A numerically stable fragile watermarking scheme for authenticating 3D models

Wei-Bo Wang, Guo-Qin Zheng, Jun-Hai Yong, He-Jin Gu

A numerically stable fragile watermarking scheme for authenticating 3D models

Wei-Bo Wang [a,b,*], Guo-Qin Zheng [a,b], Jun-Hai Yong [a,b], He-Jin Gu [c]

[a] School of Software, Tsinghua University, Beijing, PR China
[b] Key Laboratory for Information System Security, Ministry of Education of China, Beijing 100084, PR China
[c] Jiangxi Academy of Sciences, Nanchang 330029, PR China

ARTICLE INFO

Article history:
Received 4 January 2008
Accepted 18 March 2008

Keywords:
Fragile watermarking
Mesh authentication
Error prevention
Tamper detection

ABSTRACT

This paper analyzes the numerically instable problem in the current 3D fragile watermarking schemes. Some existing fragile watermarking schemes apply the floating-point arithmetic to embed the watermarks. However, these schemes fail to work properly due to the numerically instable problem, which is common in the floating-point arithmetic. This paper proposes a numerically stable fragile watermarking scheme. The scheme views the mantissa part of the floating-point number as an unsigned integer and operates on it by the bit XOR operator. Since there is no numerical problem in the bit operation, this scheme is numerically stable. The scheme can control the watermark strength through changing the embedding parameters. This paper further discusses selecting appropriate embedding parameters to achieve good performance in terms of the perceptual invisibility and the ability to detect unauthorized attacks on the 3D models. The experimental results show that the proposed public scheme could detect attacks such as adding noise, adding/deleting faces, inserting/removing vertices, etc. The comparisons with the existing fragile schemes show that this scheme is easier to implement and use.

1. Introduction

With the rapid development of internet technologies and the growth of multimedia contents, transferring digital media data via internet becomes more and more popular in recent years. These valuable data, however, are easily illegally distributed or tampered. Thus, it is desirable to develop a strong method to protect the copyright and the integrity of the digital data [10,11,29].

Watermarking has been viewed as a solution for the requirement above [1,2,8,16]. Compared with other possible solutions, such as encryption and digital signature, watermarking costs less and is computationally cheaper [34]. According to the application purpose, watermarking can be divided into the robust watermarking and the fragile watermarking. The robust watermarking focuses on protecting the ownership of digital data while the fragile watermarking intends to authenticate the integrity of digital data [10]. In general, the robust watermarking should tolerate heavy attacks and keep detectable in the extracting stage. On the other hand, the fragile watermarking is aimed to detect the slightest unauthorized attacks and to locate where the attacks happen. Both the robust and the fragile watermarking should be perceptual invisibility [1]. Another classification criterion divides watermarking into public and private according to whether the original media is needed during the extracting stage [10,11]. A private watermarking scheme references the original media to extract the watermarked information while a public scheme does not need the original media. In the fragile watermarking, the public schemes are preferred. It is because fragile schemes have to encrypt and transfer the original models for extracting the watermarks. And the encrypting and the transferring steps are costly [34]. Finally, according to the insertion domain, researchers classify the watermarking schemes into the spatial domain and the spectral domain.

Currently, there exist several watermarking schemes on different kinds of media, such as still images, video and audio [1–7,16]. 3D models are relatively new media forms on World Wide Web for many applications, such as CAD/CAM. Some watermarking schemes have been proposed to watermark the 3D models [8–15,17–21,26,27,29,34,37]. These schemes are primarily on meshes, which are considered as the "lowest common denominator" of 3D surface representations - it is easy to convert other representations to meshes [12]. Ohbuchi listed several embedding primitives in the spatial domain for 3D watermarking embedding [8], such as coordinates of a point, length of a line, etc. For the consideration of robustness, the 3D features, which are invariant to common transformation, are used to embed the watermarks. These features include the angle between polygons [41], the length of lines, the radius of curvatures, the distance to the mesh center [23,24,27,36,37], normal vectors [11,39,40], and the relative position of a vertex to its 1-ring neighbors [10,29,34,35]. Some schemes transform the

* Corresponding author at: School of Software, Tsinghua University, Beijing, PR China. Tel.: +86 010 62778801.
E-mail address: wwb84nju@gmail.com (W.-B. Wang).

models into the spherical coordinate systems for robustness [26,31,42]. Compared with the spatial domain algorithms, the spectral domain algorithms are more robust and preserve the visual effect of the models better. It is because the inserted watermark bits are diluted in all the spatial parts of the models. Some commonly used spectral analysis tools include Laplacian basis functions [22,25,28,38], radial basis functions [33] and multi-resolutions such as wavelet [9,26,32]. The drawbacks of the spectral domain algorithms are that they are complicated and computationally expensive compared to the spatial domain algorithms.

In 3D fragile watermarking, Yeung and Yeo [10] slightly perturb the vertices such that a certain hash function of each vertex's coordinates matches another hash function applied to the centroid of its neighboring vertices. Lin et al. [29] apply two different hash functions on the coordinates of vertices and perturb the vertices until the two functions are matched. In [27], Wu and Cheung quantize the distance between the centroid of the mesh models and each surface to embed the watermarks. Chou and Tseng [34] apply multi-function and adjusting-vertex method to embed the watermarks. Despite their reasonable success, these schemes suffer from several drawbacks. Among them the numerically instable problem, to our best knowledge, has not been mentioned in any reported literature.

The remaining sections are organized as follows. Section 2 analyzes the problems encountered in these schemes, primarily the numerically instable problem. In Section 3 we propose our scheme, which is numerically stable and can handle the problems mentioned. The discussion of the performance and the experimental results are provided in Section 4. Finally, conclusions are given in Section 5.

2. Problems in previous work

In this section, we analyze the problems in the existing schemes, particularly the numerically instable problem, and give possible solutions to them. A watermarking algorithm, combined the solutions together, is proposed in Section 3.

Numerically Instable Problem: The numerically instable problem is accounting for the majority of computational errors in CAD systems [30]. Currently this problem also arises in some fragile 3D watermarking schemes. However, as we known no literature has reported this.

The fragile watermarking schemes must compare the extracted bits with the original ones in order to determine if the integrity of the model has been destroyed. If the two bits are not equal, the schemes can demonstrate unauthorized changes. Watermarking calculations are primarily based on the floating-point arithmetic [34] or on the integral arithmetic [10,27,29,37]. If the operations are based on the floating-point arithmetic, the schemes need to find a proper tolerance to equal for comparing. If the tolerance is too small, the bits to be compared will be determined as not equal, even if no attack has happened. However, if the tolerance is too large, the attacks will not be found since the bits are determined as equal. Unfortunately, the tolerances differ from one embedding operation to another, which means that every time the schemes embed a watermark bit, they have to calculate a new tolerance. This means that the number of calculating the tolerance equals the number of the embedding operations. It is expensive. Furthermore, these tolerances have to be attached to the model and transferred together. This increases the transferring cost remarkably since the number of tolerances equals the number of embedded vertices. It is unreasonable only using the maximum tolerance as a criterion. It will enlarge the tolerance at most cases in which some unauthorized changes may not be detected.

The method in [34] does not consider selecting proper tolerances to equal. When no tolerance to equal is given, in practice three methods are commonly used to compare two floating-point numbers. The first one is using a very small tolerance, such as 1e-10. The second one is splitting the decimal fracture of the two floating-point numbers and comparing them as integers. The third one is rounding-off the two floating-point numbers and comparing the rounded integers. However, even when no attack happens, the method in [34] fails to determine whether the original bits and the extracted bits are equal using these methods because of the numerically instable problem.

Two formulae and several parameters are used in [34] for embedding and extracting the watermarks. The formulae are

$$x' = \begin{cases} x - (|x - x^c| \bmod k^q) + \dfrac{k^q}{k^d} w & \text{if } x > x_c, \\ x + (|x - x^c| \bmod k^q) - \dfrac{k^q}{k^d} w & \text{otherwise,} \end{cases}$$

and

$$w' = (|x^c - x'| \bmod k^q) \frac{k^d}{k^q}.$$

In the formulae, x indicates one of the three coordinates of a vertex, x^c indicates the centroid of the neighboring vertices of x, k^q and k^d are the embedded parameters, w is the watermark bit before embedding and w' is the extracted bit. The two formulae could guarantee $w = w'$ based on the fact that $(a - b - (a - b) \bmod c) \bmod c = 0$, where $a \geq b$ and $a, b, c \in P$. The method in [34] applies the first formula twice to embed w and h into x_2 and x_3 of a vertex respectively. x_2 and x_3 are the second and the third parts of the coordinates of a vertex. The relationship between w and h is that $h = \mathit{hash\,function}(w)$. The relationship $h' = \mathit{hash\,function}(w')$ should still be kept because the formulae can guarantee that the extracted bits are equal to the original bits. However, this guarantee may not be held when x, w and other parameters are floating-point numbers because of the numerically instable problem. Table 1 gives the data of an example that violates this guarantee. In Table 1, x_2 and x_3 are the second and the third parts of the coordinates of the vertex. As the two embedded bits keep the relationship $h = h(w)$, the two extracted bits, w' and h' should still be equal. In Table 1, however, the two extracted bits are 2.7847 and 1.9359 when no attacks happen. Neither one of the three common used methods can be used to determine the two extracted bits equal. The method in [34] fails here since it does not give the tolerances to equal, and the further reason is the numerically instable problem.

The best way to deal with errors is to prevent them from happening [30]. Thus, selecting watermarking schemes based on the integral arithmetic is a cheap and easy way to settle the numerically instable problem. But, even if the schemes are based on the integral arithmetic, the floating-point numbers have to be converted into integers first because 3D models are mostly represented in the floating-point numbers. A common converting method is splitting the decimal fracture part of a floating-point number and the schemes in [10,27,29,37] apply this method. However, this method has drawbacks in practice. Several attacks, such as adding noise, may also only change the decimal fracture part of the coordinates of vertices. These attacks may not be detected since the modified parts have been discarded during the converting operation before detecting. Therefore, designing a proper converting method is important for improving the stability of 3D fragile watermarking algorithms. A possible solution is to view the mantissa part of the floating-point number as an unsigned integer since it does not lose any original information.

Causality Problem: This problem first arises in Yeo's scheme [10]. The method in [10] orderly perturbs the vertices until they match the predefined relationship with the centroid of their neighboring vertices. However, the latter perturbed vertices, which are the

Table 1
An example of the numerical instabilities make the algorithm in [34] disabled

Label	Original value(x)	Centroid of the neighboring vertices (x^c)	Modified value (x')	Bits before embedding (w)	Extracted bits (w')
x_2	1226.6700	1233.5875	1226.6772	2	2.7847
x_3	157.97501	171.07650	157.97630	2	1.9359

Data structure: IEEE-754 float32. Parameters: $k_2^q = k_3^q = 0.01$, $k_2^d = k_3^d = 0.01$, $h_i = h(w_i)$.

neighboring vertices of the former perturbed ones, may influence the relationship the former vertices already satisfied.

To settle this problem, in [10] they define an ordered traversal. The perturbing of vertices could only involve the processed ones. For example, if the traversal order is $v_1, v_2, v_3, \ldots, v_n$, when perturbing v_3, only v_1 and v_2 could be involved in the relationship calculation. A drawback of this constraint is that the capability for detecting modifications will be ruined if the predefined traverse order has been changed or becomes untraceable. A vertex renumbering operation may totally disable the capability of the scheme [34]. In [29], it does not take the neighboring relationship into account when embedding the watermarks. Thus their solution eliminates the causality problem. However, this method can not detect the attacks on the topological relation, such as deleting a vertex from the model. This reduces the detecting ability of their scheme. In [34], it applies a method called adjusting method to handle this problem. It introduces the concept of adjusting vertex into the embedding stage. Each mark vertex is assigned with an adjusting vertex. An adjusting vertex is no longer eligible as a mark vertex. As mentioned above, a mark vertex suffers from the perturbation of its neighboring vertices and fails to keep the predefined relationship. In [34] they first store the unwanted amount of the perturbation of a mark vertex on its adjusting vertex. After all the mark vertices have been perturbed, the method in [34] perturbs the adjusting vertices to balance the unwanted amount of the perturbation on the mark vertices. However, this method still fails to thoroughly settle the causality problem.

Fig. 1 is a case that the method in [34] fails to settle the causality problem. In this part of a mesh model, the vertices A, B and E are selected as the mark vertices. The vertices C, D and F are their adjusting vertices respectively. The method in [34] perturbs the positions of A, B and E to meet the predefined relationship. However, when B is perturbed, it influences the relationship between A and its neighboring vertices, which are B, C and D. This is the causality problem. The method in [34] stores the error caused by B and latterly adjusts C, the adjusting vertex of A, to balance this error. Similarly, it performs the same operations on D and F to balance the errors caused by the causality problem. However, D is also the neighboring vertice of A. And the adjusting operation on D also interferes the relationship on A. The method in [34] fails to take this case into account.

So far, we found that the root of the causality problem is that the vertices to be perturbed, either the mark vertices or the adjusting vertices, are allowed to be adjacent. Consequently they could interfere with their neighboring vertices. One possible solution to this problem is forbidding the perturbed vertices to be adjacent.

There still remain some other problems in the existing schemes. The scheme in [27] is not a public one, the schemes in [15,27] could not locate the tampering regions, and the schemes in [10,29] lack the ability to control the distortion ratio caused by watermarks embedding. In Section 3, we propose a numerically stable fragile watermarking scheme intending to overcome these mentioned problems.

3. The proposed watermarking scheme

In this section, we first describe in detail the procedure of embedding and extracting the watermarks. The mesh models we operate on are stored in the binary format files. Thus the precision loss caused by converting the data from the ASCII format to the binary format or vise versa is not considered in the following discussion. After describing the embedding and the extracting procedure, we select a proper portion of vertices from the model for embedding the watermarks. We offer a *locating procedure* to locate the tampering regions. An overview of our watermarking scheme is then given. At the end of this section, the flowchart of this scheme is presented and the computational complexity is analyzed.

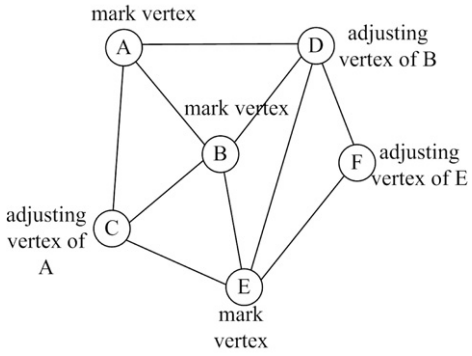


Fig. 1. A failure case of the scheme in [34]. C balances the error caused by B. However, D is also perturbed in order to balance error caused by E. This induces a problem because perturbing D also influences the relationship between A and its neighboring vertices.

3.1. Watermarks embedding and extracting method

Denoted the 3D model as $M(V, C)$, where V is the vertex set and C is the topological relation. A watermark sequence $W = (w_1, w_2, \ldots, w_n)$ is embedded into M by displacing a subset of V a small amount. In the following part we call the vertices to be watermarked as mark vertices and the other vertices as non-mark vertices. A vertex v has three coordinates, denoted as (fx_1, fx_2, fx_3), which mean that the three numbers are stored in the floating-point format. The following approach shows how to embed a watermark bit w_i into a vertex v.

First we define two operations, which are the mantissa picking up operation and the mask operation(see Fig. 2). The floating-point format defined in IEEE-754 consists of three parts: a sign part, an exponent part and a mantissa part [43]. The mantissa picking up operation is used to pick up the mantissa part from a coordinate in the floating-point format. After picking it up, the mask operation is used to select several bits from the mantissa part. These selected bits, arranged sequentially, will be modified to embed w_i. Both the mantissa picking up operation and the mask operation have their reversal operations(see Fig. 3). We use the two reversal operations to write the watermarked data back into the original coordinates.

Fig. 2 illustrates the procedure of the mantissa picking up operation and the mask operation. Here fx_i in Fig. 2 is a coordinate in the IEEE-754 floating-point format. The picked mantissa, called mx_i, operates bitwise AND with the given $mask_i$. In the following discussion, we will call the bits equaling '1' as the 1s bits. The 1s bits in $mask_i$ are called valid bits and their positions are called valid positions. Our scheme could detect the attacks which modify the bits in the valid positions. nx_i is composed of the result bits of the bit operation in the valid positions. MAX_i in Fig. 2 indicates the

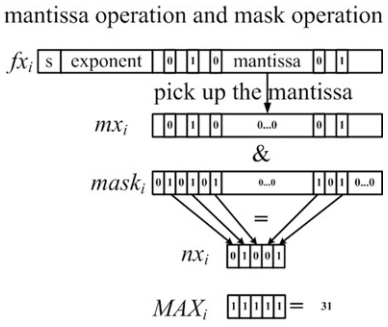


Fig. 2. The example illustrates how the mantissa picking up operation and the mask operation work. Data are in the IEEE-754 float32 format. We ignore displaying some bits in the invalid positions in this figure.

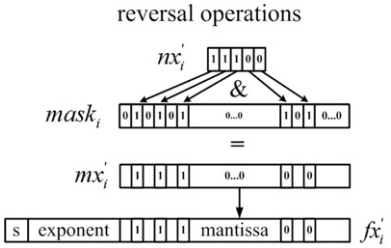


Fig. 3. The example shows how the reverse mantissa picking up operation and the reverse mask operation work. Data are in the IEEE-754 float32 format. We ignore displaying some bits in the invalid positions in this figure.

upper bound of nx_i. Fig. 3 illustrates the reversal operations of the picking up operation and the mask operation. The reverse picking up operation writes the modified nx_i' back to the corresponding positions in mx_i'. And the reverse mask operation writes the mx_i' back to fx_i to form the watermarked data fx_i'.

Watermarks embedding method: The *watermarks embedding method* is used to embed a watermark bit w_i into a vertex v. Before the embedding operation, it must identify v as a mark vertex or a non-mark vertex. The identity information is from the *mark vertex selecting principle*, which is illustrated later. In the *watermarks embedding method*, like [34], fx_1 is for embedding the identity information while fx_2 and fx_3 are modified to embed the watermarks. The difference between our scheme and the scheme in [34] is that we modify the mantissa part of the floating-point number. It is numerically stable and does not lose the detection ability. We first apply the *mantissa picking up operation* and the *mask operation* to convert fx_i into mx_i and nx_i. The embedding method is composed of the following five steps, S1–S5.

S1 This step is to embed the identity information. If v is a mark vertex, then nx_1 of v is modified to be equal to *key*, which is given by the user. If v is a non-mark vertex and nx_1 is equal to *key*, it will be changed to *key*/2. That is,

$$nx_1' = \begin{cases} key, & \text{if } nx_1 != key \text{ and } v \text{ is a mark vertex,} \\ key/2, & \text{if } nx_1 = key \text{ and } v \text{ is a non-mark vertex,} \\ nx_i, & \text{otherwise.} \end{cases} \quad (1)$$

The restriction on *key* is that $0 \le key \le MAX_1$, $key \in N$. Here we could find that the value of *MAX* determines the maximum possible distortion induced by the watermarking embedding. At the end of this step, we apply the *reversal mask operation* to calculate the modified mx_1' and apply the *reversal mantissa picking up operation* to write mx_1' back to fx_1'.

S2 This step modifies mx_2 and mx_3 of the coordinates of non-mark vertices. For $j = 2, 3$, we have

$$mx_j' = \begin{cases} mx_j, & \text{if } v \text{ is a mark vertex,} \\ mx_j | mx_1', & \text{if } v \text{ is a non-mark vertex.} \end{cases} \quad (2)$$

After calculating mx_2' and mx_3', we apply the *mask operation* to get nx_2' and nx_3'.

S3 If v is a mark vertex, this step generates the centroid of v's neighboring vertices. For $j = 2, 3$, we have

$$nx_j^c = \sum_{N(v)} nx_j' \%(|N(v)| \times key_j), \quad (3)$$

where $N(v)$ is the set of v's neighborhood, $|N(v)|$ is the size of $N(v)$, and key_j is the diffusion factor for changing the value space of the mod operation. The method generating the centroid here differs from what is used in the previous algorithms. This method can generate an integral centroid.

S4 If v is a mark vertex, this step continues modifying nx_2' and nx_3' of v to embed the watermark bit w_i through the XOR operator

$$nx_j' = \begin{cases} nx_j^c \bigoplus w_i, & j = 2, \\ nx_j^c \bigoplus h(w_i), & j = 3, \end{cases} \quad (4)$$

where $0 < w_i < MAX_2$ and $0 < h(w_i) < MAX_3$, $w_i, h(w_i) \in N$. Here h is a given hash function, such as $h(w_i) = w_i$.

S5 Finally, we apply the *reversal mask operation* and the *reversal mantissa picking up operation* to write nx_2' and nx_3' to fx_2' and fx_3'.

For a mark vertex v, through S1–S4 we successfully embed the watermark bit w_i into v. Otherwise, if v is a non-mark vertex, we modify its content via S1–S2 to indicate its identity. In short, after applying the *watermarks embedding method*, we successfully generate the watermarked data (fx_1', fx_2', fx_3') for a vertex in the model.

Watermarks extracting method: We now show how to extract the watermarks from the model. The *watermarks extracting method* also needs the identity information. The identity information can be generated directly from the watermarked model. The original model is not needed here. Thus our scheme is a public scheme.

When processing a vertex v in the watermarked model, we convert the watermarked data fx_1' of v into nx_1'. If nx_1' equals *key*, it will be labeled as a mark vertex and otherwise it will be labeled as a non-mark vertex. The identity information is gathered after labeling every vertex in the watermarked model. After gathering the identity information, we generate nx_2 and nx_3 for every mark vertex using the *mantissa picking up operation* and the *mask operation*. In addition, Eqs. (2) and (3) are used to calculate the centroid of the neighborhood of the mark vertices, which are nx_2^c and nx_3^c. Now we apply the formula

$$\begin{cases} w_i' = nx_2^c \bigoplus nx_2', \\ h_i' = nx_3^c \bigoplus nx_3', \end{cases} \quad (5)$$

to extract the watermark bits w_i' and h_i' from the mark vertex v. If the relationship from v, which is $h_i' = h(w_i')$, does not hold, it can demonstrate that attacks may happen on v or on its neighboring vertices.

3.2. Mark vertex selecting principle

In this step we describe how to select the mark vertices. We have analyzed in Section 2 that the causality problem could be eliminated by forbidding the mark vertices to be adjacent. Our principle applies this solution.

Mark vertex selecting principle: Traverse every vertex of the model and chose mark vertices following the discipline below.

The vertex chosen as a mark vertex must have no mark vertex among its neighboring vertices. Once a vertex has been chosen as a mark vertex, its neighboring vertices are no longer eligible.

Any traversal algorithm visiting each vertex exactly once could work in our principle. After selecting the mark vertices, the union of the mark vertices and their 1-ring neighborhood covers all

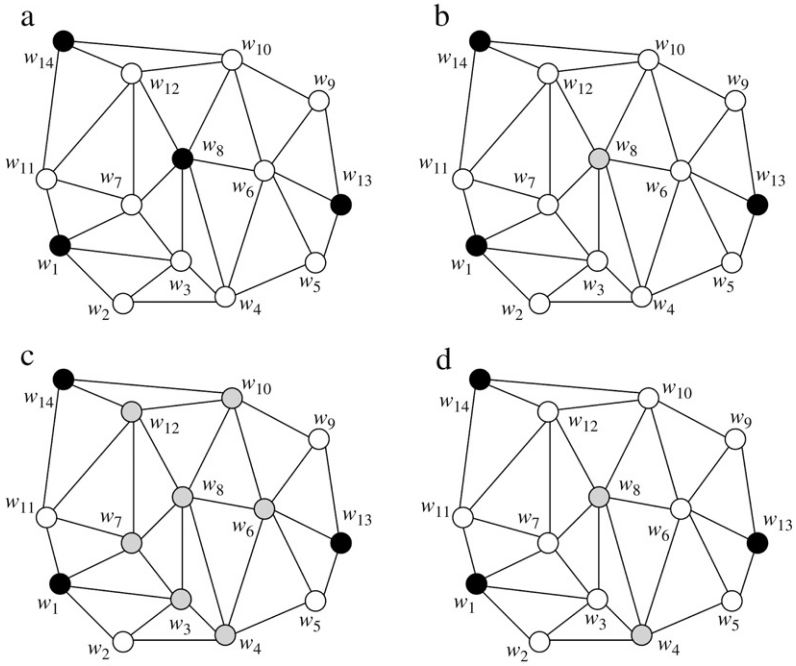


Fig. 4. A mesh with one mark vertex changed to a non-mark vertex. (a) The mark vertices have been chosen. (b) w_8 has been attacked and changed from a mark vertex to a non-mark vertex. (c) Several vertices have been labeled as suspicious. (d) The innocent mark vertices release the suspicion of some non-mark vertices. Finally w_8 and w_4 are the suspicious vertices. In these figures, the mark vertices are denoted in black, the non-mark vertices are denoted in white and the suspicious vertices are denoted in gray.

Table 2
Lists of five triangulated meshes used in our experiments and their detail information

Model names	Vertices/faces	Mark vertices	Model precision
Airplane	1335/2452	406	10^{-3}
Apple	867/1704	223	10^{-7}
Beethoven	2521/5030	694	10^{-6}
Cow	2903/5804	731	10^{-6}
Indy_car	4970/7119	1485	10^{-2}

the vertices of the model, leaving no embedding holes. It can be guaranteed that the mark vertices are uniformly distributed in the models.

This principle is quite simple, and it can thoroughly eliminate the causality problem. However, experimental data in Table 2 show that this principle chooses around 20%–31% of vertices as mark vertices. This rate is relatively low compared to 38%–45% in the method in [34] and 100% in the method in [10]. It increases the number of non-mark vertices attached to one mark vertex. And it may result in low locating accuracy: if a mark vertex is attacked, all its neighboring vertices and itself will be labeled as suspicious attacked vertices. The more non-mark vertices attached to a mark vertex, the larger the suspicious regions will be. More seriously, if a non-mark vertex is attacked, its adjacent mark vertices and all the non-mark vertices attached to these mark vertices will form the tampering regions. The tampering region has much more vertices than the attacked vertices. Thus it heavily reduces the locating accuracy. In order to achieve a good locating accuracy, we propose a locating procedure later, which could accurately locate the tampering regions.

3.3. Locating procedure

This procedure is used to accurately locate the tampering regions. Before the *locating procedure*, the identity information must be gathered. The *locating procedure* is composed of four parts, from P1 to P4.

P1 Check if there exist mark vertices in the neighborhood of every non-mark vertex. If not, set the non-mark vertices without neighboring mark vertices and the neighboring vertices of these non-mark vertices as suspicious vertices.
P2 Check if there exist mark vertices in the neighborhood of every mark vertices. If so, set the mark vertices with neighboring mark vertices and the neighboring mark vertices as suspicious vertices.
P3 Set the mark vertices which do not keep the pre-defined relationship, and the neighboring vertices of these mark vertices as suspicious vertices.
P4 Release the suspicion of the non-mark vertices adjoining the unsuspicious mark vertices.

Fig. 4 is an example of how this procedure works. In this example, the first coordinate of a vertex is modified and the vertex is changed from a mark vertex to a non-mark vertex. Fig. 4 (a) is a part of a mesh model where the mark vertices have been chosen. There are in all 14 vertices in this model, and among which 4 have been chosen as mark vertices, in black. Around 28.5% of the vertices have been chosen in this example as mark vertices, which is close to the experimental data. In Fig. 4 (b), w_8, the vertex in gray, has been changed to a non-mark vertex. There is no mark vertex in the region of w_8 and its neighboring vertices. Thus, they are labeled as suspicious. In Fig. 4 (c) they are shown in gray. In the end, in Fig. 4 (d), our scheme releases the suspicion of w_3, w_6, w_7, w_{10} and w_{12}. Finally, the suspicious vertices are w_8 and w_4. After the *locating procedure*, the vertices labeled as suspicious form the suspicious regions. These regions are displayed in different colors in order to visually show where the tampering operations happen. The reason why the *locating procedure* is effective is discussed in Section 4.2.

3.4. Overview of the scheme and the flowchart of the scheme

Fig. 5 is the overview of the scheme. In general the scheme is composed of two parts, which are the watermark embedding part and the watermark extracting part. In the watermark embedding part, the scheme first selects the mark vertices from the mesh

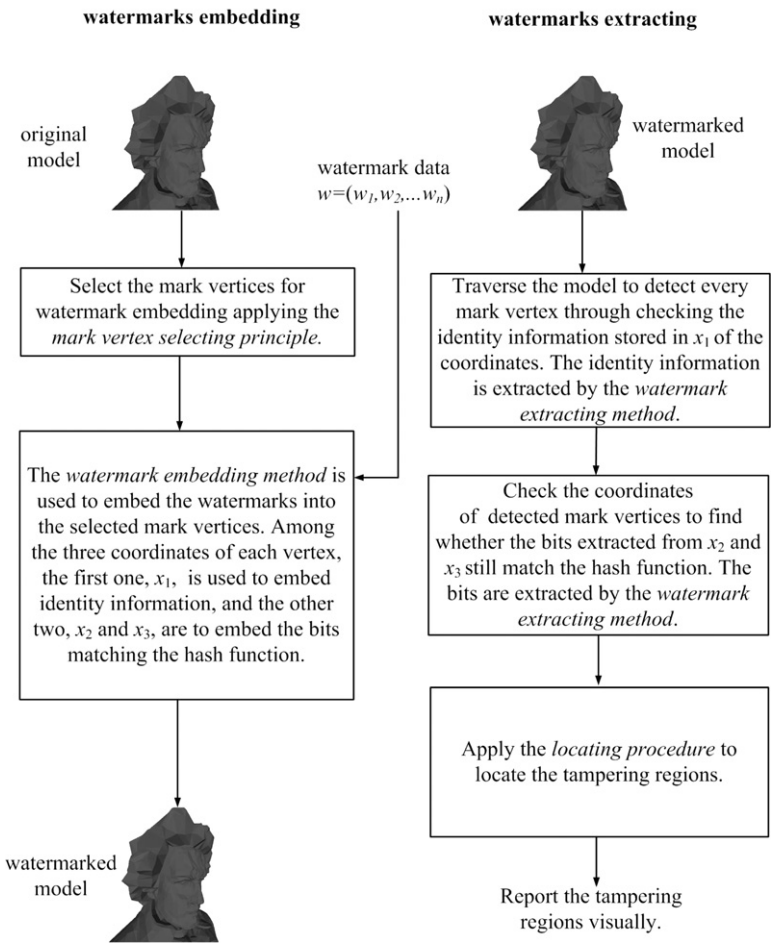


Fig. 5. Overview of this fragile watermarking scheme.

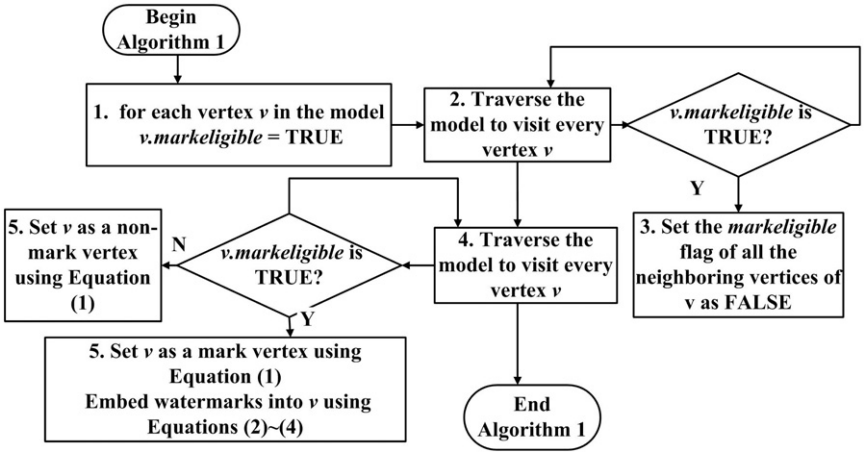


Fig. 6. The flowchart of Algorithm 1.

model following the *mark vertex selecting principle*. After that, the watermark embedding part modifies the coordinates of the vertices according to the *watermarks embedding method*. Finally it generates the watermarked models. In the watermark extracting part, the scheme first uses the *watermarks extracting method* to label the identity of every vertex. The next step is extracting the embedded bits from the mark vertices. In the end, the watermark extracting part applies the *locating procedure* to locate the tampering regions and reports them visually. The flowcharts of the two parts are shown in Figs. 6 and 7.

Fig. 6 is the flowchart of Algorithm 1. Algorithm 1 can be roughly divided into five steps. In Step 1, the *markeligible* flag of each vertex will be set as TRUE, indicating at the beginning every vertex is eligible as a mark vertex. In Steps 2 and 3, the algorithm traverses the model and selects the mark vertices according to the *mark vertex selecting principle*. In Steps 4 and 5, the algorithm uses the *watermarks embedding method* to embed the watermarks into the mark vertices. Eq. (1) is used to modify the first coordinate of each vertex in order to embed the corresponding identity information.

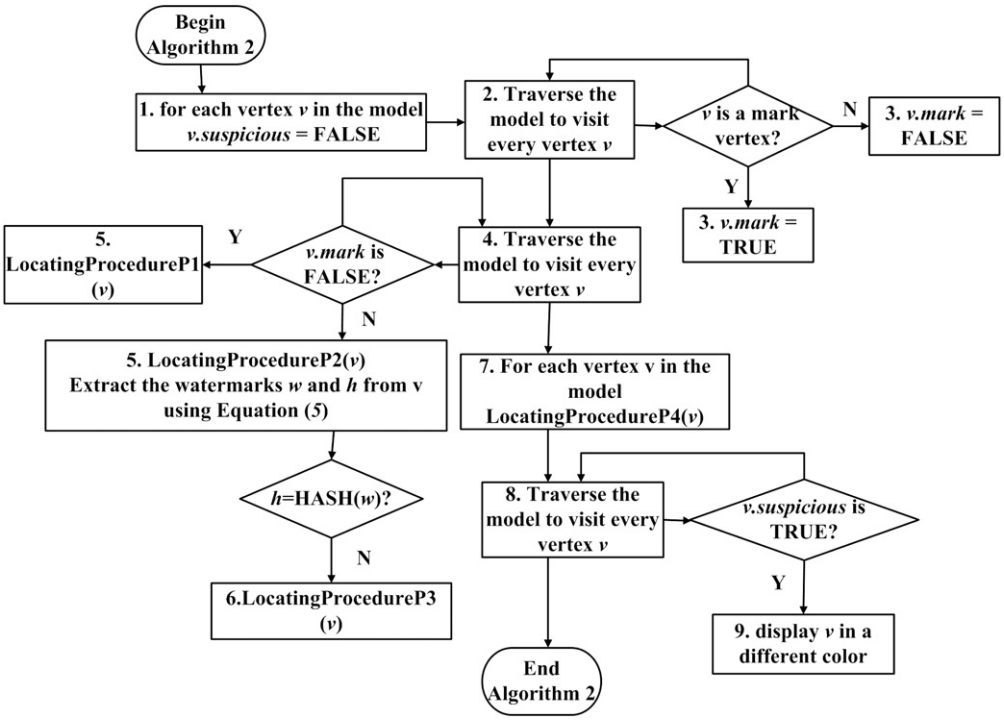


Fig. 7. The flowchart of Algorithm 2.

Eqs. (2)–(4) are to modify the other two coordinates to embed the watermark bits satisfying the predefined relationship.

Fig. 7 is the flowchart of Algorithm 2. Algorithm 2 can be roughly divided into nine steps. In Step 1, the *suspicious* flag of each vertex will be set as FALSE. In Steps 2 and 3, the algorithm traverses the model to find the mark vertices using the *watermarks extracting method*. The *mark* flag of the mark vertices will be set as TRUE. In Steps 4–7, the algorithm traverses the model twice and runs several auxiliary functions to detect the attacks following the *locating procedure*. And finally in Steps 8 and 9, the algorithm displays the suspicious attacked vertices in a different color to report the modified regions. Five auxiliary functions appear in the flowchart of Algorithm 2, which are LocatingProcedureP1 (v), LocatingProcedureP2 (v), LocatingProcedureP3 (v), LocatingProcedureP4 (v) and HASH (w). The functions LocatingProcedureP 1–4 will locate the tampering regions following the instructions from P1 to P4, mentioned in Section 3.3. The function HASH (w) will calculate the hash value of w according to the predefined hash function.

In Algorithms 1 and 2, the scheme needs to gather the information from the 1-ring vertices, such as checking if there is any mark vertex in the neighborhood or computing the centroid of the neighboring vertices. In the mesh model, the neighboring vertices of a vertex is much less than the vertices in the model. Thus, the time consumed on visiting the neighboring vertices could be viewed as a constant factor comparing with visiting the whole model. Then we can conclude that the computation complexities of Algorithms 1 and 2 are both $O(N)$, where N is the number of vertices in the mesh model.

A more detailed version of the two algorithms is provided in the Appendix A.

4. Performance discussion and experimental results

4.1. Optimization of the parameters

The fragile watermarking should be both perceptual invisible and able to detect the slightest attacks [10,11]. In this scheme, the embedding parameter *mask* determines the performance of the scheme on these two properties. The scheme sets the binary form of *mask* to determine the number of bits to be modified in the mantissa part and their positions. The proposed scheme could detect the attacks which change the bits in the valid positions. Thus, more 1s bits in *mask* increase the probability of the attacks being found. On the other hand, watermarks change the bits in the valid positions. Therefore, the more 1s bits in *mask*, the more distortions on the appearance of the model. The positions of the 1s bits could also affect both the safety and the distortion ratio issues. It is less likely for attacks to modify the low order of the mantissa part. Otherwise this will result in a small amount of modification and makes no sense. So arranging the 1s bits on the high order of the binary format of *mask* is suitable for detecting attacks. However, this will distort the original value a lot. Therefore it is recommended to apply more 1s bits in *mask* and put them in the middle of the binary format of *mask*.

Another purpose of *mask* is applied as a secret key in the extracting stage. Even if the attackers become aware of our scheme, their attacks still have a low probability of not destroying the embedded watermarks without the accurate information about the positions and the number of the watermarks. Therefore, a proper number of the 1s bits in *mask* could increase the difficulty for the attackers.

There also exist some other parameters in the scheme, such as key_i and a hash function. The hash function is a kind of the predefined relationships and is not directly related with the distortion ratio. The parameter *key* is an auxiliary parameter for *mask*. Thus selecting a proper *mask* is the key to balance the perceptual invisibility and the detecting ability. For the ease of discussion, we set $mask_1 = mask_2 = mask_3 = mask$, and a simple function $h(w) = w$ is applied.

We use the formula in [34] to measure the distortion induced by watermarking,

$$d(M, M') = \frac{1}{|M|} \sum_{i=1}^{|M|} \frac{|v_i - v_i'|}{|v_i|},$$

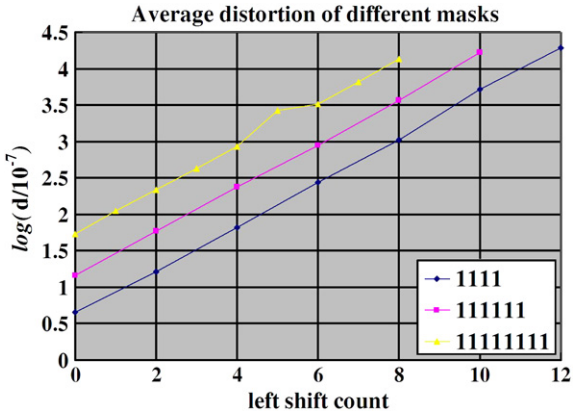


Fig. 8. Distortion curves of different embedding parameters.

where v_i and v_i' are the corresponding vertices in the original model(M) and watermarked model(M'). $|v_i - v_i'|$ is the Euclidean distance between these two vertices.

Fig. 8 gives the experimental curves between *mask* and the distortion ratio. The pattern in Fig. 8 means that *mask* have some consequent 1s bits in its binary form. The horizontal axis indicates shifting the pattern left to form *mask*. For instance, the curve labeled 111111 with 2 in the horizontal axis means that the parameter *mask* is equal to 252('11111100' in the binary form). The vertical axis represents the value of distortion ratio(d), which has been pre-processed by the log function. This figure could be used as the parameter selection guideline for the users. The data from Fig. 8 are gathered from the average distortion ratio of embedding on several different models with different precisions. The detailed information about these models and their precisions are provided in Table 2.

If users intend to control the distortion ratio under 0.01%, which is 3 in axis Y, they could choose 4080, 4032 or 3840(111111110000, 111111000000 or 111100000000 in the binary form respectively) as *mask* via looking up Fig. 8. There are eight valid bits in the binary form of 4080. It has a higher probability to detect the attacks. Thus it is the recommended parameter among the three given parameters. From the discussion above, our scheme can control the watermark intensity through selecting the parameters. It is very important, especially for CAD applications.

4.2. Discussion on detecting attacks

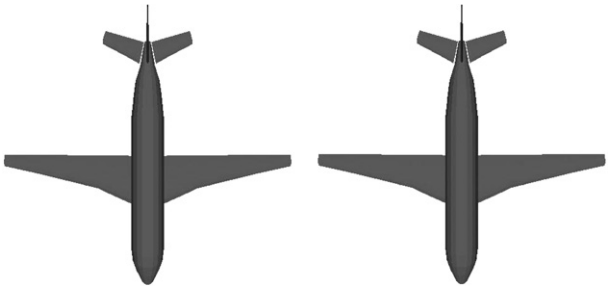
In this section we discuss the performance of this scheme on detecting the modifications on the model. The modifications mentioned in this section include adding noise, adding/deleting faces, inserting/removing vertices, remeshing the model, renumbering vertices, and translating/rotating/scaling the model. These modifications are common in practical use and widely used in the former literature to test the performance of the watermarking schemes [10,27,29,34,37]. As discussed later, the proposed scheme in this paper can detect these kinds of modifications on the models. We need to point out that this scheme is immune to certain incidental data operations. These operations do not destroy the integrity of the model, such as re-numbering vertices. Translating/rotating/scaling the model are also not breaking the integrity of the model. The users may still use the model if it is transformed. Ideally the fragile watermarking scheme could provide the convenience to filter these operations. However, our scheme can only view them as kinds of attacks. It needs further work to improve the scheme.

Adding noise: The attack adding noise only modifies the coordinates of vertices and does not modify the topological relation. The coordinates in the model are represented as (fx_1, fx_2, fx_3). fx_1 is used to embed the identity information and the other two are used to embed the watermark bits. If fx_2 and fx_3 are modified, the extracted bits will not hold the hash relationship. In P3 of the *Locating procedure* the attacks can be detected. Three different situations may happen when fx_1 is modified. One is that a vertex v is changed from a mark vertex to a non-mark vertex. This can be detected in P1 of the *locating procedure* since v is a non-mark vertex and there is no mark vertex in v's neighboring vertices. The second situation is that a vertex v is changed from a non-mark vertex to a mark vertex and this will result in two adjacent mark vertices. P2 of the *locating procedure* can detect this situation. The third situation is that the vertices keep their identity. It is hard for a mark vertex to keep its identity when it is attacked since the modified fx_1 can hardly keep equal to *key*. If there are many 1s bits in the binary form of *mask*, as discussed in Section 4.1, the probability that the modified fx_1 of a mark vertex remains equal to *key* can be reduced. If a non-mark vertex v is modified but remains its identity, the attacks still could be found in P3 of the *locating procedure*. It is because fx_1 of non-mark vertices are used to calculate the centroid of the neighboring of the mark vertices in S2 of the *watermarks embedding method*. Modifying fx_1 will result in that the extracted bits do not hold the hash function.

Adding/deleting faces and inserting/removing vertices: These attacks modify the topological relation of the model. Since adding or deleting the faces is implemented through inserting or removing vertices and modifying the topological relation accordingly, we only discuss inserting/removing vertices. There exist two situations when inserting a vertex. If a mark vertices v is inserted, which means that the mantissa part of fx_1 of v matches Eq. (1) coincidentally, it will be adjacent to several vertices. Two situations arise here. One is that there are mark vertices among these adjacent vertices and then v could be detected in P2 of the *locating procedure*. The other is that no mark vertex exists among these vertices. In this situation the extracted bits from v do not keep the hash function. And it could be detected in P3 of the *locating procedure*. When inserting a non-mark vertex, two different situations also arise. If there is no mark vertex in the neighborhood of v, it could be detected in the P1 of the *locating procedure*. If there exists a mark vertex in the neighboring vertices of v, denoted as v', inserting v will change the centroid of the neighborhood of v'. Therefore inserting v is found. The situation of removing a vertex is similar to that of inserting a vertex. Removing a mark vertex will form a region without any mark vertex and it could be found in P1 of the *locating procedure*. And removing a non-mark vertex v will change the centroid of the neighborhood of the mark vertex v adjacent and v is found in P3 of the *locating procedure*.

Remeshing the model: Remeshing or retriangulating the model heavily changes the connectivity information. This kind of attacks alters the neighboring vertices set of the mark vertices. Subsequently it changes the centroid of the neighborhood of mark vertices. When extracting the watermarks, the extracted bits w and h from the mark vertices could not satisfy the predefined relationship. Thus the topology modifications around the mark vertices can be detected. The mark vertices are uniformly distributed in the 3D models in our scheme. As a result, either remeshing or re-triangulating part or even the whole watermarked models could be detected.

Re-numbering vertices: It depends on whether the topological relation is modified when viewing re-numbering the vertices as an attack or an incidental operation. If the re-numbering operation does not change the neighboring vertices of a vertex v, the operation is harmless. Our scheme does not detect this kind of operation since it does not change any centroid of the neighborhood of mark vertices. If the topological relation is



(a) *Airplane* precision: 10^{-3}

(b) *Apple* precision: 10^{-7}

Fig. 9. The parameters selection in [34] depends on the precisions of the models. The parameters for the two models are: $k_2^q = k_3^q = 0.01, k_2^d = k_3^d = 100, h(w_i) = w_i$.

modified, it changes the adjacent non-mark vertices of mark vertices. Consequently it changes the centroid of the neighborhood of mark vertices. Re-numbering the vertices can be found then.

Translating/rotating/scaling: These three operations transform the models. These transforming operations modify the coordinates of the vertex. Thus this situation is somewhat the same as adding noise. As discussed in the adding noise paragraph, our scheme could detect this kind of attacks.

Now that we have discussed the performance of our scheme, we want to emphasize that the key to achieve the declared performance is to select an appropriate embedding parameter. As discussed in Section 4.1, a good embedding parameter *mask* should have many 1s bits in the middle of the binary format of *mask*. The scheme may fail to detect the attacks if there are few 1s bits in the binary format of *mask* or the 1s bits are arranged in the low order of *mask*.

4.3. Discussion of the convenience of implementation and usage

The schemes in [10,29] both need to perturb the mark vertices until they match the predefined relationship. However, in [10], the perturbing procedure is through a trial and error method. It is not convenient to implement and the induced distortion could not be controlled. In [29], they first calculate a region around the original position. Moving the vertex to the positions in this region does not lead to an unacceptable distortion. Then it tests every place in this region until it finds a position where the relationship can be satisfied. It has two drawbacks. The first is that it is still embedding through a trial and error method. And the second is that if no position in this region could satisfy the relationship, it has to abandon embedding on this vertex, which leaves an embedding hole. Attacks on these holes could not be detected by the scheme in [29]. In our scheme, the perturbed position could be generated from Eq. (4) directly. It is easy to implement and leaves no embedding hole. Also, it could control the distortion ratio through selecting a proper *mask*.

As for using the scheme, the scheme in [34] has to select different embedding parameters according to the different precisions of the models. Fig. 9 (a) and (b) are two models with different precisions. The left part of the graphs is the original models and the right part of the graphs is the watermarked models. The parameters are

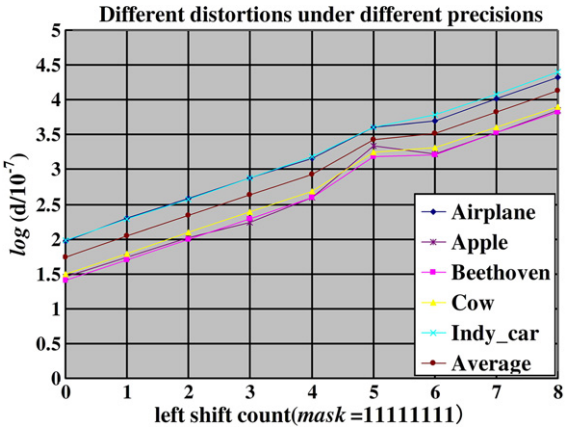
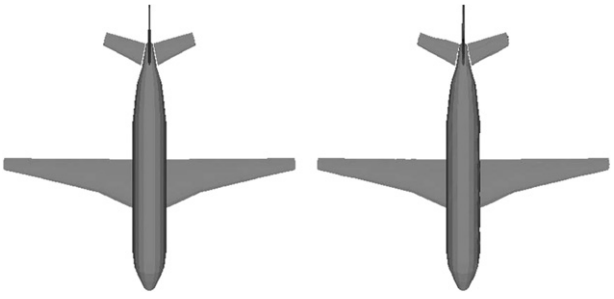
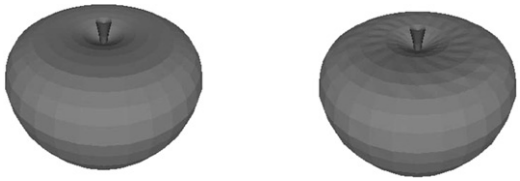


Fig. 10. The distortion curves of the models with different precisions.



(a) *Airplane* precision: 10^{-3}



(b) *Apple* precision: 10^{-7}

Fig. 11. The parameters for the two models are: $mask_1 = mask_2 = mask_3 = 65280, h(w_i) = w_i$.

the same, but the visual effects of the watermarked models are remarkably different. Because the two models have different precisions, users have to select parameters according to the precisions of the models. It is not convenient in practical use. In our scheme, the watermarks are embedded into the mantissa part. Modifying the same position of the mantissa part will result in different distortion ratios because of the different exponent part. The amount of distortions on the models with low precision is larger than that on the high precision models. Thus the parameters in our scheme are adaptive to the precisions of the models.

Fig. 10 illustrates the distortion ratios of several models with different precisions. The embedding parameters are the same. From Fig. 10 we can find that the curves are close to the average line while there is a significant difference of the magnitude in the precisions of models. We embed watermarks into two models, whose curves in Fig. 10 vary most, *airplane* and *apple*, with the same parameters. Fig. 11 shows that the same embedding parameters results in the same visual effects for the two watermarked models. Thus we can conclude that in our scheme the same parameters results in a close distortion ratio regardless of the precisions. The advantage of this property is that users only need to look up the average distortion curve in Fig. 8 to select a suitable *mask* once. The adaptive property eases the use of our scheme.

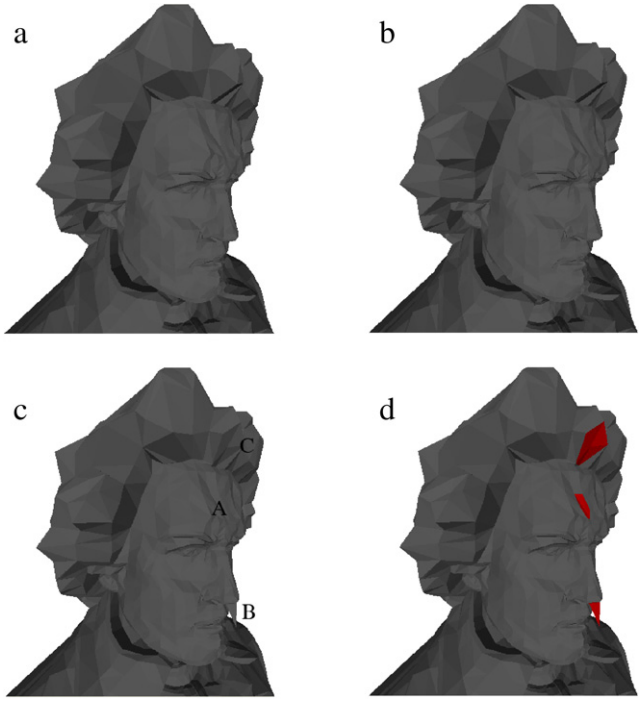


Fig. 12. One example of attacks detection for 3D models using our scheme. (a) The original *Beethoven* model. (b) The watermarked model, embedding parameter=4080. (c) Several attacks on the model. Label A denotes the region with 0.01% noise added. Label B denotes the region with an extra vertex inserted, and label C denotes the region with a vertex deleted. (d) Our scheme accurately locates these attacks visually.

4.4. Experimental results

We conduct some experiments on several mesh models, which have different precisions, to test the performance of the proposed fragile watermarking scheme. Table 2 gives the detailed information about the mesh models used in our experiment. The data structure of the coordinates is IEEE-754 float32 [43].

From Table 2 we can find that our scheme selects around 20%–31% of the vertices as mark vertices. Some existing schemes use the correlation value between the embedded watermarks and extracted watermarks to demonstrate the authentication results [10,27]. However, Lin [29] suggests that different attacks may have different intensity and it is hard to find a unique threshold to figure out whether the model is attacked. For example, modifying a vertex of a large model changes the correlation value little. Besides, visually locating the tampering region is a useful property [10,29, 34]. Thus, our scheme shows the tampering regions visually.

Fig. 12 illustrates that our scheme accurately detects and locates several kinds of attacks simultaneously. Fig. 12 (a) shows the original *Beethoven* model. The model has 2521 vertices and 5042 faces; 694 vertices are selected as the mark vertices. Fig. 12 (b) is the same view of the watermarked model, which is visually identical with the original model. The watermarked model in Fig. 12 (c) has been attacked by various operations illegally: invisible noise added, extra vertex added and one vertex being deleted. These regions are labeled as 'A', 'B' and 'C' respectively. Fig. 12 (d) illustrates the located suspicious regions in red. From Fig. 12 (d) we can find that the regions in red are exactly where the tampering operations happen. The experimental results verify the accuracy of our locating procedure.

5. Conclusions and future work

The main contributions of this paper are listed below.

- This paper is the first paper, to the best of our knowledge, to report and analyze the numerically instable problem in the 3D fragile watermarking field.
- This paper provides a numerically stable fragile watermarking scheme. The scheme represents the mantissa part of the floating-point number as an unsigned integer. After that, the scheme operates the integer by the bit XOR operator to embed the watermarks. It is numerically stable since it only applies the integral and the bit arithmetics.
- Furthermore, this scheme eliminates the causality problem through forbidding the mark vertices to be adjacent. Meanwhile, this method does not lose the detecting ability.
- Compared with other algorithms, this scheme is easy to implement and use, as discussed in Section 4.3. This scheme can control the watermarking intensity, which is important for CAD applications.
- This paper discusses the performance of this scheme on detecting several possible modifications. It can detect several attacks which happen simultaneously, such as adding noise, adding/deleting faces and remeshing the model, as discussed in Section 4.2. Moreover, the scheme is immune to some incidental operations, such as re-numbering the vertices. The experimental results agree with our discussion.

There are two main drawbacks in our scheme. First, it may fail to detect the attacks if the embedding parameters are not well selected, as discussed in Section 4.1. Second, it can not filter some harmless operations, such as transforming the model. Our future research will focus on increasing the convenience of the scheme to be immune to these harmless operations.

Acknowledgements

The research was supported by Chinese 973 Program (2004CB7 19400), and the National Science Foundation of China (60533070, 60625202, 90715043). The third author was supported by the project sponsored by a Foundation for the Author of National Excellent Doctoral Dissertation of PR China (200342), and a Program for New Century Excellent Talents in University(NCET-04-0088).

Appendix. The pseudo codes of Algorithms 1 and 2

Each vertex v in Algorithm 1 has one flag, *markeligible*, indicating whether this vertex is eligible to be selected as a mark vertex. M indicates the 3D model. $N(v)$ is the set of the neighboring vertices of v.

Each vertex v in Algorithm 2 has two flags, *mark* and *suspicious*. The flag *mark* shows if the vertex is a mark vertex and *suspicous* denotes if the vertex is a suspicious attacked vertex. M and $N(v)$ are for the 3D model and the neighborhood of v respectively. The variable *HasMarkVertexNeighbors* will be set to FALSE if there is no mark vertex in the neighboring set of v.

ALGORITHM 1: *Mark vertex selection and watermarks-embedding*

```
1  for each vertex v ∈ M
2      do v.markeligible ← TRUE
3  End for
4  for each vertex v ∈ M
5      do if v.markeligible = TRUE
6          then for each vertex v_n ∈ N(v)
7                  do v_n.markeligible ← FALSE
8              End for
9      End if
10 End for
```

```
for each vertex v ∈ M
    do if v.markeligible = TRUE
        then Set v as a mark vertex using Eq. (1)
             Embed w into v using Eqs. (2)–(4)
        else Set v as a non-mark vertex using
             Eq. (1)
      End if
End for
```

ALGORITHM 2: *Watermarks-extracting and tampering locating*

```
for each vertex v ∈ M
    do v.suspicious ← FALSE
       if v is a mark vertex
         then v.mark ← TRUE
         else v.mark ← FALSE
       End if-else
End for
for each vertex v ∈ M
    do if v.mark = FALSE
         then HasMarkVertexNeighbors ← FALSE
              for each vertex v_n ∈ N(v)
                  do if v_n.mark = TRUE
                       then HasMarkVertexNeighbors
                            ← TRUE
                  End if
              End for
              if HasMarkVertexNeighbors = FALSE
                then v.suspicious ← TRUE
                     for each vertex v_n ∈ N(v)
                         do v_n.suspicious ← TRUE
                     End for
              End if
         else HasMarkVertexNeighbors ← FALSE
              S ← ∅
              for each vertex v_n ∈ N(v)
                  do if v_n.mark = TRUE
                       then HasMarkVertexNeighbors
                            ← TRUE
                            put v_n in S
                  End if
              End for
              if HasMarkVertexNeighbors = TRUE
                then v.suspicious ← TRUE
                     for each vertex v_s ∈ S
                         do v_s.suspicious ← TRUE
                     End for
              End if
              if v.suspicious ≠ TRUE
                then Extract the watermarks w and h
                     using Equation (5)
                     if h ≠ hash(w)
                       then v.suspicious ← TRUE
                            for each vertex v_n ∈ N(v)
                                do v_n.suspicious
                                   ← TRUE
                            End for
                     End if
              End if
       End if-else
End for
for each vertex v ∈ M
    do if v.mark = TRUE && v.suspicious = FALSE
         then for each vertex v_n ∈ N(v)
                  do v_n.suspicious ← FALSE
              End for
End if
End for
for each vertex v ∈ M
    do if v.suspicious = TRUE
      then display v in a different color
End if
End for
```

References

[1] Cox IJ, Kilian J, Leighton T, Shamoon T. Secure spread spectrum watermarking for images, audio, and video. In: IEEE International Conference on Image Processing. 1996.

[2] Podilchuk CI, Zeng WJ. Image-adaptive watermarking using visual models. IEEE Journal on Selected Areas in Communications 1998;16(4):525–39.

[3] Hsiao JH, Chen CS, Chien LF, Chen MS. A new approach to image copy detection based on extended feature sets. IEEE Transactions on Image Processing 2007; 16(8):2069–79.

[4] Hartung F, Girod B. Watermarking of uncompressed and compressed video. Signal Processing 1998;66(3):283–301.

[5] Wang HX, Lu ZM, Pan JS, Sun SH. Robust blind video watermarking with adaptive embedding mechanism. International Journal of innovative Computing Information and Control 2005;1(2):247–59.

[6] Swanson MD, Zhu B, Tewfik AH, Boney L. Robust audio watermarking using perceptual masking. Signal Processing 1998;66(3):337–55.

[7] Wang XY, Zhao H. A novel synchronization invariant audio watermarking scheme based on DWT and DCT. IEEE Transactions on Signal Processing 2006; 54(12):4835–40.

[8] Ohbuchi R, Masuda H, Aono M. Watermarking three-dimensional polygonal models through geometric and topological modifications. IEEE Journal on Selected Areas in Communications 1998;16(4):551–60.

[9] Kanai S, Date H, Kishinami T. Digital watermarking for 3-D polygons using multiresolution wavelet decomposition. In: Proceedings 6th IFIP WG 5.2 GEO-6. 1998.

[10] Yeo BL, Yeung MM. Watermarking 3D objects for verification. IEEE Computer Graphics & Application 1999;19(1):727–35.

[11] Benedens O. Geometry-based watermarking of 3-D models. IEEE Computer Graphics & Application 1999;19(1):46–55.

[12] Praun E, Hoppe H, Finkelstein A. Robust mesh watermarking. In: Proceedings of the 26th annual Conference on Computer Graphics and Interactive Techniques. 1999.

[13] Benedens O. Affine invariant watermarks for 3-D polygonal and NURBS based models. In: Proceedings of Information Security Workshop. 2000.

[14] Wagner MG. Robust watermarking of polygonal meshes. In: Proceedings Geometric Modeling and Processing. 2000.

[15] Fornaro C, Sanna A. Public key watermarking for authentication of CSG models. Computer-Aided Design 2000;32(12):727–35.

[16] Lin CY, Chang SF. A robust image authentication method distinguishing JPEG compression from malicious manipulation. IEEE Transactions on Circuits and Systems for Video Technology 2001;11(2):153–68.

[17] Yin KK, Pan ZG, Shi JY. Robust mesh watermarking based on multiresolution processing. Computers & Graphics 2001;25(3):409–20.

[18] Kankanhalli MS, Chang EC, Guan X, Huang Z, Wu Y. Authentication of volume data using wavelet-based foveation. In: Proceedings 6th Eurograhphics Workshop on Multimedia. 2002.

[19] Ohbuchi R, Takahashi S, Miyazawa T, Mukaiyama A. Watermarking 3D polygonal meshes in the mesh spectral domain. In: Proceedings of the Graphics Interface. 2001.

[20] Ohbuchi R, Mukaiyama A, Takahashi S. A frequency-domain approach to watermarking 3-D shapes. Computer Graphics forum 2002;21(3):373–82.

[21] Cayre F, Macq B. Data hiding on 3-D triangle meshes. IEEE Transactions on Signal Process 2003;51(4):939–49.

[22] Cayre F, Rondao-Alface P, Schmitt F, et al. Application of spectral decomposition to compression and watermarking of 3D triangle mesh geometry. Signal Processing-image Communication 2003;18(4):309–19.

[23] Yu ZQ, Ip HHS, Kwok LF. A robust watermarking scheme for 3D triangular mesh models. Pattern Recognition 2003;36(11):2603–14.

[24] Jang BJ, Moon KS, Huh Y, et al. A new digital watermarking for architectural design drawing using LINEs and ARCs based on vertex. Lecture notes in computer science, vol. 2939. 2004. p. 544–57.

[25] Li L, Zhang D, Pan ZG, et al. Watermarking 3D mesh by spherical parameterization. Computers & Graphics-UK 2004;28(6):981–9.

[26] Cho WH, Lee ME, Lim H, Park SY. Watermarking technique for authentication of 3-D polygonal meshes. Lecture notes in computer science, vol. 3304. 2005. p. 259–70.

[27] Wu HT, Cheung YM. A fragile watermarking scheme for 3D meshes. In: Proceeding the 7th Workshop on Multimedia and Security. 2005.

[28] Alface PR, Macq B. Feature-based watermarking of 3d objects towards robustness against remeshing and de-synchronization. In: Proceedings of the SPIEIS and T Electronic Imaging. 2005.

[29] Lin HYS, Liao HYM, Lu CS, Lin JC. Fragile watermarking for authenticating 3-D polygonal meshes. IEEE Transactions on Multimedia 2005;7(6):997–1006.

[30] Piegl LA. Ten challenges in computer-aided design. Computer-Aided Design 2005;37(4):461–70.
[31] Zafeiriou S, Tefas A, Pitas I. Blind robust watermarking schemes for copyright protection of 3D mesh objects. IEEE Transactions on Visualization and Computer Graphics 2005;11(5):596–607.
[32] Cho JW, Kim MS, Prost R, et al. Robust watermarking on polygonal meshes using distribution of vertex norms. Lecture notes in computer science, vol. 3304. 2005. p. 283–93.
[33] Wu JH, Kobbelt L. Efficient spectral watermarking of large meshes with orthogonal basis functions. Visual Computer 2005;21(8–10):848–57.
[34] Chou CM, Tseng DC. A public fragile watermarking scheme for 3D model authentication. Computer-Aided Design 2006;38(11):1154–65.
[35] Bors AG. Watermarking mesh-based representations of 3-D objects using local moments. IEEE Transactions on Image Processing 2006;15(3):687–701.
[36] Kwon KR, Lee SH, Lee EJ, et al. Watermarking for 3D CAD drawings based on three components. Lecture notes in computer science, vol. 4109. 2006. p. 217–25.
[37] Cheung YM, Wu HT. A sequential quantization strategy for data embedding and integrity verification. IEEE Transactions on Circuits and Systems for Viedo Technology 2007;17(8):1007–116.
[38] Lavoue G, Denis F, Dupont F. Subdivision surface watermarking. Computers & Graphics-UK 2007;31(3):480–92.
[39] Lee SH, Kwon KR. A watermarking for 3D mesh using the patch CEGIs. Digital Signal Processing 2007;17(2):396–413.
[40] Cho JW, Prost R, Jung HY. An oblivious watermarking for 3-D polygonal meshes using distribution of vertex norms. IEEE Transactions on Signal Processing 2007;55(1):142–55.
[41] Cheng YM, Wang CM. An adaptive steganographic algorithm for 3D polygonal meshes. Visual Computer 2007;23(9–11):721–32.
[42] Lee SH, Kwon KR. Mesh watermarking based projection onto two convex sets. Multimedia Systems 2008;13(5–6):323–30.
[43] IEEE standard for binary floating-point arithmetic. ANSI/IEEE Standard 754-1985 1985.