A New Watermarking Method for 3D Models Based on Integral Invariants

Yu-Ping Wang and Shi-Min Hu, *Member*, *IEEE*

Abstract—In this paper, we propose a new semifragile watermarking algorithm for the authentication of 3D models based on integral invariants. A watermark image is embedded by modifying the integral invariants of some of the vertices. In order to modify the integral invariants, the positions of a vertex and its neighbors are shifted. To extract the watermark, all the vertices are tested for the embedded information, and this information is combined to recover the watermark image. The number of parts of the watermark image that can be recovered will determine the authentication decision. Experimental tests show that this method is robust against normal use modifications introduced by rigid transformations, format conversions, rounding errors, etc., and can be used to test for malicious attacks such as mesh editing and cropping. An additional contribution of this paper is a new algorithm for computing two kinds of integral invariants.

Index Terms—Semifragile watermark, 3D model watermarking, integral invariants.

1 Introduction

Digital watermarking has been studied over many years for digital content copyright protection and authentication. As 3D models are used in a wide variety of fields, the necessity to protect their copyrights becomes crucial.

The first article on 3D model watermarking was published in 1997 by Ohbuchi et al. [3]. Several years passed, and a lot of new algorithms were developed. Theoretically, there are two categories of watermarking algorithms: spatial-domain methods and frequency-domain methods. Spatial-domain methods embed the watermark by directly modifying the positions of vertices, the colors of texture points or other elements representing the model. The frequency-domain methods embed the watermark by modifying the transform coefficients.

There is no unified standard to test which algorithm is better. There are some applications where one method is found to be better suited than another. Nevertheless, watermarking algorithms are usually characterized by the four following properties:

- *Validation.* The watermark can be fully extracted from the unattacked model.
- *Invisibility.* Watermarked models should look similar to the original model.
- *Capacity.* This corresponds to the amount of information that can be embedded in the models.
- *Robustness.* The watermark should survive different types of attacks.

Spatial-domain algorithms work on certain 3D model invariants like Triangle Similarity Quadruple (TSQ), Tetrahedral Volume Ratio (TVR) [3], [4], [5], [6], Affine Invariant Embedding (AIE) [7], [8], etc., to embed the watermark. But most of them are very sensitive to noise. Most frequency-domain algorithms provide better robustness and use wavelet analysis [9], [10], Laplace transforms [11], [12], and other such transforms [14], [15], [16]. Nevertheless, the 3D model distortions making up the watermark fail to be invisible, or extraction routines require the original watermarked model in order to obtain hidden information.

Robust watermarking of 3D models has been widely researched in recent years, and great developments have been achieved in both frequency-domain algorithms [17] and spatial-domain algorithms [18]. Although the problems have been well defined by researchers working on image watermarking, there has been little research into fragile watermarking until recently [13], [19], [20]. This kind of watermarking scheme focuses on finding where and how the models have been modified or attacked. For many applications, this watermarking scheme is often too restrictive to be usable, as model compression and format conversion are not permitted. It is desired that the hidden data be robust to unintentional changes like model compression, rigid transformation, and random noise originating from format conversion. Similar to image watermarking nomenclature [21], we characterize watermarking algorithms showing this property as semifragile.

In this paper, we propose a new semifragile spatial-domain watermarking algorithm for the authentication of 3D models based on integral invariants. It can survive under rigid transforms and certain noise attacks. Our idea mainly comes from other spatial-domain algorithms like [3], [7], and [13]. Since we wish to embed the watermark with geometrical invariants, we use integral invariants to achieve this. Based on the good character of integral invariants that have proven useful in parameterization [22], registration [23], and classification [24] applications, we believe that

- *The authors are with the Tsinghua National Laboratory for Information Science and Technology and the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, P.R. China. E-mail: wyp05@mails.tsinghua.edu.cn, shimin@tsinghua.edu.cn.*

Manuscript received 16 June 2007; revised 31 Oct. 2007; accepted 11 Aug. 2008; published online 15 Aug. 2008.
Recommended for acceptance by L. De Floriani.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number TVCG-2007-06-0070.
Digital Object Identifier no. 10.1109/TVCG.2008.101.

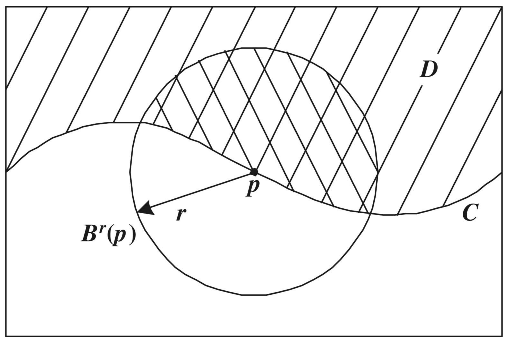


Fig. 1. Area invariant for planar curves $A^r(p)$ is the area of the cross-hatched part, which is the intersection of the circular disk $B^r(p)$ and domain D.

they can also be used in our problem. With each vertex that will undergo watermarking, the integral invariants of the unwatermarked model are calculated. These invariants are then slightly changed to embed the watermark image parts. In order to change the integral invariants to the new value, the positions of the vertex and its neighbors are modified. The extraction routine is the inverse process of the insertion routine. All the vertices are traversed, computing the integral invariants and trying to match the embedded information. Once matched, the embedded information is extracted at each vertex from the two integral invariants, and this data is combined to form the extracted watermark. By analyzing the false-positive probability, the final authentication decision can be made.

Note that it is needed to compute the integral invariants of some of vertices once in the insertion procedure, and the integral invariants of all the vertices once in the extraction procedure. In practice, we find that the previous algorithm given in [2] to compute the integral invariants is inefficient for this application. Therefore, we have developed a faster algorithm for computing integral invariants.

The structure of this paper is organized as follows: In Section 2, we briefly introduce integral invariants theory for 3D models and provide a new algorithm for computing two kinds of invariants used in our watermarking method. In Section 3, we explain the watermark insertion and extraction algorithms in detail. In Section 4, we show some of our experimental results. Finally, in Section 5, we conclude and mention potential improvement in future work.

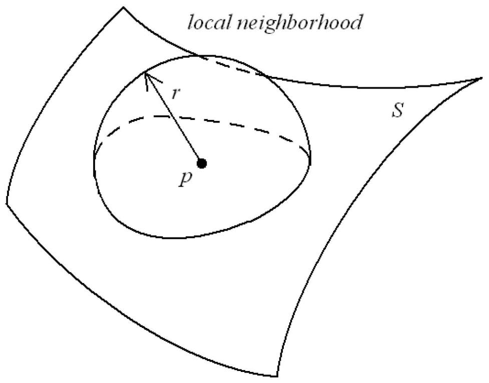


Fig. 2. Local neighborhood in R^3 is a ball $B^r(p)$ around p.

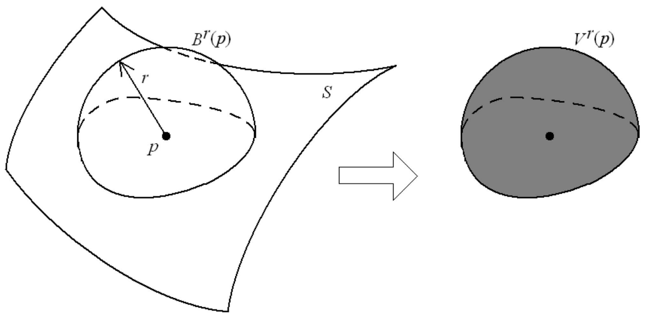


Fig. 3. Volume invariant in R^3 is the volume of $B^r(p)$ intersected with domain D.

2 Integral Invariants

2.1 The Concept of Integral Invariants

The concepts of integral invariants were first introduced by Manay et al. [1]. They studied integral invariants for curves in a plane. An example of such an invariant is the area invariant. It is suitable for estimating the curvature of a curve C at a point p, where C is assumed to be the boundary of a planar domain D (see Fig. 1). Consider the circular disk $B^r(p)$ of radius r, centered at p, and compute the area $A^r(p)$ of its intersection with the domain D. This is obviously a way to estimate curvature on a scale defined by the kernel radius r. Manay et al. show the superior performance of this and other integral invariants on noisy data, especially for the reliable retrieval of shapes from geometric databases.

Pottmann et al. [2] extended the concept of integral invariants from R^2 to R^3. They introduced integral invariants for surfaces with the integral of a local neighborhood in 3D space (see Fig. 2). Here, we state two kinds of integral invariants, which will be used in the following sections.

If we extend the area invariant from R^2 to R^3, we get the volume invariant (see Fig. 3). Considering the local neighborhood as a ball $B^r(p)$ of radius r, centered at p, the volume invariant is defined as $V^r(p)$, which is the volume of $B^r(p)$ intersected with the domain D, which is the inner part of the model.

Similarly, considering the local neighborhood as a sphere $S^r(p)$ of radius r, centered at p, the area invariant (see Fig. 4) is defined as $SA^r(p)$, which is the area of $S^r(p)$ intersected with the domain D.

These kinds of integral invariants have been proved to be more robust against noise than traditional differential

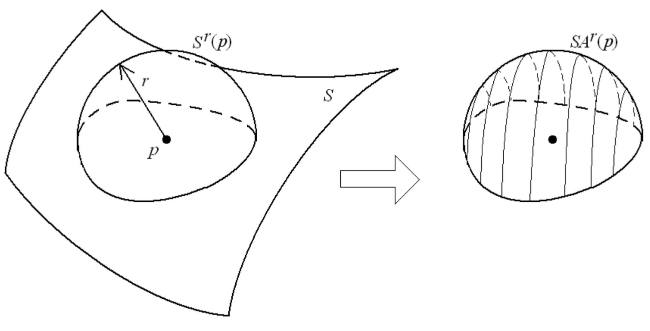


Fig. 4. Area invariant in R^3 is the surface area of $S^r(p)$ intersected with domain D.

invariants such as curvature [22], [23], [24]. Explicit proofs and experimental results can be found in [2].

2.2 A Faster Algorithm for Computing Integral Invariants

When integral invariants were extended into the R^3 case in [2], an algorithm for computing integral invariants was given. Its main idea is to discretize the area invariant and the volume invariant. For the area invariant $SA^r(p)$, a triangle-mesh model of a sphere with radius r is used as a discretization of the sphere's local neighborhood. The algorithm moves the center point of the sphere model to the vertex p and tests each vertex of the sphere model whether it is inside the domain D. The value of the area invariant is obtained by summing the area of the triangles inside domain D. For the volume invariant $V^r(p)$, the algorithm first represents the sphere model and domain D with 3D bitmaps where 1 means inside the sphere/domain and 0 means outside the sphere/domain. The volume invariant is then computed by the fast Fourier transform (FFT).

In this section we introduce a faster algorithm for computing integral invariants. For convenience, in the following section, we will suppose that invariants are computed with a local neighborhood of radius r, centered at vertex p.

The area invariant is computed first, and then, the volume invariant is computed based on the area invariant.

Area invariant. For a 3D mesh model, we assume that the sphere $S^r(p)$ can intersect the surface of the model with a set of arcs on $S^r(p)$. This is always true when the model is closed and when the radius is not large enough to let $B^r(p)$ include the whole model.

The purpose of the algorithm is to compute the area surrounded by this set of arcs. The two endpoints forming each arc are the intersection of $S^r(p)$ with edges of the model, which are easy to obtain. Then, an approximation of the area is computed by replacing each of these arcs with the great arc on $S^r(p)$ with the same endpoints. Therefore, the intersection surface becomes a polygon on the sphere. Note that this polygon can be disconnected or a polygon with holes. These cases should be treated specially as sum or subtraction of several simple polygons, which are connected and without holes. For convenience, the following sections will only discuss the simple polygon case.

The formula for computing the area of a spherical triangle is

$$S = (\alpha + \beta + \gamma - \pi) \cdot r^2, \tag{1}$$

where α, β, and γ are the three spherical angles of the spherical triangle. It is easy to extend this formula to any spherical polygon as

$$S = \left(\sum \alpha_i - (n-2)\pi\right) \cdot r^2, \tag{2}$$

where α_i is the spherical angle between two adjacent great arcs and n is the total number of great arcs.

Volume invariant. After we get the area invariant, we start to compute the volume invariant. Fig. 5 provides sketches of this computational procedure to help explain it. In Figs. 5a, 5b, and 5c are 2D sketches, and Fig. 5d is a 3D sketch. First, we multiply the area invariant by $r/3$, which is the volume of an irregular cone (the shaded part in Fig. 5a).

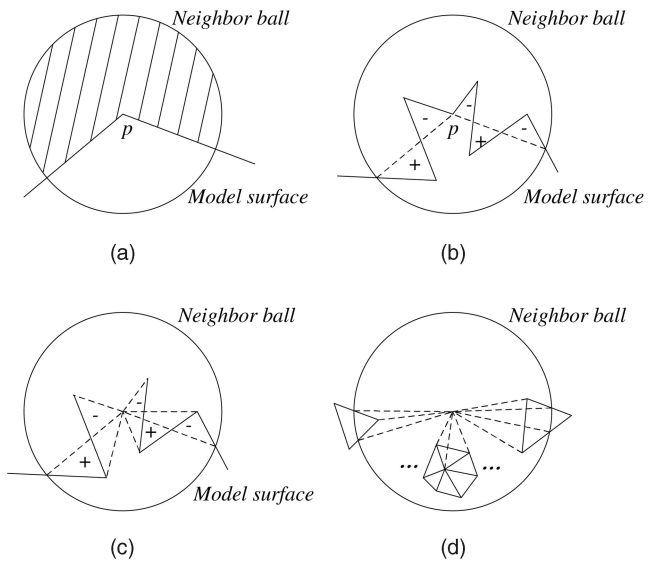


Fig. 5. Computing the volume invariant from the area invariant.

To compute the volume invariant, the volume of the irregular cone should be modified by adding or subtracting the parts labeled "+" or "−" in Fig. 5b. Directly determining these parts requires computationally expensive operations to find the intersections between the irregular cone and the model surface. Instead, the volume modification can be computed more easily by converting the problem to adding or subtracting the tetrahedron volumes (denoted as "+" or "−" in Fig. 5c) constructed with all the triangular facets inside $B^r(p)$ as base and p as apex.

Note that special processing will be needed near the boundary of the ball $B^r(p)$ (for convenience, later, this kind of balls are named as neighbor balls when point p is not designated). The volume of the outer tetrahedron part should be removed. There are two cases: one vertex of the triangular facet is outside the ball $B^r(p)$, or two vertices of the triangular facet are outside (the triangle at the right or left in Fig. 5d, together with the common case at the middle). Finally, we get the volume invariant.

The algorithm flow is shown in Fig. 6. By repeating this algorithm at each vertex of the model, we will generate the invariants for all the vertices.

2.3 Algorithm Analysis

The complexity of the algorithm can be derived from the analysis below.

In step 1 of the algorithm, we traverse all vertices around point p. If the number of vertices is m, this step costs $O(m)$ in time. m is proportional to the intersection surface area ($O(r^2)$) and inversely proportional to the average facet area ($O(l^2)$, where l is the average edge length). We conclude that $m = O((r/l)^2)$, so the time complexity in this step is $O((r/l)^2)$.

In steps 2 and 3 of the algorithm, we compute the intersection points, as well as the spherical angle formed by these points. Similar to the last step, the time complexity here is $O(r/l)$.

1) Find the vertices around point p, and divide them into 3 classes: inner, cross, and outer. Vertices of class inner are in the ball $B^r(p)$, and all of their direct neighbors (there is an edge between them) are in the ball. Vertices of class cross are in the ball $B^r(p)$, but at least one of their direct neighbors is out of the ball. Vertices of class outer are out of the ball $B^r(p)$, but at least one of their direct neighbors are in the ball.
2) For all the edges that have a vertex of class outer and a vertex of class cross, compute the intersect point of the edge and sphere $S^r(p)$. Note that these points construct a "circle", and we name the assembly of these points as $P^r(p)$.
3) For each point in $P^r(p)$, compute its spherical angle between its two adjacent points, and sum them as $AG^r(p)$.
4) Compute $SA^r(p) = AG^r(p) - (n-2)\pi$, where n is the number of points in $P^r(p)$.
5) For each facet whose three vertices are of class inner or cross, compute the volume of the tetrahedron constructed by the facet and point p, and sum them as $VI^r(p)$. Note that the volume can be negative.
6) For each facet that crosses the sphere $S^r(p)$, compute the volume of the inner part of the tetrahedron constructed by the facet and point p, and sum them as $VO^r(p)$. Note that the volume can be negative.
7) Compute $V^r(p) = AG^r(p) \cdot r/3 + VI^r(p) + VO^r(p)$

Fig. 6. Algorithm flow for computing integral invariants of a single vertex.

In step 4 of the algorithm, we compute the area invariant value. The time complexity is $O(1)$.

In steps 5 and 6 of the algorithm, we traverse all the facets around point p. Similar to the analysis of step 1, the time complexity is $O((r/l)^2)$.

At last, in step 7, we compute the volume invariant value. The time complexity is $O(1)$.

Overall, the time complexity for computing the area and volume invariants of a single vertex is $O(r^2 \cdot l^{-2})$. The total time complexity is $O(v \cdot r^2 \cdot l^{-2})$, where v is the number of vertices of the mesh model.

Fig. 7 shows that the computing time increases with the number of facets. We have tested our method with seven sphere models composed of different numbers of facets. In Fig. 7, the horizontal axis is the number of facets, while the vertical axis is the computing time (in milliseconds). The seven curves respectively represent seven different values for r (from average edge length to seven times the average edge length).

Table 1 shows the total computing time for different models for a radius five times the average edge length. In this table, column "Time cost" shows the computing time using our algorithm, while columns "Grid Build," "A. Inv.," and "V. Inv." respectively show the computing time for the

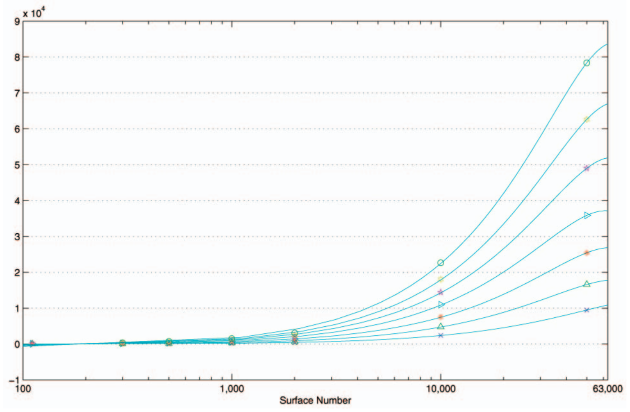


Fig. 7. Computing time increases with increasing numbers of facets.

three steps given in [2]. The grid size parameter in [2] we used is 1/256 of the largest dimension. The time unit is in milliseconds. The timings in Fig. 7 and Table 1 were measured on a 2.8-GHz Pentium 4 running Windows XP.

We can see that as the number of vertices and facets increases, the computing time increases as well on the whole yet is still shorter than the total computing time for the algorithm in [2].

The reason why our algorithm is faster comes from our use of (2). We compute the area and volume invariants with a less complex and, therefore, faster method.

From the results, we can also see the advantage of the algorithm in [2]: if we do not need the area invariant, the total computing time for the volume invariant is almost constant. This constant time may be shorter than the computing time from our algorithm when models become large enough (see the value for Buddha and AsianDragon). Still, there is a bottleneck in the algorithm in [2] for computing area invariants.

Furthermore, when we analyze why the algorithm in [2] performs in an almost constant time, we find that the time is determined by the grid size. The grid size can also determine the error of the algorithm. An error occurs when $B^r(p)$ intersects a grid cube. As the grid size gets smaller, so does the error. However, memory costs and computing time increase significantly.

In the algorithm presented in this article, the error in computing the volume invariant occurs when the model facets intersect $B^r(p)$. If a facet occasionally passes through the center point p, there is no error.

Although we have no method for precisely computing the invariants, we can compare computed volume invariant

TABLE 1
Computing Time Comparison with the Algorithm in [2] with Different Models (Unit: Milliseconds)

Model Name	#Vertex	#Facet	Time cost	Grid Build	A. Inv.	V. Inv.
Maxplanck	11370	22658	17840	113958	37877	30358
Armadillo	23201	46398	29357	115944	74630	21461
Bunny	34834	69451	22531	117364	106983	25310
Buddha	120875	241782	171104	94423	364659	14657
AsianDragon	123365	246730	153779	111049	372119	23046

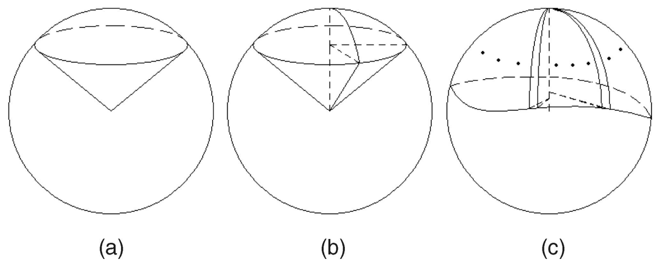


Fig. 8. Examples of changing the area invariant.

values from these two algorithms and estimate the distance between these two algorithms. For the bunny model, the maximum distance is 0.005695, while the maximum error of the algorithm in [2] for this grid size is 0.008. We can conclude from this example that the error of our algorithm is relatively small.

3 Watermarking for a 3D Model

As we mentioned earlier, the integral invariants are robust against noise. Therefore, if we somehow modify the integral invariants to a specific value by changing the vertices' positions, we can apply to the model a watermark strong enough to resist noise attacks.

In this section, we will give more details about how to change these invariants first and then show how these procedures serve as subroutines of a model watermarking algorithm. Our current work modifies the area invariant and the volume invariant.

For convenience, we name O the vertex at the center of the sphere, R the sphere radius, N the average normal of O, and T the point that satisfies $OT = R \cdot N$.

3.1 Changing the Area Invariant

If the model surface around point p is a conical surface (Fig. 8a), the formula giving the area of the spherical intersection is $S = 2\pi Rh$, where S is the area, R is the sphere radius, and h is the height of the cap. Thus, to change the area invariant, we have to change the value of h.

Furthermore, if we only have part of a conical surface (Fig. 8b), we reach the same conclusion for the area of the partial cap with the vertex at the top: we have to change the value of h.

For any 3D surface (Fig. 8c), the area of spherical intersection can be approximated by a number of partial caps like in the last step. Therefore, if we change the value of h for every cap, we change the spherical area.

Therefore, in order to change the area invariant of a 3D mesh model, we can approximately change the value of h of all outer and cross vertices (see the algorithm shown in Fig. 6). The shifting distance is determined by

$$\Delta h = \frac{\Delta S}{2\pi R}. \tag{3}$$

3.2 Changing the Volume Invariant

The basic idea is to move the inner vertices (see the algorithm shown in Fig. 6) along the direction of N. Note that for inner vertices, if we move them along the direction of N for a certain distance, the influence to the volume can be easily calculated. Each movement is independent of the movement of other vertices of the same type. As a result, if we specify how many vertices we move and by how much they are moved along the direction of N, the changed volume can be calculated using the formula given below:

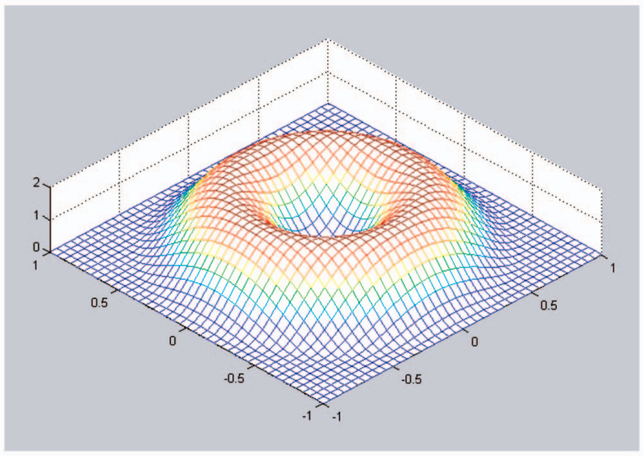


Fig. 9. The optimization function used in changing the volume invariant.

$$\Delta V = -\frac{1}{3}\sum A_i d_i, \tag{4}$$

where A_i is the area of the polygon formed by neighbors of vertex i projected in the direction of N, and d_i is the distance that vertex i is moved along the direction of N. We can see that this is a linear formula for the specified model.

The opposite problem is stated as follows: for a given volume change, by how much should the vertices be moved, while ensuring the watermark invisibility? This is an optimization problem. Since there is no well-proven standard to evaluate invisibility, this optimization problem cannot be formalized easily. The current method we use is to optimize the moving distance to a special function (5) of the distance from the vertex to O (represented with r_i). The modified model is obtained from the original one stamped with the shape shown in Fig. 9. We think that this method is suitable for human vision: especially referring to the effect known as Troxler fading (Troxler, 1804), which states that if you attempt to focus on the center point, the surrounding circles will fade after a few seconds. The special function is given as follows:

$$d_i \propto 1 - \cos\left(2\pi \frac{r_i}{R}\right). \tag{5}$$

The optimization problem is solved as follows: since (5) shows a direct ratio relationship, if we set one of the d_i (for example, d_0), all other d_i and the volume change $\Delta V(d_0)$ for that particular d_0 can be computed. Let d_0' be the value of d_0 that gives the volume change $\Delta V_{\text{watermark}}$ for our watermark. The ratio of $\Delta V(d_0)$ and $\Delta V_{\text{watermark}}$ is equal to the ratio of d_0 and d_0'. From

$$d_0' = \frac{\Delta V_{\text{watermark}}}{\Delta V(d_0)} d_0, \tag{6}$$

we get the d_0'. Next, we compute all the d_i', and the new vertex positions are found.

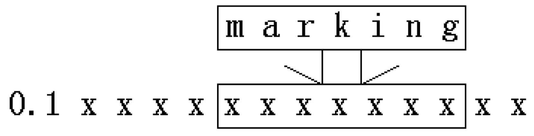


Fig. 10. Replacing bits of a floating-point number to insert information.

3.3 Watermarking a Model

We now use these two integral invariants to insert watermarks into the mesh model, as well as to extract watermarks from models.

First, we choose a monochrome image as the watermark image. This image may be the logo of a company or a group that owns the copyright of the model. Before inserting the watermark into the mesh, we transform it with an Arnold transformation. This is a scrambling procedure that makes the image look like white noise [25]. We will show its usefulness later in this section.

Insertion. The watermark is inserted as follows: First, place neighbor balls centered on the model vertices, making sure that none of them *intersect* each other. Here, intersection means not only that the balls themselves do not intersect but also that the related vertices of the three classes (see the algorithm in Fig. 6) do not overlap. This can be accomplished with the following steps. Traverse all the vertices, trying to place a neighbor ball around each of them. If a new neighbor ball does not intersect any other existing neighbor ball, we add it to the neighbor ball set; otherwise, we discard it. Repeat this procedure on all vertices until no more neighbor balls can be placed. This makes the process of changing invariants in each neighbor ball independent of changes in the other neighbor balls.

Then, we change each invariant value (treated as a floating-point number) by modifying its bit notation, as Fig. 10 shows. The modified bit positions in the invariants are parameters of the algorithm, namely, P_L and P_H are respectively the distance from the point to the lower bit and to the higher bit. For example, in Fig. 10, $P_L = 12$, and $P_H = 6$. $P_L - P_H + 1$ will be the number of embedded bits. Higher positions may cause lower invisibility, and lower positions may cause lower robustness against noise attacks.

The inserted information is part of the scrambled watermark image and its sequence number. We use the indexed localization technique used in [3]. Since we can change two kinds of invariants, there is enough capacity for both the watermark image and the sequence number information. In practice, we change the area invariants to embed the sequence number and change the volume invariants to embed the watermark image.

Note that changing the area invariant will potentially change the volume invariant. Therefore, the area invariant should be changed first, and the volume invariant should be changed after changing the area invariant and updating the vertices' coordinates. By changing the invariants of each neighbor ball, the insertion process is accomplished.

Extraction. The input to the extraction procedure is a model and a watermark image, and the goal is to decide whether the input model has been watermarked with the input watermark image and what kind of attack the model has been subject to. This is why we consider this algorithm as a semifragile watermarking method: it is able to detect which area of the model is attacked.

TABLE 2
Judgement Rules in Testing Procedure

Room for a new ball?	Recovered image Complete?	What kind of attack?
N	Y	Endurable noise
N	N	Cropped
Y	Y	Small local attack
Y	N	Large local attack

First, we prepare an output image with the same size as the input watermark image. Then, all the vertices are traversed to extract the watermark. With each vertex, we try to place a neighbor ball around it. If the ball intersects an existing neighbor ball, it is discarded. Otherwise, we compute the invariants and check the inserted bits. This is the opposite of the insertion procedure. We assume these bits to be the watermark image part and its sequence number. If the assumed sequence number is in the range of the expected sequence number, we test whether the assumed watermark image part is the same as the input watermark image part. If both match, we identify the current neighbor ball as one of the original neighbor balls of the insertion procedure, add this neighbor ball to a set, and copy the assumed watermark image part to the output image at the same position. Otherwise, if the sequence number or the watermark image part do not match, we discard that neighbor ball and continue the traversal. The procedure ends when all the vertices have been traversed.

Model authentications can be done by evaluating how many output image parts have been recovered. We compute the probability of *false-positive* claims, corresponding to the incorrect assertions that a model is watermarked when it is not. This probability can also act as a confidence level. The method to compute this probability is presented in the **Analysis** procedure, and a practical example is also given.

If the input model is watermarked and it is not attacked, the output image should be the same as the watermark image. There is no more room for neighbor balls. Otherwise, if some parts of the model are attacked, there should be room for neighbor balls in the attacked area. Or if the model is cropped, the extracted watermark image would be incomplete.

After the extraction process is finished, all the vertices are traversed to test if there is any room for a new neighbor ball. Then, the decision can be made by applying the rules described in Table 2.

Finally, since the watermark image is first scrambled, the output image after descrambling may lose some random pixels yet still show the information representing the copyright. (See the results in Section 4.)

The flowchart of the insertion process and the extraction process is shown in Fig. 11.

Analysis. The following analysis will compute the probability that a model is not watermarked and an N-size watermark image is recovered with n-size parts.

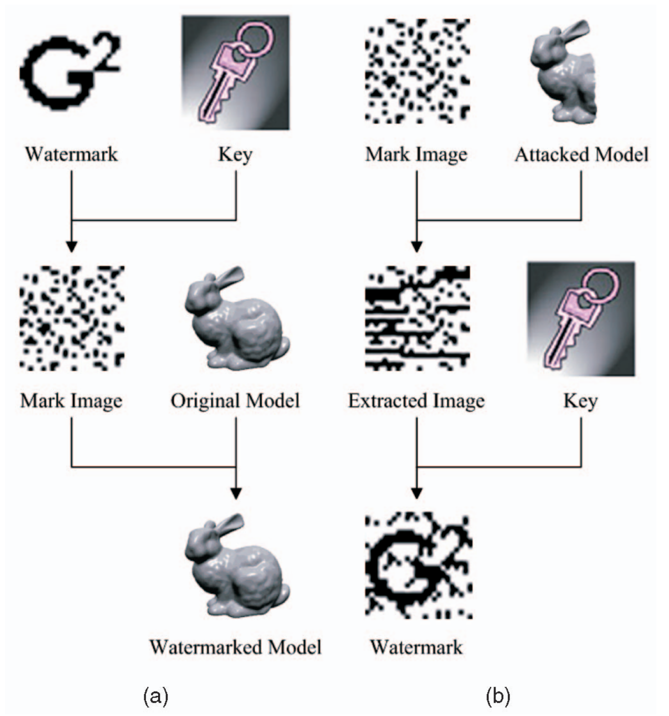


Fig. 11. (a) The insertion procedure. (b) The extraction procedure.

We consider that the integral invariant values' bits for a model that was not watermarked are absolutely random. The probability that the value of a single vertex matches a certain sequence number and the corresponding watermark image part is $p = 1/2^{C_a+C_v}$, where $C_a = P_{La} - P_{Ha} + 1$ and $C_v = P_{Lv} - P_{Hv} + 1$ (see the insertion procedure) are, respectively, the capacity of the area and volume invariants. The probability that at least one of the vertices' invariants (numbering V) matches a certain sequence number and corresponding watermark image part is $P = 1 - (1-p)^V$. Since there are n watermark image parts, the total probability is $(1-(1-p)^V) \cdot (1-(1-p)^{V_2}), \ldots, (1-(1-p)^{V_n})$, where V_i is V minus the number of vertices that have matched one of the previous $i-1$ parts of the watermark image. Since every V_i is smaller than V, we get an upper bound of this probability, as $(1-(1-p)^V)^n = P^n$. For a given problem, p and V are constants, but we can adjust the value of n in order to have this probability small enough.

For instance, let us embed a 24 × 24 watermark image into a model of 34,834 vertices (the bunny model). If we set $P_{La} = P_{Lv} = 12$ and $P_{Ha} = P_{Hv} = 6$, we have $C_a = C_v = 7$ and $p = 2^{-14}$. $\lceil 24 \times 24 \div 7 \rceil = 83$ watermark image parts will be embedded into the model. For the model made up of 34,834 vertices, if we set the radius of the neighbor ball to five times the average edge length, a single neighbor ball will cover approximately 100 vertices. Therefore, there will be at most $V = 34{,}834/100 = 348$ neighbor balls. We can then deduce that $P \doteq 2$ percent. If we want to reduce the probability to less than 10^{-10}, we should set the threshold to seven parts (49 bits). This means that after the extraction procedure, if there are less than seven (8 percent) watermark image parts recovered, we conclude negatively on the model authentication (the model was not watermarked). Otherwise, we conclude positively when the false-positive probability is lower than 10^{-10}.

TABLE 3
False-Positive Probability in Several Cases

C_a	C_v	V	r/l	$size$(bits)	$capacity$(bytes)
7	8	34834	5	48	348
4	4	34834	5	68	174
7	8	34834	10	32	91
4	4	34834	10	80	45
7	8	123365	5	56	1233
4	4	123365	5	11600	616×
7	8	123365	10	40	325
4	4	123365	10	284	162

If the model is larger, we can either increase the capacity of the invariants or increase the radius of the neighbor ball. Increasing the capacity of the invariants will decrease the value of the probability p but may lower the robustness. Increasing the radius of the neighbor ball will decrease the maximum number of neighbor balls but will decrease the information capacity. Another solution is to increase the size of the watermark image, which will have the threshold relatively unchanged. We chose our final parameters by finding a trade-off between these three solutions. Some other cases are shown in Table 3, where r/l is the radius divided by average edge length, $size$ is the minimal watermark image size, and $capacity$ is the maximal watermark image size. The data is obtained assuming that the false-positive probability is lower than 10^{-10}. Note that the case marked with a "×" is an impossible case: it is not possible to have the false-positive probability lower than 10^{-10}, so we should change one or more parameters.

4 Experimental Test

In this section, we show some experimental test results, including the visualization of watermarked models, the distortion error results, and the comparison results of robustness against noise and cropping. If there is no special mention, the tests are done on the bunny model (34,834 vertices and 69,451 facets) using a 24 × 24 monochrome image as the input watermark image (see Fig. 12). Parameters are set as follows: $P_{La} = 12$, $P_{Ha} = 6$, $P_{Lv} = 13$, $P_{Hv} = 6$, and the radius of the neighbor ball is five times the average edge length.

4.1 Watermark Invisibility

Fig. 13 shows the model before and after embedding the watermark. In the figure, green, yellow, and red vertices

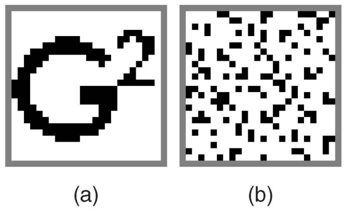


Fig. 12. (a) The watermark. (b) The scrambled watermark.

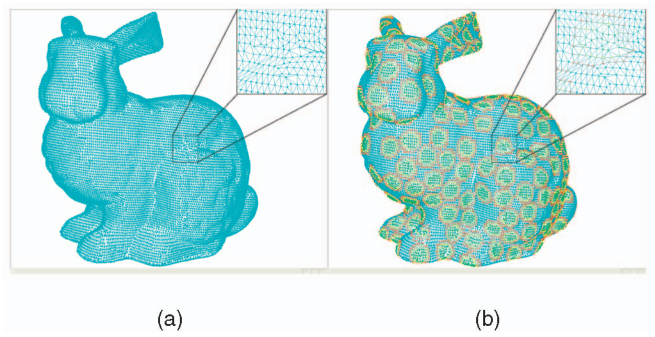


Fig. 13. (a) The original model. (b) The watermarked model.

represent the three vertex classes (inner, cross, and outer) where the watermark is embedded.

But if we choose some "bad" parameters, the difference between the original and watermarked model becomes visible (Fig. 14).

Experimental tests using other models are shown in Fig. 15. Fig. 15a shows the original model, Fig. 15b shows the watermarked model, and Fig. 15c shows the appearance if they are put together (the blue one is the watermarked).

Error measurements calculated using Metro [26] are given in Table 4, where models 1 to 5 are respectively Maxplanck, Armadillo, Bunny, Buddha, and AsianDragon. In Table 4, the "AEL" row is the model average edge length, the "Area" and the "Area W." rows are the areas of all triangles of the original model and the watermarked model, the "Mean" row is the mean distance between the corresponding points before and after watermarking, and the "Haus." row is the Hausdorff distance between the original model and the watermarked model.

We conclude that the watermarking process is nearly invisible.

4.2 Robustness against Additive Noise

As we stated above, the watermark can survive under noise attacks. We can easily show that robustness against noise increases with the radius of the kernel ball (see Fig. 16). In this figure, the horizontal axis is the ratio of the neighbor ball radius and the average edge length, while the vertical axis is the noise amplitude when the watermark survives with less than 1 percent bit error rate (*BER*). We add a random number uniformly drawn from $-a$ to $+a$ to each vertex coordinate, where a is the noise amplitude. In this figure, the noise is applied to every model vertex.

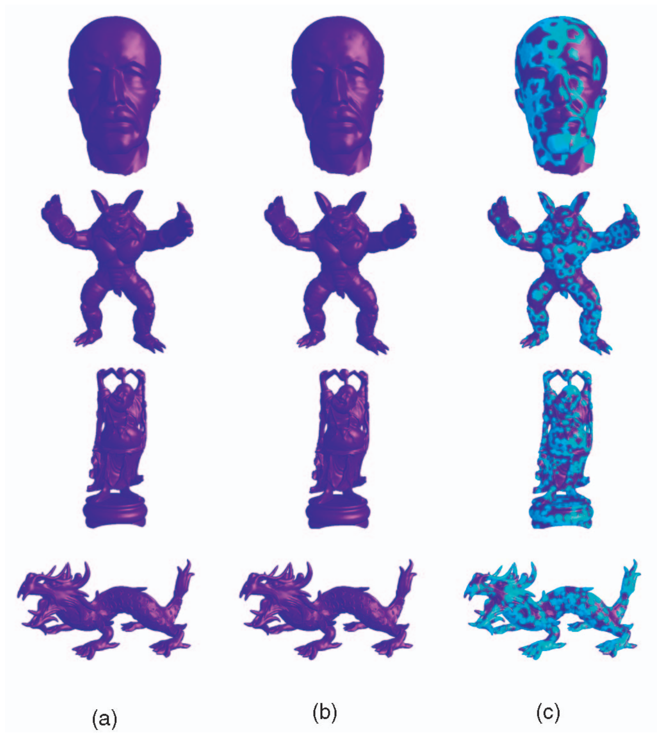


Fig. 15. (a) The original model. (b) The watermarked model. (c) The overlayed models.

TABLE 4
Distortion Error Cased by Watermark Insertion

Model No.	1	2	3	4	5
Vertex No.	11370	23201	34834	120875	123365
Facet No.	22658	46398	69451	241782	246730
Area	8.665	6.589	0.0571	5.665	3.829
Area W.	8.675	6.596	0.0572	5.696	3.835
AEL(10^{-3})	30.2	18.8	1.47	7.34	5.90
Mean(10^{-4})	3.84	1.72	0.17	3.08	2.84
Haus.(10^{-3})	9.58	7.17	0.322	3.12	2.06

In order to compare our results with that of other algorithms from the literature, we use the method in [9] to test robustness against noise. In Fig. 17, we compare our method with those in [9] and [3]. The horizontal axis is the

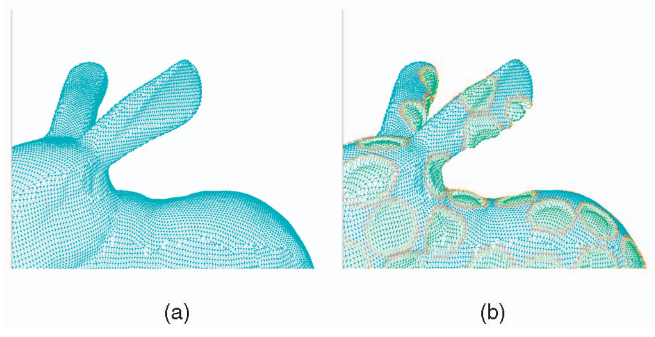


Fig. 14. Bad parameters cause the changes to become visible. (a) The original model. (b) The watermarked model.

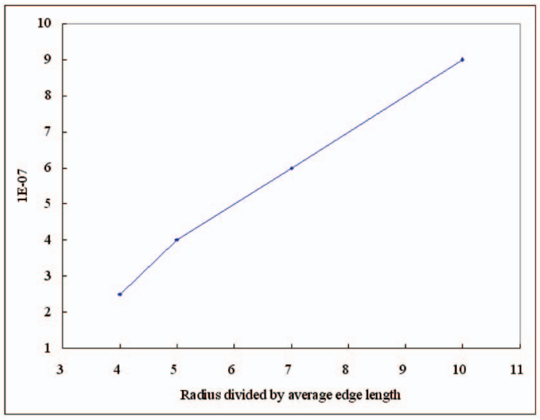


Fig. 16. Robustness against noise increases with the radius of the neighbor ball.

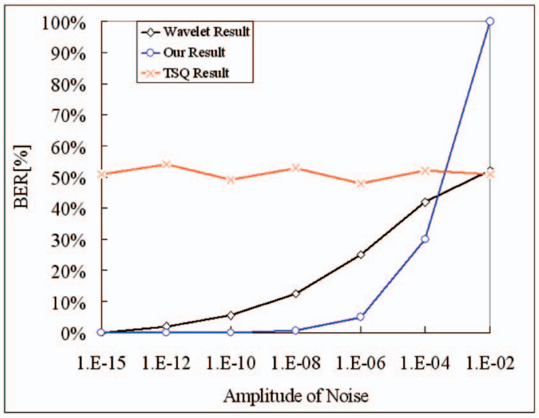


Fig. 17. Comparison of robustness against noise with [3] and [9].

noise amplitude, and the vertical axis is the BER result after the previously described noise attack was applied to all the vertices. In this figure, we can see that the method in [3] (the red line), whose robustness to noise was unspecified, is actually very sensitive to noise. This means that the watermark can be totally broken by little noise. The extracted watermark is a random binary string with a nearly 50 percent BER. The figure also shows that our method's BER is lower than the result in [9] (the black line). Note that for 10^{-2}, our BER is nearly 100 percent instead of a rate around 50 percent for the TSQ method. This can be simply explained: if the watermark is completely broken, our method outputs nearly no bits into the output image.

Similar to the experiments in [9], we test our method when noise attacks are applied to some of the vertices. The comparison result is shown in Fig. 18. In this figure, the horizontal axis is the percentage of vertices that undergo a noise attack of amplitude 10^{-5}. The figure clearly shows that our method (the blue line) is more robust against noise than the wavelet method.

4.3 Robustness against Cropping

As we mentioned in Section 3, if the model is cropped, the output image will be an incomplete watermark image.

We test the robustness against cropping as follows: First, we use each plane from a set of parallel planes to produce a cropped version of the model. For each model produced, its degree of cropping is measured by counting the percentage of remaining vertices. Fig. 19 shows some examples demonstrating the effects of cropping. After the extraction procedure, only a fraction of the watermark is recovered.

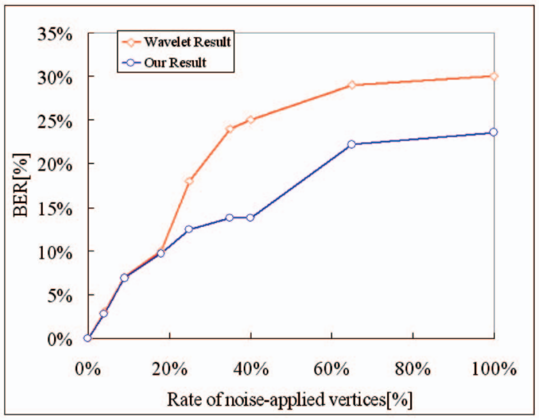


Fig. 18. BER for different rates of noise.

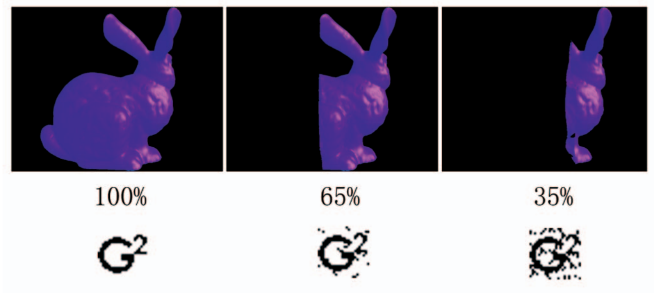


Fig. 19. Watermark images extracted from models cropped by different amounts.

5 Conclusion and Future Work

We have presented a semifragile watermarking method based on integral invariants. It is a spatial domain method robust against rigid transformations and noise attacks. Experimental tests show that this method is suitable to determine whether a model was attacked.

We could improve our algorithm by increasing its embedding capacity, currently limited to two integral invariants. One solution would be using multiresolution analysis methods to simplify the model and embed a watermark at the corresponding simplified model vertex. Another solution would be to find a method to simultaneously change four kinds of integral invariants. These solutions are the directions of our future work.

Acknowledgments

The authors are grateful to Dr. Paul Rosin for his helpful advice. And the authors are also deeply thankful for the reviewers' detailed comments, which helped to improve the quality of the paper. This work was supported by the National Basic Research Project of China (Project 2006CB303104), the Natural Science Foundation of China (Project 60673004, U0735001), and the National High Technology Research and Development Program of China (Project 2007AA01Z336). The models used in this paper are courtesy of Stanford University and AIM@SHAPE shape repository.

References

[1] S. Manay, B.W. Hong, A.J. Yezzi, and S. Soatto, "Integral Invariant Signatures," *Proc. Eighth European Conf. Computer Vision (ECCV '04)*, pp. 87-99, 2004.
[2] H. Pottmann, Q.X. Huang, Y.L. Yang, and S. Kölpl, "Integral Invariants for Robust Geometry Processing," technical report, Geometry Preprint Series, Vienna Univ. of Technology, 2005.
[3] R. Ohbuchi, H. Masuda, and M. Aono, "Watermarking Three-Dimensional Polygonal Models," *Proc. Fifth ACM Int'l Conf. Multimedia (Multimedia '97)*, pp. 261-272, 1997.
[4] R. Ohbuchi, H. Masuda, and M. Aono, "Embedding Data in 3D Models," *Proc. Fourth Int'l Workshop Interactive Distributed Multimedia Systems and Telecomm. Services (IDMS '97)*, pp. 1-10, 1997.

[5] R. Ohbuchi, H. Masuda, and M. Aono, "Watermarking Three-Dimensional Polygonal Models through Geometric and Topological Modifications," *IEEE J. Selected Areas in Comm.,* vol. 16, no. 4, pp. 551-560, 1998.

[6] R. Ohbuchi, H. Masuda, and M. Aono, "Data Embedding Algorithms for Geometrical and on Geometrical Targets in Three-Dimensional Polygonal Models," *Computer Comm.,* vol. 21, no. 15, pp. 1344-1354, 1998.

[7] O. Benedens and C. Busch, "Towards Blind Detection of Robust Watermarks in Polygonal Models," *Proc. Eurographics '00,* pp. C199-C208, 2000.

[8] O. Benedens, "Affine Invariant Watermarks for 3D Polygonal and NURBS Based Models," *Proc. Third Int'l Workshop Information Security (ISW '00),* pp. 15-29, 2000.

[9] S. Kanai, H. Date, and T. Kishinami, "Digital Watermarking for 3D Polygons Using Multiresolution Wavelet Decomposition," *Proc. Sixth IFIP WG 5.2 Int'l Workshop Geometric Modeling,* pp. 296-307, 1998.

[10] F. Uccheddu, M. Corsini, and M. Barni, "Wavelet-Based Blind Watermarking of 3D Models," *Proc. ACM Multimedia and Security Workshop,* pp. 143-154, 2004.

[11] R. Ohbuchi, A. Mukaiyama, and S. Takahashi, "A Frequency-Domain Approach to Watermarking 3D Shapes," *Proc. Eurographics '02,* pp. 373-382, 2002.

[12] F. Cayre, P. Rondao-Alface, F. Schmitt, B. Macqb, and H. Maître, "Application of Spectral Decomposition to Compression and Watermarking of 3D Triangle Mesh Geometry," *Signal Processing,* vol. 8, no. 4, pp. 309-319, 2003.

[13] F. Cayre, O. Deviller, F. Schmitt, and H. Maître, *Watermarking 3D Triangle Meshed for Authentication and Integrity,* INRIA Research Report RR-5223, June 2004.

[14] E. Praun, H. Hoppe, and A. Finkelstein, "Robust Mesh Watermarking," *Proc. ACM SIGGRAPH '99,* pp. 325-334, 1999.

[15] K.K. Yin, Z.G. Pan, J.Y. Shi, and D. Zhang, "Robust Mesh Watermarking Based on Multiresolution Processing," *Computers & Graphics,* vol. 25, no. 3, pp. 409-420, 2001.

[16] L. Li, D. Zhang, Z.G. Pan, J.Y. Shi, K. Zhou, and K. Ye, "Watermarking 3D Mesh by Spherical Parameterization," *Computers & Graphics,* vol. 28, no. 6, pp. 981-989, 2004.

[17] J.H. Wu and L. Kobbelt, "Efficient Spectral Watermarking of Large Meshes with Orthogonal Basis Functions," *Visual Computer,* vol. 21, nos. 8-10, pp. 848-857, 2005.

[18] J.W. Cho, R. Prost, and H.Y. Jung, "An Oblivious Watermarking for 3-D Polygonal Meshes Using Distribution of Vertex Norms," *IEEE Trans. Signal Processing,* vol. 55, no. 1, pp. 142-155, 2007.

[19] S.K. Lee and Y.S. Ho, "A Fragile Watermarking Scheme for Three-Dimensional Polygonal Models Using Triangle Strips," *IEICE Trans. Comm.,* vol. 87, no. 9, pp. 2811-2815, 2004.

[20] C.M. Chou and D.C. Tseng, "A Public Fragile Watermarking Scheme for 3D Model Authentication," *Computer-Aided Design,* vol. 38, no. 11, pp. 1154-1165, 2006.

[21] D. Zou, Y.Q. Shi, Z. Ni, and W. Su, "A Semi-Fragile Lossless Digital Watermarking Scheme Based on Integer Wavelet Transform," *IEEE Trans. Circuits and Systems for Video Technology,* vol. 16, no. 10, pp. 1294-1300, 2006.

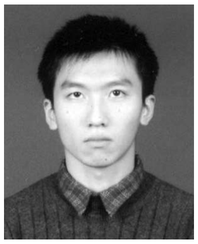
[22] Y.L. Yang, Y.K. Lai, S.M. Hu, and H. Pottmann, "Robust Principal Curvatures on Multiple Scales," *Proc. Fourth Eurographics Symp. Geometry Processing (SGP '06),* pp. 223-226, 2006.

[23] H. Pottmann, Q.X. Huang, Y.L. Yang, and S.M. Hu, "Geometry and Convergence Analysis of Algorithms for Registration of 3D Shapes," *Int'l J. Computer Vision,* vol. 67, no. 3, pp. 277-296, 2006.

[24] Y.K. Lai, Q.Y. Zhou, S.M. Hu, J. Wallner, and H. Pottmann, "Robust Feature Classification and Editing," *IEEE Trans. Visualization and Computer Graphics,* vol. 13, pp. 34-45, 2007.

[25] W. Ding, W.Q. Yan, and D.X. Qi, "Digital Image Scrambling Technology Based on Arnold Transformation," *J. Computer-Aided Design and Computer Graphics,* vol. 13, no. 4, pp. 338-341, 2001.

[26] P. Cignoni, C. Rocchini, and R. Scopigno, "Metro: Measuring Error on Simplified Surfaces," *Computer Graphics Forum,* vol. 17, no. 2, pp. 167-174, 1998.



Yu-Ping Wang is currently a PhD student in the Department of Computer Science and Technology, Tsinghua University, Beijing. He is also with the Tsinghua National Laboratory for Information Science and Technology. His research interests include digital watermarking, computer graphics, geometric modeling, and processing.

Shi-Min Hu received the PhD degree from Zhejiang University in 1996. He is currently a chair professor in the Department of Computer Science and Technology, Tsinghua University, Beijing. He is also with the Tsinghua National Laboratory for Information Science and Technology. His research interests include digital geometry processing, video processing, rendering, computer animation, and computer-aided geometric design. He is on the editorial board of *Computer Aided Design*. He is a member of the IEEE.