A New Watermarking Method for 3D Model based on Integral Invariant

Yu-Ping Wang[1], and Shi-Min Hu[1]

Technical Report **TR-080301**
Tsinghua University, Beijing, China

[1]Department of Computer Science and Technology, Tsinghua University, Beijing, China.

A new watermarking method for 3D model based on integral invariant

Yu-Ping Wang[1*] and Shi-Min Hu[1]
[1]Department of Computer Science and Technology, Tsinghua University, China
wyp05@mails.tsinghua.edu.cn, shimin@tsinghua.edu.cn
Phone:+86 10 6279 7231; Fax:+86 10 6279 7459

***Abstract*— In this paper, we propose a new semi-fragile watermarking algorithm for the authentication of 3D models based on integral invariants. To do so, we embed a watermark image by modifying the integral invariants of some of the vertices. Basically, we shift a vertex and its neighbors in order to change the integral invariants. To extract the watermark, test all the vertices for the embedded information, and combine them to recover the watermark image. How many parts can the watermark image be recovered would help us to make the authentication decision. Experimental test shows that this method is robust against rigid transform and noise attack, and useful to test purposely attack besides transferring noise and geometrical transforming noise. An additional contribution of this paper is a new algorithm for computing two kinds of integral invariants.**

***Index Terms*— Semi-fragile watermark, 3D model watermarking, Integral invariants.**

I. 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 position of vertices, the color of texture points or other elements representing the model. The frequency domain methods embed the watermark by modifying the transformation coefficients.

There is no unified standard to test which algorithm is better. There are some applications where one method is found to be best suited than the other. Nevertheless, watermarking algorithms are usually characterized with the four following properties: - Validation: the watermark can be fully extracted from the original 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 noise attacks.

Spatial domain algorithms work on certain 3D model invariants, like TSQ, TVR [3]–[6], AIE [7], [8], etc. to embed the watermark. But most of them are very sensitive to noise. Frequency domain algorithms provide better robustness ability by using wavelet analysis [9], [10], Laplace transforms [11], [12] and other transforms [14]–[16]. Nevertheless, the 3d model distortions fail to be invisible, or extraction routines require the original watermarked model 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, fragile watermarking has not been researched abundantly 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. Similarly to image watermarking nomenclature [21], we characterize watermarking algorithms showing this property as semi-fragile.

In this paper, we propose a new semi-fragile 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], [13]. Since we wish to embed the watermark with geometrical invariants, we introduce 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 they can also be put to use in our problem. First, we calculate the current integral invariant of the vertex that will undergo watermarking. We then change these values slightly to embed the watermark image parts. Finally, we modify the position of the vertices and their neighbors in order to change the integral invariants to the new value. The extraction routine is the inverse process of the insertion routine. We compute the integral invariants for all the vertices and try to match the embedded information. Once matched, we extract the embedded information at each vertex from two integral invariants and combine this data to form the extracted watermark. By analyzing the false-positive probability, we finally make the authentication decision.

Notice that we need to compute the integral invariants of part of vertices once in the insertion procedure, and the integral invariants of all the vertices once in the extraction procedure.

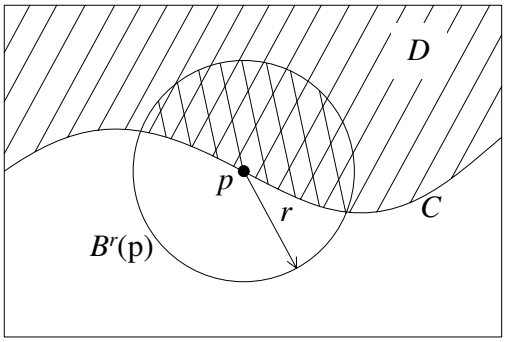

Fig. 1. Area invariant for planar curves

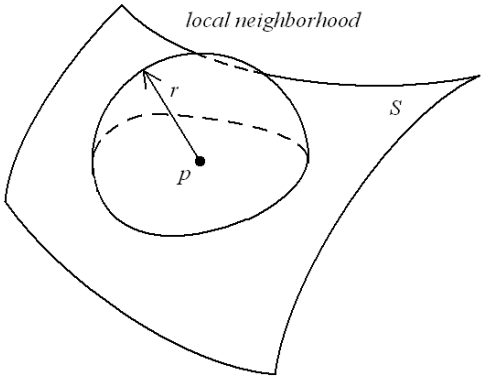

Fig. 2. Local neighborhood in $\mathbb{R}^3$

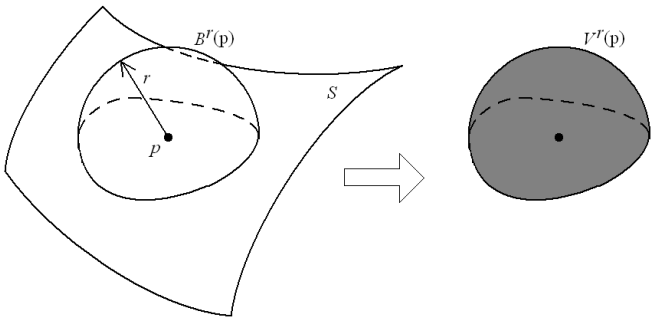

Fig. 3. Volume invariant in $\mathbb{R}^3$

In practice, we find that the known algorithm to compute the integral invariants is not fit to this application. Therefore, we have brought a faster algorithm for computing integral invariants.

The structure of this paper is organized as follows. In section II, 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 III, we explain the watermark embedding and extraction algorithms in detail. In section IV, we show some of our experimental results. Finally, in section V, we conclude and mention potential improvement in future work.

II. Integral invariants

A. 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 to estimate 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 el 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 would like to introduce two kinds of them used in the following sections.

If we extend area invariant from R^2 to R^3, we get volume invariant (see Fig 3). Considering the local neighborhood as a ball $B^r(p)$ of radius r, centered at p, and 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, and 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 more robust against noise than traditional differential invariants such as curvature [22]–[24]. Explicitly proving and experimental results can be found in [2].

B. A faster algorithm for computing integral invariants

In this subsection, 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 main procedure is to first compute the area invariant and then the volume 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

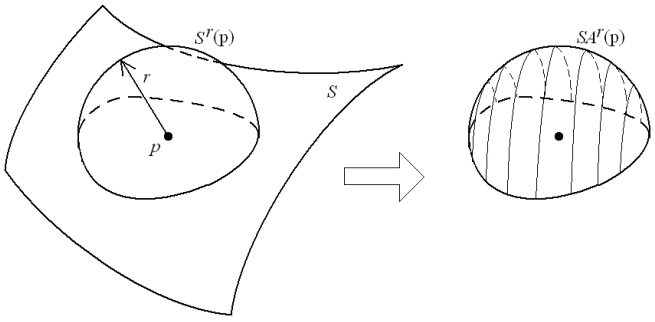

Fig. 4. Area invariant in $\mathbb{R}^3$

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 end points forming each arc are the intersection of $S^r(p)$ with edges of the model. It is easy to calculate them. We then compute an approximation of the area by replacing each of these arcs with the great arc on $S^r(p)$ with the same end points. Therefore the intersection surface becomes a polygon on the sphere.

The formula for computing the area of a spherical triangle is:

$$S = \alpha + \beta + \gamma - \pi \tag{1}$$

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:

$$S = \sum \alpha_i - (n-2)\pi \tag{2}$$

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

After we get the area invariant, we start to compute the volume invariant. Fig 5 is sketches of this computing procedure to help understanding. In Fig 5, a, b, and c are 2D sketches, and d is a 3D sketch. First, we multiply the area invariant with $r/3$, which is the volume of a irregular cone (the shadow part in Fig 5a). Then, we compute the difference between this volume and the volume invariant by plus or minus part of the volume difference ('+' or '-' parts in Fig 5b). This is hard to compute directly, but can be computed easier by converting it to plus or minus the tetrahedron constructed with all the triangular facet inside $B^r(p)$ as underside and p as apex ('+' or '-' parts in Fig 5c). Notice that special process will be need near the boundary of neighbor ball. We need to get rid of the volume of outer tetrahedron part. There are two kinds of cases, one vertex of the triangular facet is outside the neighbor ball, or two vertices of the triangular facet is outside (see Fig 5d, together with the common case). Finally, we get the volume invariant.

Overall, we give the list of algorithm flow in Fig 6.

Repeat the algorithm shown in Fig 6 for each vertex of the model, we will get the invariants for all the vertices.

C. Algorithm analyzing

The complexity of the algorithm is easily 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 step 2 and step 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 $O(r/l)$.

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

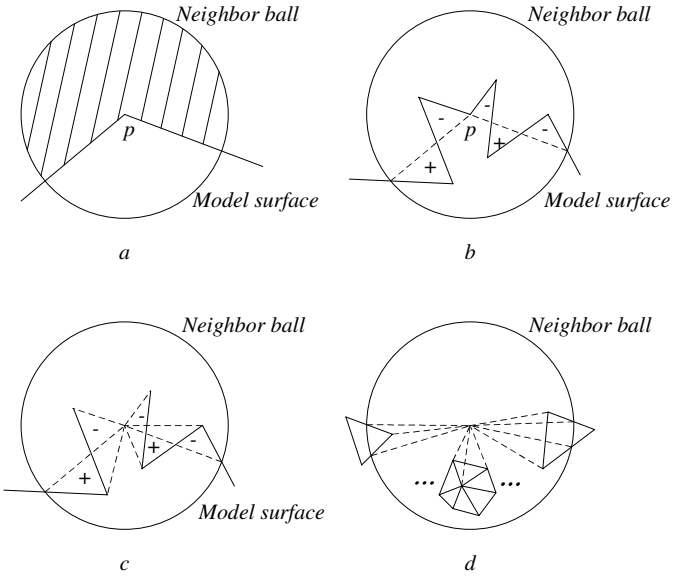

Fig. 5. Computing volume invariant from area invariant

In step 5 and step 6 of the algorithm, we traverse all the facets around point p. Similarly 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 amount of facets. We tested our method with seven sphere models composed of different number 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).

Fig 8 shows the total computing time for different models, for a radius five times the average edge length. In this figure, column "Time cost" shows the computing time using our algorithm, while the columns "Grid Build", "A. Inv.", and "V. Inv." respectively show the computing time for the three steps given by [2]. The grid size parameter in [2] we used is $1/256$ of the largest dimension. The time unit is in millisecond.

We can see that, as the number of vertices and facets increase, the computing time increases as well on the whole, yet is still shorter than the total computing time.

The reason that our algorithm is faster comes from our usage of the formula (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 from [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 get large enough (see the value for Buddha and AsianDragon). Still, there is a bottleneck in the algorithm from [2] for computing area invariants.

Furthermore, when we analyze why the algorithm in [2]

1) Find the vertices around point p, divide them into 3 classes: inner, cross, and outer. Vertexes 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. Vertexes of class cross are in the ball $B^r(p)$, but at least one of their direct neighbors is out of the ball. Vertexes of class outer are out of the ball $B^r(p)$.
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 $S^r(p)$. Notice 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 to $AG^r(p)$.
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 to $VI^r(p)$. Notice that the volume can be negative.
6) For each facet that cross the sphere $S^r(p)$, compute volume of the pyramid construct with the inner part of the facet and point p, and sum them to $VO^r(p)$. Notice 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 invariants of a single vertex

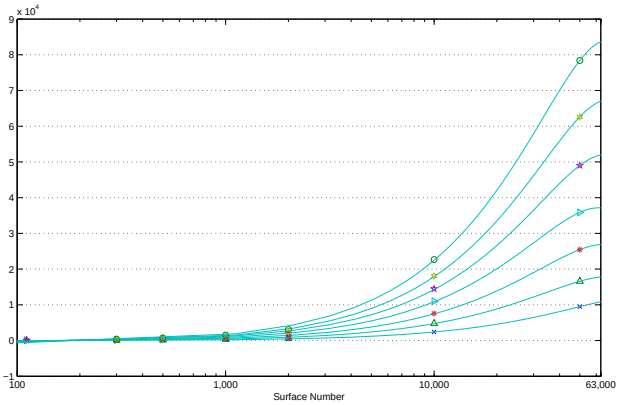

Fig. 7. Computing time increases with the increasing of facet number

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 happens when the $B^r(p)$ 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 volume invariant computing error happens when the model facets intersect $B^r(p)$. If a facet occasionally passes through the center point p, there is no error.

Since we have no method for precisely computing the invariant, we can compare computed volume invariant values from these two algorithms and estimate the distance between these algorithms. For the bunny model, the distance is 0.005695, while the error of the reference algorithm is 0.008. We can conclude from this example that the error of our algorithm is relatively small.

Model Name	#Vertex	#Facet	Time cost	Grid Build	A. Inv.	V. Inv.
Maxplunk	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. Computing time compare with algorithm in [2] with different models(unit:ms)

III. Watermarking for 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 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 $OT = R \cdot N$.

A. Changing the area invariant

If the neighbor surface is a taper (Fig 9 a), 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.

Further, if we only have part of a taper (Fig 9 b), 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 9 c), the area of spherical intersection can be approximated by a number of partial caps like in the last step. So 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 shift is determined by:

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

B. Changing the volume invariant

The basic idea is to move the vertex along the direction of N. Notice that for inner vertices (see the algorithm shown in Fig 6), if we move them along the direction of N a certain distance, the influence to the volume can be easily calculated. That shift is independent of the movement of other vertices of the same type. As a result, if we specify how many vertices

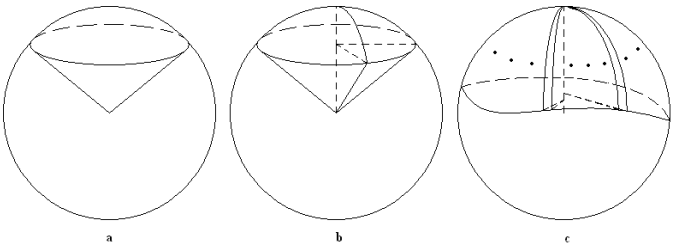

Fig. 9. Examples of changing area invariant

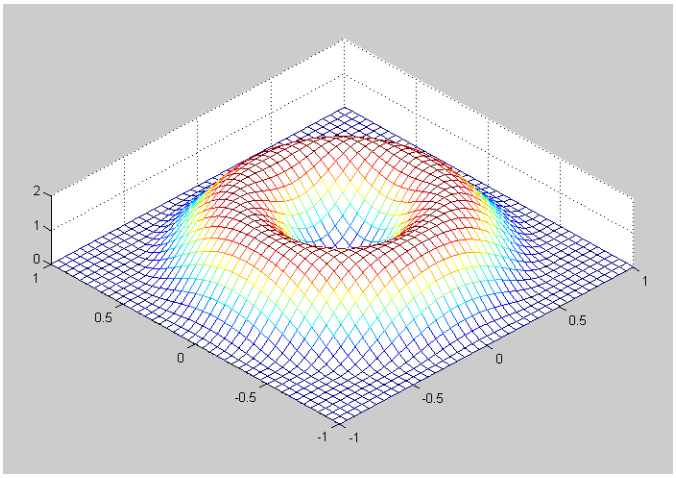

Fig. 10. The optimization function used in changing volume invariant

we move and by how much they are moved along the direction of N, the volume change can be calculated with the formula listed below.

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

where A_i is the area of 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 (formula (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 10). We think that this method is fit to human vision: specially 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.

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

The optimization problem is solved as follows: since formula (5) shows a direct ratio relation, 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

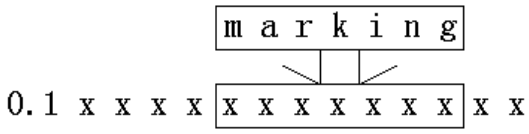

Fig. 11. Replace bits of a float-point number to insert information

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 formula 6, we get the d_0'. Further we compute all the d_i', and the new vertices positions are found.

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

C. 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 use at a later point in this subsection.

Insertion. The watermark is inserted as follows. First, we place balls centered on the model vertices, making sure that none of them *intersect* each other. Here, intersection not only means the balls themselves do not intersect, but the related vertices of the three classes (see Algorithm in Fig 6) also do not overlap. This can be accomplished with the following steps. We 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. We do this procedure on all vertices until no more neighbor ball can be placed. This makes the process of changing invariants in each neighbor ball independent of the change the other neighbor balls.

Then, we change each invariant value (treated as floating point numbers) by modifying its bit notation, as Fig 11 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 11, $P_L = 12$ and $P_H = 6$. $P_L - P_H + 1$ will be the amount 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 the 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 scrambled number and change the volume invariants to embed the watermark image.

Notice that changing the area invariant will potentially change the volume invariant. So we change the area invariant first, update the vertices coordinates and change the volume invariant next. By changing the invariants of each neighbor ball, the insertion process is accomplished.

Extraction. First, we prepare an output image the same size as the watermarked image. We then traverse all the vertices to to extract the watermark. We try to place a neighbor ball around each vertex. If the ball intersects an existing neighbor ball, we discard it. 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 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 part of the watermark image is the same as the real watermark image part. If both match, we identify the current neighbor ball as one of the original neighbor balls of the insertion procedure. We add that neighbor ball to a set and copy the assumed part of the watermark image 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.

If the watermarked model 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 part of the model is attacked, there should be room for neighbor balls in the attacked area. Also, if the model is cropped, the extracted watermark image is incomplete.

After the extraction process is finished, we still need to execute a test procedure. We traverse all the vertices to test if there is any more room for a neighbor ball. We apply the rules described in the following table to deduce what kind of attack the model has been subject to. This is why we declare this algorithm as a semi-fragile watermarking method: it is able to detect what area of the model was attacked.

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

Model authentications can be done by evaluating how much of the output image is complete. 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.

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 of section)

The flow chart of the insertion process and the extraction process is shown in Fig 12.

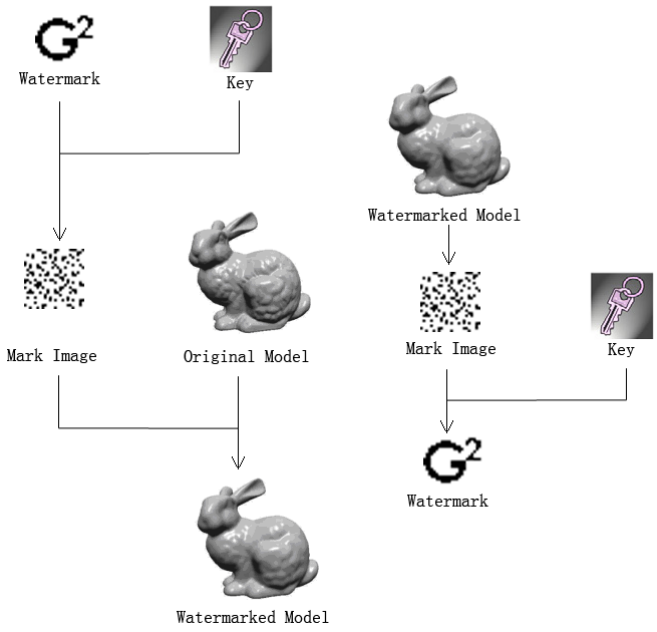

Fig. 12. The insertion process(Left) and The extraction process(Right)

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

We consider that the integral invariant values' bits for a model that was not watermarked are absolutely random. The probability that a single point value 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 V vertices' invariants matches a certain sequence number and corresponding watermark image part is $P = 1 - (1-p)^V$. Since there are n watermark image parts, the totally probability is $(1-(1-p)^V)\cdot(1-(1-p)^{V_2})\cdot\cdots\cdot(1-(1-p)^{V_n})$, where V_i is V minus the number of vertices that have matched one of the previously $i-1$ 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, we embed a 24×24 watermark image into a model of 34834 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 34834 vertices, if we set the radius of the neighbor ball to 5 times the average edge length, a single neighbor ball will cover approximately 100 vertices. Therefore, there will be at most $V = 34834/100 = 348$ neighbor balls. We can then deduce $P \doteq 2\%$. If we want to reduce the probability to less than 10^{-10}, we should set the threshold to 7 parts (49 bits). This means that after the extraction procedure, if there are less than 7 (8%) 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}.

If the model is larger, we can either increase the capacity of invariants or increase the radius of the neighbor ball. Increasing the capacity of 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 tradeoff between these three solutions. Some other cases are shown in Fig 13, 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}. Notice that the case with a $'\times'$ is an impossible case: it is impossible to make the false-positive probability lower than 10^{-10}, so we should change one or many parameters.

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

Fig. 13. False-positive probability in several cases

IV. Experimental test

In this section, we show some experimental test results, including the visualization of watermarked models, the distortion error results, the comparison of anti-noise ability and anti-crop test results. If there is no special mention, the tests are done on the bunny model (34834 vertices, 69451 facets) using a 24×24 monochrome image as input watermark image (see Fig 14). Parameters are set as $P_{La} = 12$, $P_{Ha} = 6$, $P_{Lv} = 13$, $P_{Hv} = 6$, the radius of the neighbor ball is 5 times the average edge length.

A. Watermark invisibility

Fig 15 shows the model before and after embedding the watermark. In the figure, green, yellow, and red vertices represent the three vertex classes (inner, cross, and outer) where the watermark is embedded.

But if we choose some 'bad' parameters, the difference of the original and watermarked model would be visible (Fig 16).

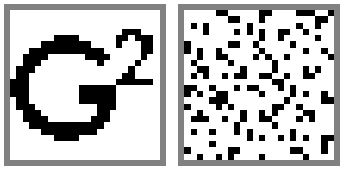

Fig. 14. The watermark(Left) and The scrambled watermark(Right)

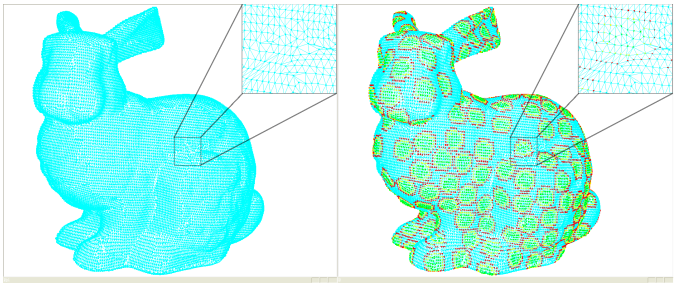

Fig. 15. The original model(Left) and The watermarked model(Right)

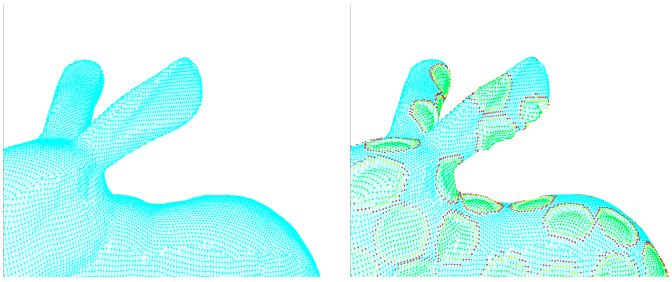

Fig. 16. Bad parameters cause the changing visible. The original model(Left) and The watermarked model(Right)

Experimental tests using other models is shown in Fig 17. The first column is the original model, the second is the watermarked model, and the third is the appearance if they are put together (the blue one is the watermarked).

Error measurements calculated using Metro [26] are given in Fig 18, where models 1 to 5 are respectively Maxplunk, Armadillo, Bunny, Buddha, and AsianDragon. In this figure, the 'AEL' row is the model average edge length, the 'Max' and the 'Mean' row is the maximum and mean distance between the corresponding points before and after watermarking, the 'Area' and the 'Area W.' row are the area of all triangles of the original model and the watermarked model 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.

B. Anti-noise capability

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 19). 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% BER (bit error rates). We add a random number 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.

In order to compare our results with other algorithms from the literature, we use the method from [9] to test anti-noise robustness. In Fig 20, we compare our method with those in [9] and [3]. The horizontal axis is the noise amplitude, and the vertical axis is the BER result after the previously described

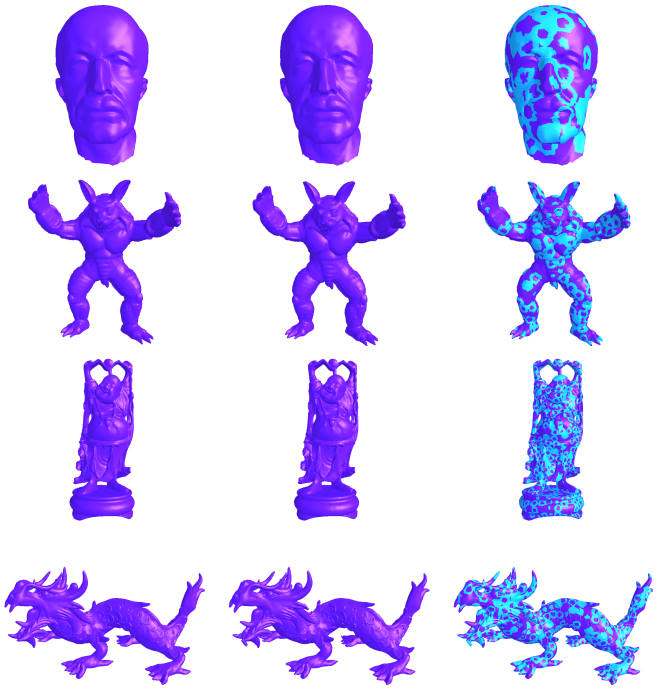

Fig. 17. The original model (left), the watermarked model (middle) and the overlayed models (right)

Model No.	1	2	3	4	5
Vertex No.	11370	23201	34834	120875	123365
Facet No.	22658	46398	69451	241782	246730
AEL(1E-3)	30.2	18.8	1.47	7.34	5.90
Max(1E-4)	44.4	30.6	3.22	51.0	43.6
Mean(1E-4)	3.84	1.72	0.17	1.26	1.20
Area	8.665	6.589	0.0571	5.665	3.829
Area W.	8.675	6.596	0.0572	5.68	3.839
Haus.(1E-3)	9.58	7.17	0.322	25.18	21.74

Fig. 18. Distortion error cased by watermark insertion

noise attack was applied to all the vertices. In this figure, we see that the method in [3] (the red line), which is not known to be robust against noise, is actually very sensitive to noise. This means that the watermark can be totally broken with little noise. The extracted watermark is a random binary string with a nearly 50% BER. The figure also shows that our method's BER is lower than the result from [9] (the black line). Notice that for 10^{-2}, our BER is nearly 100% instead of a rate around 50% 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.

Similarly to the experiments in [9], we test our method when noise attacks are applied to certain vertices. The comparison result is shown in 21. In this figure, the horizontal axis is the amount 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.

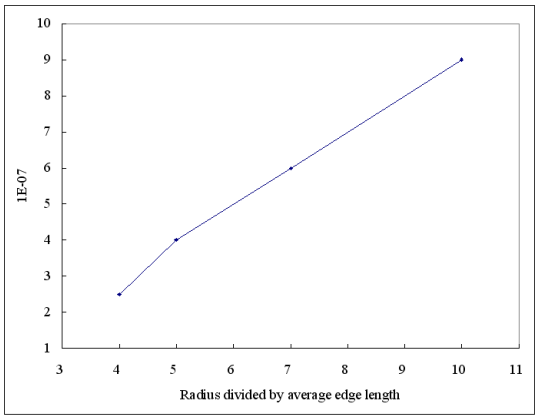

Fig. 19. Robustness against noise increases with the radius of the neighbor ball

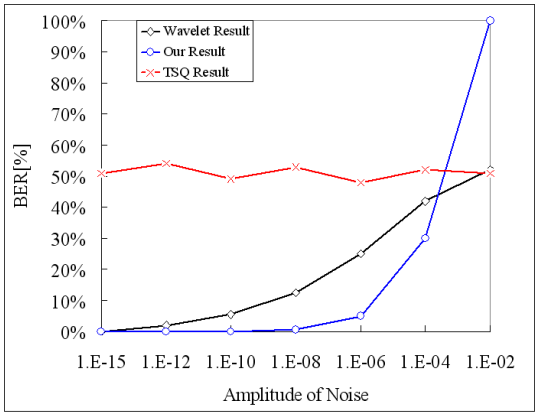

Fig. 20. Comparison of anti-noise ability with [3] and [9]

C. Anti-crop capability

As we mentioned in section III, if the model is cropped, the output image will not be a complete watermark image.

We test the anti-crop capability as follows. First, we crop the model with a set of parallel planes and count the percentage of remaining vertices. Then, we select some of these vertices as the input model to test the anti-crop capability. After the extraction procedure, only a fraction of the watermark is recovered (see Fig 22). The percentage shown in the figure is the remaining vertices percentage which is only an approximation ($|error| < 1\%$).

V. Conclusion and Future Work

We have presented a semi-fragile watermarking method based on integral invariants. It is a spatial domain method robust against rigid transforms 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 multi-resolution analysis methods to convert the model into a simplified model and embed a watermark at the corresponding simplified model vertex. Another solution would be finding a method to change four

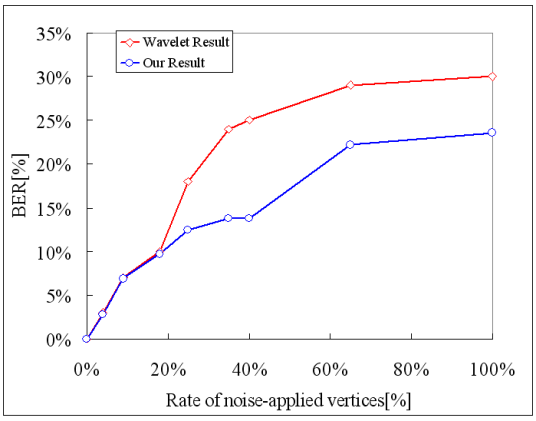

Fig. 21. BER for different rates of noise

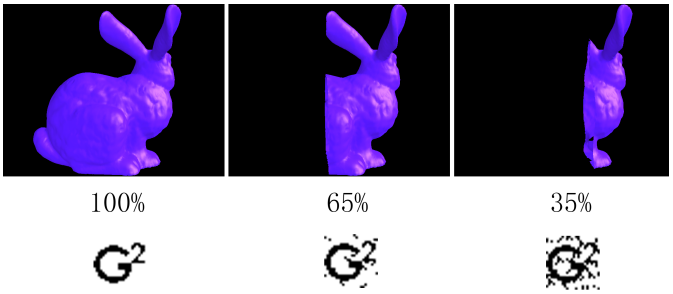

Fig. 22. Watermark images extracted from cropped models with different percentages

kinds of integral invariants simultaneously. These solutions are the directions of our future work.

REFERENCES

[1] Siddhart Manay, Byung-Woo Hong, Anthony J. Yezzi, and Stefano Soatto *Integral invariant signatures*, In Proceedings of ECCV 2004, number LNCS 3024, pages 87-99. Springer, 2004.

[2] Helmut Pottmann, Qixing Huang, Yongliang Yang, and Stefan KAolpl. *Integral invariants for robust geometry processing*, Technical report, Geometry Preprint Series, Vienna Univ. of Technology, 2005.

[3] Ohbuchi R, Masuda H, Aono M. *Watermarking Three Dimensional Polygonal Models*, Proc. ACM Multimedia 97, 261-272, 1997.

[4] Ohbuchi R, Masuda H, Aono M. *Embedding data in 3D models*, Proceedings of the European Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services 97, Darmstadt, 1-10, 1997.

[5] Ohbuchi R, Masuda H, Aono M. *Watermarking three-dimensional polygonal models through geometric and topological modifications*, IEEE Journal on Selected Areas in Communication, 16(4), 551-560, 1998.

[6] Ohbuchi R, Masuda H, Aono M. *Data embedding algorithms for geometrical and on geometrical targets in three-dimensional polygonal models*, Computer Communications, 21(15), 1344-1354, 1998.

[7] Benedens O, Busch C. *Towards blind detection of robust watermarks in polygonal models*, Proceedings of Eurographics, Interlaken, C199-C208, 2000.

[8] Benedens O. *Affine invariant watermarks for 3D polygonal and NURBS based models*, Proceedings of the 3rd International Workshop Information Security, Wollongong, 15-29, 2000.

[9] Kanai S, Date H, Kishinami T. *Digital watermarking for 3D polygons using multiresolution wavelet decomposition*, Proceedings of International Workshop on Geometric Modeling, Tokyo , 296-307, 1998.

[10] Uccheddu F, Corsini M, Barni M. *Wavelet-based blind watermarking of 3D models*, Proceedings of the 2004 Multimedia and Security Workshop, Magdeburg, 143-154, 2004.

[11] Ohbuchi R, Mukaiyama A, Takahashi S. *A frequency-domain approach to watermarking 3D shapes*, Proceedings of Eurographics'02, Saarbrucken, 373-382, 2002.

[12] Cayre F, Rondao-Alface P, Schmitt F, et al. *Application of spectral decomposition to compression and watermarking of 3D triangle mesh geometry*, Signal Processing, 18(4):309-319, 2003.

[13] Cayre F, Deviller O, Schmitt F, Maitre H. *Watermarking 3D triangle meshed for authentication and integrity*, INRIA Research Report RR-5223, June 2004.

[14] Praun E, Hoppe H, Finkelstein A. *Robust mesh watermarking*, Computer Graphics Proceedings, Annual Conference Series, ACM SIGGRAPH, Los Angeles, 325-334, 1999.

[15] Yin K K, Pan Z G, Shi J Y, et al. *Robust mesh watermarking based on multiresolution processing*, Computers & Graphics, 25(3):409-420, 2001.

[16] Li L, Zhang D, Pan Z, et al. *Watermarking 3D mesh by spherical parameterization*, Computers & Graphics, 28(6):981-989, 2004.

[17] Wu JH, Kobbelt L. *Efficient spectral watermarking of large meshes with orthogonal basis functions*, Visual Computer, 21(8-10):848-857, 2005.

[18] Cho JW, Prost R, Jung HY. *An oblivious watermarking for 3-D polygonal meshes using distribution of vertex norms*, IEEE Transactions on Signal Processing, 55(1):142-155, 2007.

[19] Lee SK, Ho YS. *A fragile watermarking scheme for three-dimensional polygonal models using triangle strips*, IEICE Transactions on Communications, E87B(9):2811-2815, 2004.

[20] Chou CM, Tseng DC. *A public fragile watermarking scheme for 3D model authentication*, Computer-Aided Design, 38(11):1154-1165, 2006.

[21] Dekun Zou, Yun Q. Shi, Zhicheng Ni, and Wei Su. *A Semi-Fragile Lossless Digital Watermarking Scheme Based on Integer Wavelet Transform*, IEEE Transactions on Circuts and Systems for Video Technology, 16(10):1294-1300, 2006.

[22] Yong-Liang Yang, Yu-Kun Lai, Shi-Min Hu and Helmut Pottmann. *Robust Principal Curvatures on Multiple Scales*, Proceedings of 4th Eurographics Symposium on Geometry Processing, Eurographics Association, p.223-226, 2006.

[23] Helmut Pottmann, Qi-Xing Huang, Yong-Liang Yang and Shi-Min Hu. *Geometry and Convergence Analysis of Algorithms for Registration of 3D Shapes*, International Journal of Computer Vision, 67(3), 277-296, 2006.

[24] Yu-Kun Lai, Qian-Yi Zhou, Shi-Min Hu, Johannes Wallner and Helmut Pottmann. *Robust Feature Classification and Editing*, IEEE Transaction on Visualization and Computer Graphics, Vol.13, pp.34-45, 2007.

[25] Ding Wei, Yan Weiqi, Qi DongXu. *Digital Image Scrambling Technology Based on Arnold Transformation*, Journal of Computer-Aided Design & Computer Graphics, 13(4), 338-341, 2001.

[26] Cignoni P, Rocchini C and Scopigno R. *Metro:Meaturing Error on Simplified Surfaces*, Computer Graphics Forum, 17(2), 167-174, 1998.