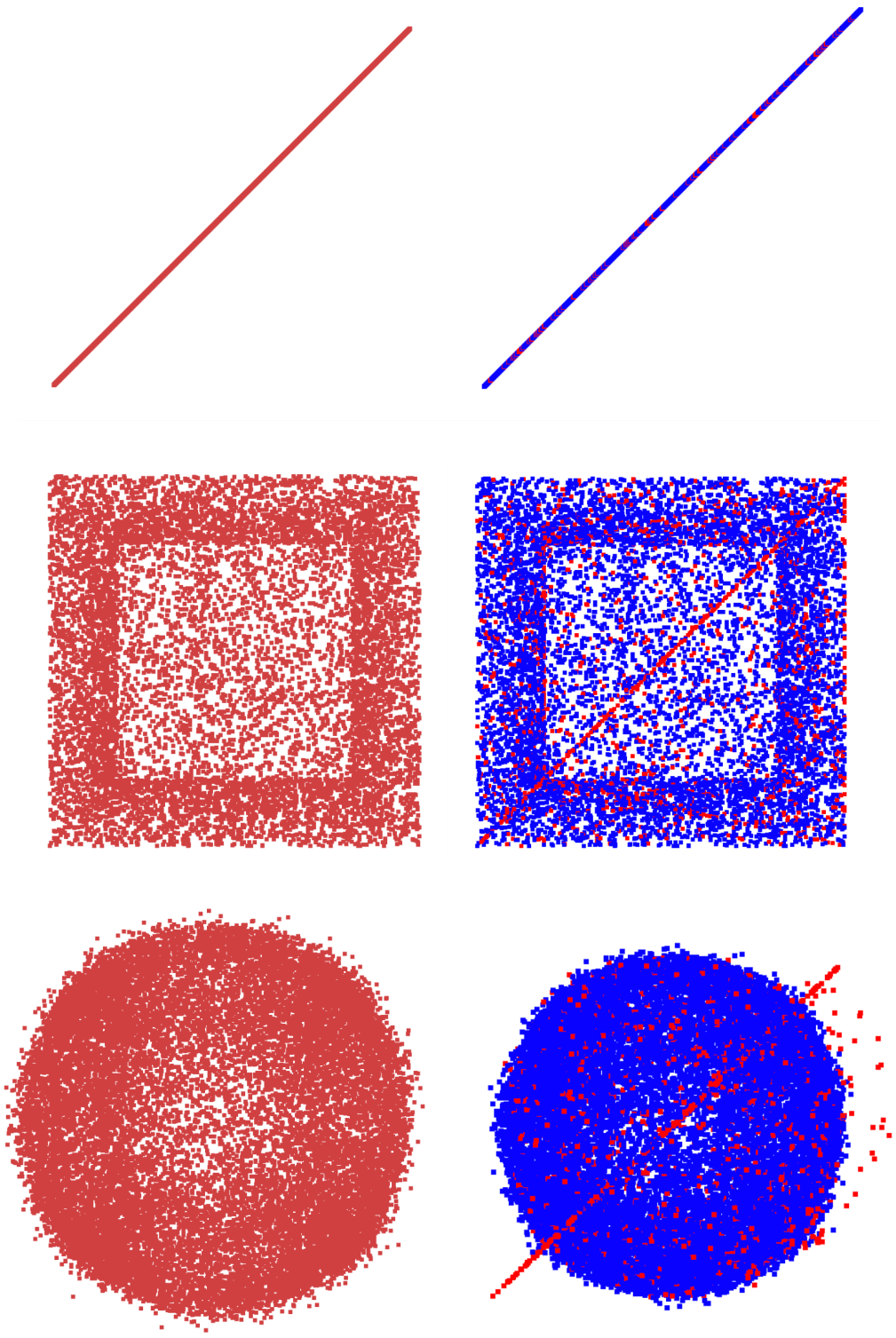

Fig. 1. Original Models(left) and Stego Models(right) for **Straight Line, Cube, Sphere, Kitten, and Radar.** In the stego models, blue represents unchanged points during embedding, and red represents changed points during embedding.

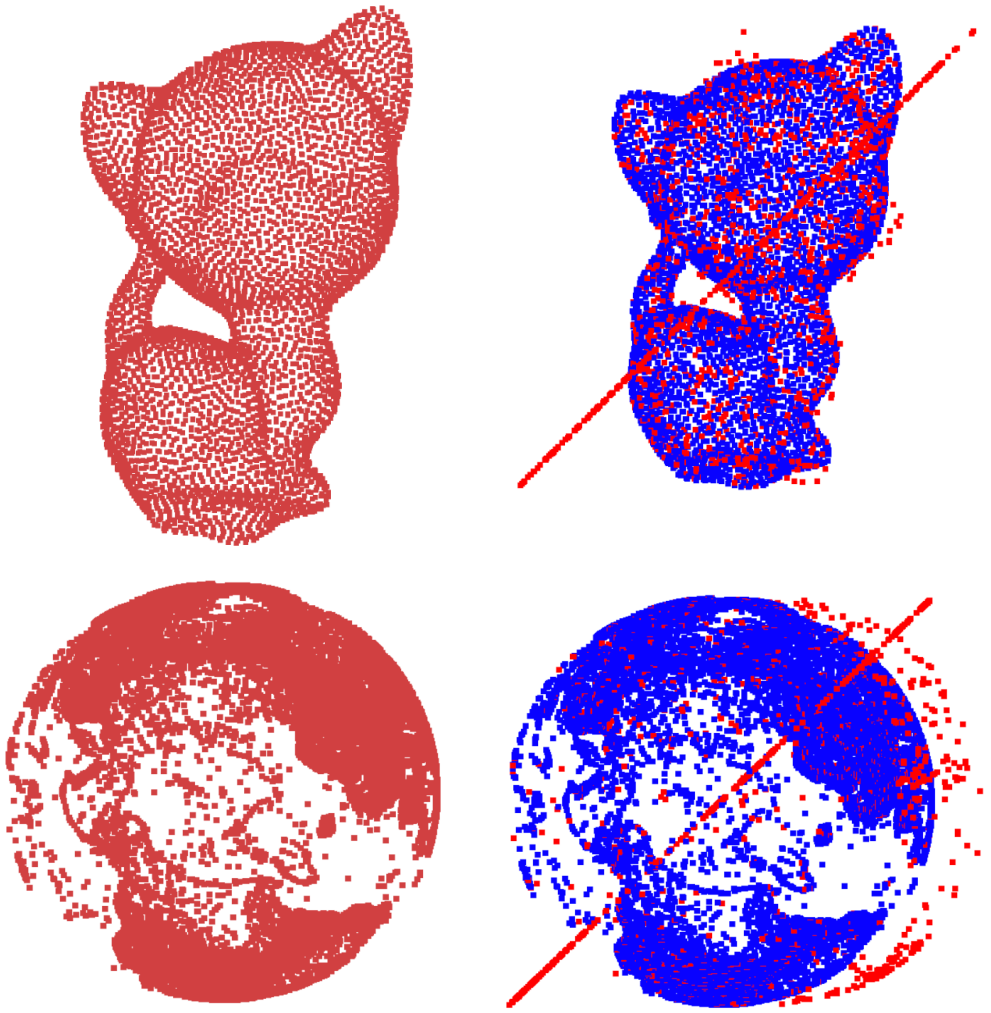

Fig. 1. (Continued)

5. Issues and Vulnerabilities

During thorough testing, I identified specific vulnerabilities in the algorithm I had designed and implemented, following a rigorous analysis of various input point set models. Attached are a few images that illustrate comparisons between the original point set models and the stego models.

5.1 Axis Choices

The testing images show an apparent red line crossing the center of the stego models. This issue is unexpected and makes steganography lose much of its strength since steganography is all about being unnoticeable.

The problem can stem from inadequate axis choices in embedding. The input point set model is utilized directly to construct the axes, potentially leading to alterations in the middle value of an interval that make the changes more apparent. One potential solution is utilizing PCA analysis to calculate the input model's eigenvectors and gravity center and then reposition all points accordingly. This approach would ensure that significant changes to the middle values in intervals would occur in the same direction as the eigenvectors, resulting in less noticeable changes.

5.2 Payload Recovering Accuracy

Currently, the payload recovery rate is only sometimes 100%, which implies that the payload may only be partially extracted at times. While the "bad" axis choice discussed in the preceding section could contribute to this, "bad" interval choices could also be responsible.

Following the random assignment of intervals for each bit, the program proceeds to the embedding operations. However, some bits could be allocated "bad" intervals, where the start and end values are in close proximity or even identical. Such intervals could cause precision issues during embedding, resulting in a disturbed order of intervals. Consequently, these "bad" intervals could cause the extraction process to locate the wrong intervals, resulting in the recovery of incorrect bits.

Potential precision concerns may arise during the embedding or extraction of bits, and it may be beneficial to utilize a filter that discards intervals where the start and end values differ by less than a predetermined threshold. While this method can help maintain accuracy and dependability, it may result in a capacity of less than 1.5N in the point set model.

5.3 Attack Handling

At this point, the implementation cannot withstand attacks on the stego model, which can lead to extraction issues. However, there are avenues for enhancement. Incorporating PCA analysis[1] can bolster the program's resilience to attacks like translation, rotation, and scaling. Additionally, integrating watermarking into this algorithm could aid in detecting potential attacks on the stego model.

6. Conclusion and Future Work

This report presented an implementation of an algorithm that hides and extracts information in 3D point set models using steganography. The algorithm uses an interval structure proposed by Chung-Ming Wang and Peng-Cheng Wang in 2006[1], which turns a three-point structure into a two-state object. With a 3D point set model of N points, this algorithm can theoretically embed 1.5N bits at most. The program works as expected in testing, although there are visible vulnerabilities and inaccuracies, as discussed, and several possible future improvements.

This project shows the capabilities of steganography in geometric inputs and acknowledges the valuable prior work in this area. Several potential improvements could be explored in future research to enhance this implementation further. As previously discussed, incorporating PCA analysis could improve the model's accuracy, and watermarking techniques would verify whether the stego model has been attacked. Additionally, encrypting the payload text before embedding would provide an additional layer of security. Furthermore, it may be beneficial to intelligently select the optimal intervals for embedding bits by analyzing the complete point set or implementing advanced machine-learning techniques.

References

[1] Wang, C., & Wang, P. (2006). Steganography on point-sampled geometry. *Computers & Graphics*, *30*(2), 244-254. https://doi.org/10.1016/j.cag.2006.01.030

[2] Girdhar, A., & Kumar, V. (2018). Comprehensive survey of 3D image steganography techniques. *IET Image Processing*, *12*(1), 1-10. https://doi.org/10.1049/iet-ipr.2017.0162

[3] F. Cayre and B. Macq, "Data hiding on 3-D triangle meshes," in IEEE Transactions on Signal Processing, vol. 51, no. 4, pp. 939-949, April 2003, doi: 10.1109/TSP.2003.809380.