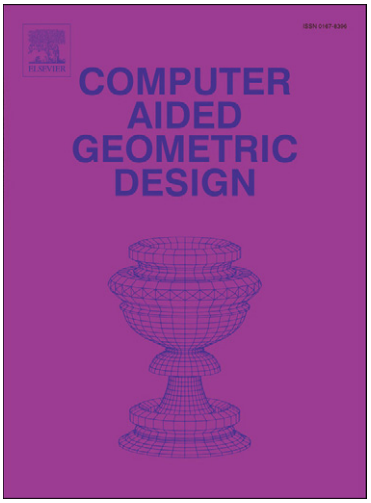


Optimization approach for 3D model watermarking by linear binary programming

Yu-Ping Wang, Shi-Min Hu

PII: S0167-8396(10)00024-5
DOI: 10.1016/j.cagd.2010.02.003
Reference: COMAID 1208

To appear in: *Computer Aided Geometric Design*

Received date: 17 July 2008
Revised date: 24 January 2009
Accepted date: 12 February 2010

Please cite this article as: Y.-P. Wang, S.-M. Hu, Optimization approach for 3D model watermarking by linear binary programming, *Computer Aided Geometric Design* (2010), doi: 10.1016/j.cagd.2010.02.003

Optimization Approach for 3D Model Watermarking by Linear Binary Programming

Yu-Ping Wang[a,b], Shi-Min Hu[a,b]

[a] *Tsinghua National Laboratory for Information Science and Technology*
[b] *Department of Computer Science and Technology, Tsinghua University, China 100084*

Abstract

Watermarking algorithms provide a way of hiding or embedding some bits of information in a watermark. In the case of watermarking a 3D model, many algorithms employ a so-called indexed localization scheme. In this paper, we propose an optimization framework with two new steps for such watermarking algorithms to improve their capacity and invisibility. The first step is to find an optimal layout of invariant units to improve capacity. The second step is to rearrange the correspondence between the watermark units and the invariant units to improve invisibility. Experimental tests show that by using this framework, the capacity and invisibility of watermarking algorithms can be greatly improved.

Key words: 3D model watermarking, Linear binary programming

1. Introduction and related works

With the development of CAD, biomedical imaging, 3D animation and other applications, 3D models play an important role in a wide variety of fields. Meanwhile, increasing network bandwidths coupled with compression techniques for 3D triangle meshes have brought 3D models to the public via the Internet. Copyright protection of 3D models is necessary. As an effective method of protecting copyright, 3D model watermarking algorithms have been researched in recent years.

Although there is no unified criterion to decide which algorithm is better, watermarking algorithms are usually characterized by the four following properties:

1. *Validation:* it should be possible to fully extract the watermark from an un-attacked model. Authentication can be decided by whether extraction is successful or not.
2. *Invisibility:* the watermarked model should look similar to the original model, otherwise watermarked models are useless in practice. This property is particularly important when watermarking CAD-oriented models.

Email addresses: wyp05@mails.tsinghua.edu.cn (Yu-Ping Wang), shimin@tsinghua.edu.cn (Shi-Min Hu)

3. *Capacity:* this corresponds to the amount of information that can be embedded into the model. Larger capacity will broaden possible usage of the data hiding algorithm, and make the authentication decision more reliable.
4. *Robustness:* the watermark should survive different types of attacks.

The main difference between watermarking a 3D model and watermarking an image is the type of attacks. In the 3D model case, re-ordering attacks should be resisted first. Such attacks arise because no implicit order exists for the vertices of 3D models, unlike pixels for image data. An image watermarking algorithm could embed some bits of watermark in every pixel of a 2D image, and simply recompose the information by using the order of the pixels during extraction. But this can not be done when dealing with 3D models since the extraction procedure may obtain bits of the watermark without order. Thus, some way to index the vertices must be considered by any 3D model watermarking algorithm. We call this *localization* following Cayre et al. (2004).

In the first article on 3D model watermarking in 1997 Ohbuchi et al. (1997b), one solution to the localization problem called indexed localization was given, and this has also been employed by many later 3D model watermarking algorithms such as Yeo and Yeung (1999); Benedens and Busch (2000); Benedens (2000); Cayre et al. (2004); Chou and Tseng (2006); Wang and Hu (2009). Methods of this kind divide the watermark into several parts, give each part of the watermark a subscript, and embed each part the of watermark together with its subscript and a marker. During the extraction procedure, the subscript determines the order of the extracted watermark parts and the marker determines whether the extracted watermark parts are valid. Although information capacity is wasted by adding subscripts, it is a robust way to solve the localization problem.

Another solution to the localization problem can be called global localization, and is employed by Praun et al. (1999); Li et al. (2004); Uccheddu et al. (2004); Wu and Kobbelt (2005). Methods of this kind attempt to recover the order of model vertices after a re-ordering attack. Most algorithms employing this scheme are spectral algorithms. Algorithms employing principal component analysis (PCA) or spherical parameterization can be considered to be of this kind. However, the order recovered by this scheme depends on the integrity of the model. If the watermarked model is cropped, the order can have a break in the middle, or may be destroyed if the first vertex is lost. Advanced algorithms use the registration technique introduced in Besl and McKay (1992); Chen and Medioni (1992); Alekseyev (2003), which aligns the attacked model with the un-attacked watermarked model (or the original model in Kanai et al. (1998)). After successful registration, the vertex order can be recovered even if the model is cropped. But using the un-attacked model or the original model (for simplicity, the reference model) limits applicability of such algorithms.

A compromise scheme is called local localization by Ohbuchi et al. (1997b). This scheme divides the vertices into groups, acting as global localization within each group, and acting as indexed localization between groups. However, this scheme does not perform well and few papers employ it in practice.

Most 3D model watermarking algorithms, including all spectral algorithms, employ a global localization scheme. One reason is that there is no spectral domain if the order of

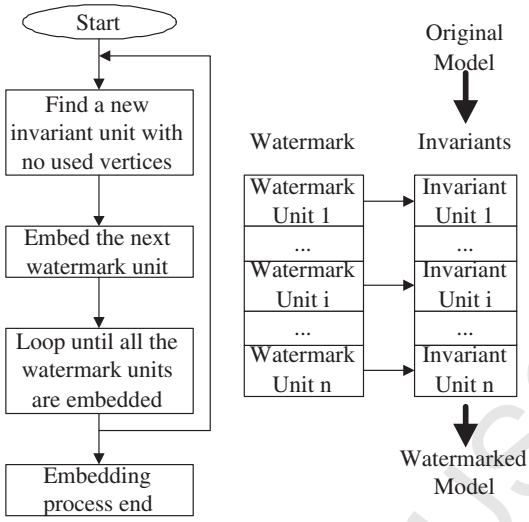


Figure 1: Typical watermarking algorithm flow chart

vertices is not determined first. Another reason is that by using the reference model, global localization schemes can provide greater capacity and better robustness than indexed localization schemes. However, global localization schemes are vulnerable under the counterfeit original attack introduced in Craver et al. (1998). In this paper, we thus focus on watermarking algorithms employing indexed localization schemes, and we will show that indexed localization schemes can be much improved.

As noted, indexed localization schemes first divide the watermark into several parts and give each part of a watermark a subscript. A *watermark unit* is defined as a triple (W, S, M), where W is the watermark part, S is its subscript and M is the marker. In order to embed these bits into a 3D model, algorithms of this kind always look for some invariants. Different algorithms choose different invariants, and the robustness of algorithms is determined by the properties of the invariants. If the invariants remain unchanged after the model is modified by various kinds of attack, the watermarking algorithm is said to be robust against such attacks. By modifying the coordinates of various vertices, these invariants are set to a new value as a watermark unit is embedded. An *invariant unit* is defined as the smallest set of vertices that may be modified in order to embed a watermark unit.

Algorithms repeatedly look for an invariant unit that shares no vertices with the valid invariant units used in prior iterations, and embed a watermark unit into it. An invariant unit is *valid* if it is (or going to be) used to embed a watermark unit. After embedding each watermark unit, the vertices in the invariant unit are flagged as *used* to avoid being traversed again by the following iterations. By doing so, it is guaranteed that the valid invariant units do not share vertices with each other, otherwise modification of one invariant unit may change other invariant units too, and destroy watermark units embedded in prior loops. Thus, a typical flow chart of a watermarking algorithm is as shown in figure 1.

Algorithms cannot, however, obtain the largest set of valid invariant units by simply

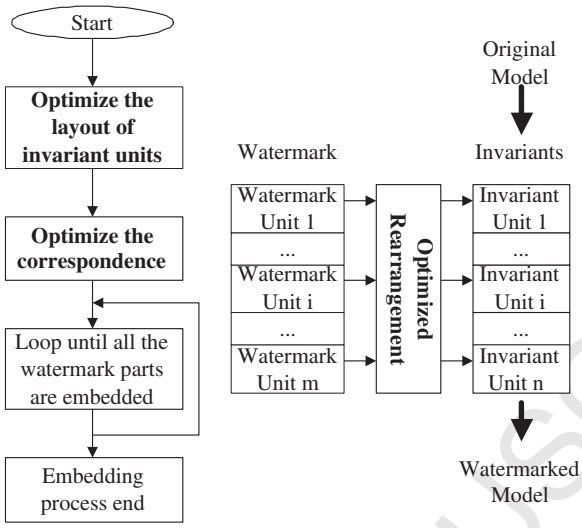


Figure 2: The optimized watermarking algorithm flow chart

traversing all vertices. Since having more valid invariant units will provide a larger capacity, that it is useful to find an optimal layout of invariant units to improve the capacity.

On the other hand, there is an implicit rule within the flow chart shown in figure 1: the watermark unit indexed i is embedded into the ith invariant unit. However, this implicit rule is not necessary. If we break this rule, we can rearrange the correspondence between watermark units and invariant units to improve the invisibility.

Therefore, we change the flow chart of existing watermarking algorithms from that in figure 1 to the one shown in figure 2; the extraction procedure remains the same. The key problem is to optimize the layout of invariant units and the correspondence between watermark units and invariant units. As we will show later, these two problems are instances of linear binary programming problems.

For convenience, we call existing watermarking algorithms *un-optimized algorithms*, and algorithms acting like figure 2 *optimized algorithms*.

The rest of this paper is organized as follows. In section 2, we explain the optimization framework in detail. In section 3, we explain how the optimization framework works on several watermarking algorithms, and further terminology will be clarified using examples. In section 4, we show experimental results. Finally, in section 5, we summarize our work and discuss its limitations.

2. Optimization framework

As mentioned, we convert the flow chart of existing watermarking algorithms from one like figure 1 into one like figure 2. As figure 2 shows, there are two steps before the core embedding procedure. The first step optimizes the layout of invariant units to improve capacity; the second step optimizes the correspondence between watermark units and invariant

units to improve invisibility.

The first step provides the valid invariant units which act as input of the second step. After the second step, the correspondence between watermark units and invariant units is determined, which allows execution of the core embedding procedure. The latter is the core of existing watermarking algorithms, which describes how to modify the vertices of an invariant unit when embedding a watermark unit. Our optimization framework leaves the core embedding procedure unchanged.

The rest of this section explains these two optimization steps.

2.1. Improving capacity

As explained, valid invariant units should share no vertices with each other; they are a subset of all possible invariant units. Our purpose is to maximize the number of valid invariant units. This can be transformed into a *maximum independent set problem*:

$$\max\{x|Ax \leq \mathbf{1}, x \in \{0,1\}^n\} \tag{1}$$

where x is a vector with n rows and A is a matrix with m rows and n columns, where m is the number of mesh vertices and n is the number of possible invariant units. The ith element of x is 1 or 0, representing whether the ith possible invariant unit is valid or not, respectively. The element in the jth row and the ith column of A is 1 or 0 representing whether the jth vertex is included in the ith possible invariant unit or not.

For a given problem, A is an input which can be obtained from the model and the watermarking algorithm, while x is the output which is the layout of invariant units we are seeking. Producing matrix A relies on properties of invariant units. Rather than flagging the vertices of the valid invariant units as used, we traverse all possible invariant units and record the vertices included in them using the matrix A.

This kind of maximum independent set problem is known to be NP complete, but it can be approximately solved using various modern optimization technique such as ant colony optimization algorithms Leguizamon and Michalewicz (2001). For this problem, we first construct a graph of possible invariant units. In this graph, each node represents a possible invariant unit, and if two possible invariant units share at least one vertex, there is an edge between the two corresponding nodes. Thus, the problem is converted to the one of finding the maximum node set such there is no edge between any pair of nodes in this set. As a heuristic rule, a node is selected with a higher probability if it has a lower degree in the graph. The complexity for this algorithm is $O(nm)$ as is the space complexity for matrix A. The first step to improve the capacity relies on its result. In practice, we find that if we always add nodes with lowest degree, the result is good enough.

2.2. Improving invisibility

Invisibility generally refers to how little the appearance of the model is changed by watermarking. It is hard to define, and many researchers have consider it. Cignoni et al. (1998) and Aspert et al. (2002) consider geometric errors based on Hausdorff distance which do not correlate well with human perception. The two methodologies in Corsini et al.

(2007) are better in this aspect. Lavoué et al. (2006) propose a method named MSDM which extended a 2D similarity method introduced in Wang et al. (2004).Bian et al. (2009) evaluated visibility using strain energy, but there is still no standard method.

Although the invisibility evaluation function plays an important role in the following optimization step, we make some coarse assumptions: if any evaluation function become standard, the function used in the following step can be replaced by this new standard. Our coarse assumption is that smaller local vertex movement causes a smaller local appearance change. *Local* here means the vertices around one invariant unit. We can get a coarse local evaluation function by measuring local vertex movement.

If invisibility is defined as the *sum* of all local changes, the optimization problem for maximizing invisibility can be written as:

$$\begin{aligned} &\min \textstyle\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \\ \text{such that}\quad &\textstyle\sum_{j=1}^{n} x_{ij} = 1, \quad \sum_{i=1}^{n} x_{ij} = 1, \quad x_{ij} \in \{0,1\}. \end{aligned} \tag{2}$$

where c and x are matrices with n rows and n columns, c is the input and x is the output, c_{ij} is the value of the local evaluation function if the jth watermark unit is embedded in the ith invariant unit, and x_{ij} is 1 or 0, representing whether the jth watermark unit embeds the $i$the invariant unit or not. There must be exactly one 1 in each row and column of x, because each watermark unit can be embedded only once, and only one watermark unit can be embedded into each invariant unit. $\sum_{j=1}^{n} c_{ij}x_{ij}$ represents the local change in the ith invariant unit, and the goal of this optimization is to minimize the sum of all local changes. This is an assignment problem which can be solved using the Hungarian algorithm. The complexity for this algorithm is $O(n^3)$, but it can run much faster in practice.

Otherwise, if the invisibility is defined as the *maximum* of all local changes, the optimization problem can be as:

$$\begin{aligned} &\min\max \textstyle\sum_{j=1}^{n} c_{ij}x_{ij} \\ \text{such that}\quad &\textstyle\sum_{j=1}^{n} x_{ij} = 1, \quad \sum_{i=1}^{n} x_{ij} = 1, \quad x_{ij} \in \{0,1\}. \end{aligned} \tag{3}$$

where c and x have the same meaning as in equation 2. The only difference is the goal, which is to minimize the maximum of all local changes, not the sum. This problem is much easier than the previous one. It can be solved with a greedy version of the Hungarian algorithm:

1) Sort the n^2 elements in matrix c in ascending order. Let the sorted array be C.
2) Let $L = n$ and $R = n^2$.
3) Pick the first $K = (L + R)/2$ elements in C, and try to find an assignment with the chosen elements. If it cannot be done, let $L = K$, otherwise let $R = K$.
4) Repeat until $L = R - 1$.

The complexity of this algorithm is determined by the first step, sorting an array with n^2 elements, and is thus $O(n^2 log(n))$.

Note that the n here represents the number of valid invariant units which is determined in the first step, and it is usually much less than the number of vertices (and the number of possible invariant units). Thus, both cases of problem can be solved in reasonable time.

The experimental tests in section 4 use the Hausdorff distance in Cignoni et al. (1998) to evaluate invisibility, which is a widely used method. Since the Hausdorff distance can be expressed as the maximum geometric error, the formulation in equation 3 is suitable.

3. Working with existing algorithms

From Section 2, we can summarize the main framework of optimized algorithms as having the following steps:

1. First traverse all possible invariant units, and record the relationship between the invariant units and the vertices in the matrix A in equation 1.
2. Using A, the invariant units graph is constructed and the first optimization step is performed. After the first step, the set of valid invariant units is determined.
3. Next, the optimized algorithm traverses all valid invariant units and all watermark units, and local changes to invisibility that each pair may cause is recorded in matrix c in equation 3.
4. Using c, the second optimization step is performed, giving the relationship between valid invariant units and watermark units.
5. Finally, the core embedding procedure is performed.

In existing watermarking algorithms, the method used to create watermark units from a watermark is always the same. A watermark with W bits is split into w bit pieces, of which there are $p = \lceil W/w \rceil$. A watermark unit includes an m bit marker, an s bit subscript, and a w bit watermark, altogether $m + s + w$ bits. The marker is used to determine whether the watermark unit is valid: a valid watermark unit should include an m bit previously chosen marker. The subscript is used to determine the order of watermark units. We require that $2^s \geq p$, i.e. the number of watermark units must be no greater than the maximum number the s bit subscript can express.

The only differences when optimizing different algorithms are the characters of invariant units. Different watermarking algorithms may employ different invariants. In order to convert an existing algorithm to an optimized algorithm, the methods used to create invariant units should be considered.

In the following of this section, we will explain how our optimization framework operates with three existing watermarking algorithms as examples: the TSQ (Triangle Similarity Quadruple embedding) algorithm, the AIE (Affine Invariant Embedding) algorithm and the IIE (Integral Invariant Embedding) algorithm. Since we do not change the core embedding method in which a watermark unit is embedded into a invariant unit, we omit details of these algorithms; references are given below.

3.1. Using the TSQ algorithm

The TSQ algorithm was introduced by Ohbuchi et al. (1997a,b, 1998a,b). It was the first watermarking algorithm dealing with 3D models, and has influenced almost all other algorithms. Therefore we apply our optimization framework to it first.

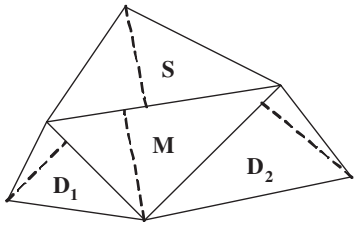


Figure 3: The invariant unit of the TSQ algorithm

An invariant unit for the TSQ algorithm is called an MEP in Ohbuchi et al. (1997a)It includes the vertices associated with four triangles shown in figure 3.

Each invariant unit has triangles named M, S, D_1 and D_2. Triangle M embeds the marker, S embeds the subscript, and D_1 and D_2 embed the watermark data. For each triangle, two ratios are used to embed data: the ratio of the height to the opposite edge length, and the ratio of the other two edges lengths. These two ratios can be easily obtained from a triangle, and as they determine a set of similar triangles, are invariant under similarity transforms.

As can be seen, an invariant unit for the TSQ algorithm is determined by the triangle M. The un-optimized algorithm traverses all the facets of the original model, and tests if the invariant unit determined by the facet is valid. Once a valid invariant unit has been found, the algorithm picks the next watermark unit to carry out the core embedding procedure: the marker is first embedded into the triangle M by moving the three vertices of triangle M. Then the subscript and the data are embedded by moving the vertex that the corresponding triangle does not share with the triangle M.

Thus, the optimized algorithm can consider all possible invariant units by considering all the facets of the original model. Since the invariant unit includes only 6 vertices, the first optimization step can be quickly finished. But the set of valid invariant units sets may contain over 1000-3000 valid invariant units (depending on the model size), which can cause the second optimization step to take more time.

3.2. Using the AIE algorithm

The AIE algorithm was introduced by Benedens and Busch (2000). This watermarking algorithm focuses on providing robustness against affine transforms by using as an invariant the Nielson-Foley norm introduced in Nielson and Foley (1989). It directly describes how the watermark bits are divided into parts and embedded into vertices with a bit-level description.

The invariant unit for AIE algorithm is called an embedding primitive in Benedens and Busch (2000), and comprises the vertices shared by two adjacent triangles T_1 and T_2, as shown in figure 4.

The four vertices of the two triangles T_1 and T_2 may be used to determine affine coordinates which work as the basis of the Nielson-Foley norm. Thus, for each vertex other than these four vertices, a norm value can be calculated which is an affine invariant. The algorithm embeds the watermark into this norm value, and thus the watermark can survive under affine transform attacks.

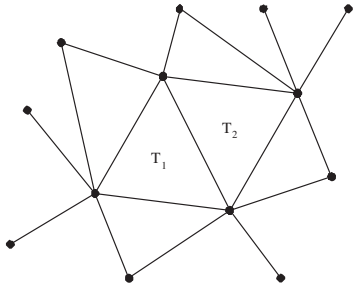


Figure 4: Invariant unit of the AIE algorithm

The invariant unit for the AIE algorithm can be determined by the edge that triangle T_1 and triangle T_2 share. The un-optimized algorithm traverses all edges of the original model, and tests if the invariant unit determined by the edge is valid. Once a valid invariant unit has been found, the algorithm picks the next watermark unit to carry out the core embedding procedure. During embedding, the vertices of an invariant unit next to the four vertices of triangle T_1 and T_2 are divided into four groups. 6 bits are embedded into the vertices of each group, representing index or data, by moving the norm value to the middle of the nearest interval numbered by these 6 bits.

Therefore, the optimized algorithm can consider all possible invariant units by considering all edges of the original model. As in the TSQ algorithm, the invariant unit of AIE algorithm includes no fewer than 8 vertices, so the first optimization step is quick, but the second optimization step takes longer.

3.3. Using the IIE algorithm

The IIE algorithm was introduced by Wang and Hu (2009). It is a semi-fragile watermarking algorithm that provides authentication and has a certain extent of robustness.

The invariant unit is called a neighborhood ball, and includes a center vertex and those vertices within a certain distance of it, as shown in figure 5.

The IIE algorithm employs two kinds of integral invariants introduced by Yang et al. (2006): an area invariant and a volume invariant. By moving the vertices within the neighborhood ball, these two integral invariants can be set to any value desired by the algorithm.

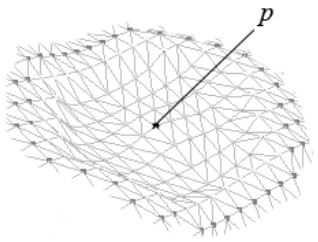


Figure 5: Invariant unit of the IIE algorithm

These integral invariants are robust against noise, giving the IIE algorithm good robustness against noise.

The invariant unit for the IIE algorithm can be determined by the vertex p at the center. The un-optimized algorithm traverses all vertices of the original model, and tests if the invariant unit determined by the vertex is valid. Once a valid invariant unit has been found, the algorithm picks the next watermark unit and embeds it. During the embedding procedure, the vertices within the neighborhood ball are moved on the surface of the sphere to set the value of the area invariant, and are moved along the normal vector to set the value of the volume invariant. The area invariant is used to embed the index, and the volume invariant is used to embed the data.

Thus, the optimized algorithm can consider all possible invariant units by considering all vertices of the original model. The invariant units used in the IIE algorithm may include about 100-200 vertices (depending on the radius of the ball), so the first optimization step is slower in this case, but since there are only about 100-400 valid invariant units (depending on the model size), the second optimization step is quicker.

4. Experimental results

In this section, we show some experimental test results. With each of the three algorithms above, we show the original model, a watermarked model obtained using the un-optimized algorithm and a watermarked model obtained using the optimized algorithm. The timings were measured on a 2.8GHz Pentium 4 running Windows XP. We used the following 3D models: Max Planck (11370 vertices, 22658 facets), the Armadillo (23201 vertices, 46398 facets) and the Bunny (34834 vertices, 69451 facets).

As stated, the first step of our optimization method improves the capacity and the second step improves the invisibility. Capacity is measured by the number of valid invariant units, and invisibility is measured by the Hausdorff distance between the original model and the watermarked model which is computed using Metro, Cignoni et al. (1998).

4.1. Results using the TSQ algorithm

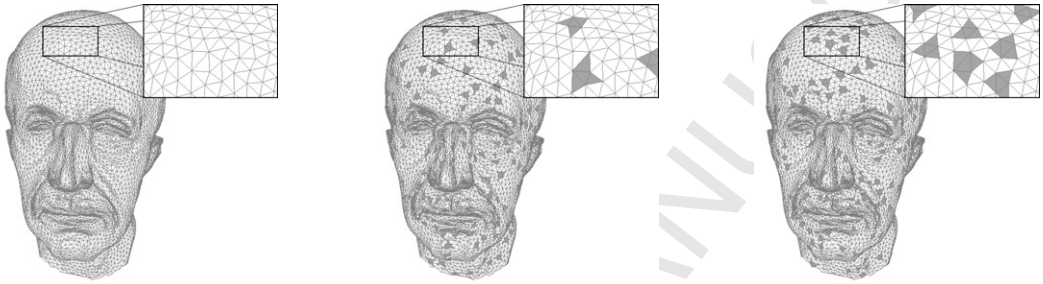
Table 1 shows comparative results between the un-optimized TSQ algorithm and the optimized TSQ algorithm. The "No." and "Opt No." rows show the number of valid invariant units in the un-optimized algorithm and the optimized algorithm respectively. The "Dis" and "Opt Dis" rows show the Hausdorff distance between the original model and the watermarked models. The "Time" row shows the time cost of the two optimization steps.

Figure 6 shows the visual results for the Max Planck model; from (a) to (c) are the original model, the watermarked model using the TSQ algorithm and the watermarked model using the optimized TSQ algorithm. In (b) and (c), the shaded triangles were modified by watermarking.

As can be seen in Figure 6, neither the un-optimized algorithm nor the optimized algorithm can arrange the invariant units tightly. This is because the embedding procedure

Table 1: Optimization using TSQ.

Model	Maxplanck	Armadillo	Bunny
No.	593	1124	1741
Opt No.	1702	3462	5337
Improvement	187%	208%	206%
Time	1.17s	3.52s	4.46s
Dis	0.01256	0.00397	0.000325
Opt Dis	0.00971	0.00194	0.000170
Improvement	22.7%	51.1%	47.7%
Time	20.6s	85.1s	176s



(a) Original model (Max Planck) (b) Watermarked model (un-optimized algorithm) (c) Watermarked model (optimized algorithm)

Figure 6: Original model and watermarked models (using TSQ).

itself of the TSQ algorithm can cause failures. One kind of failure occurs if the two ratios correspond to no real triangle, and another occurs it the longest edge of triangle M is changed after embedding. The second kind of failure often happens when the triangles of the original model are similar to equilateral triangles. The result shows that during the second optimization step, the optimized algorithm can effectively avoid such failing cases.

4.2. Results using the AIE algorithm

As in the previous section, table 2 shows comparative results, and figure 7 shows the visual results using the Armadillo model, using the AIE algorithm. In (b) and (c), the shaded facets were modified by watermarking.

As we can see in figure 7, the optimized algorithm can tighten the arrangement of invariant units. From the data in table 2, we can see that even though the optimized algorithm has embedded more information, the overall invisibility has been increased by the second optimization step.

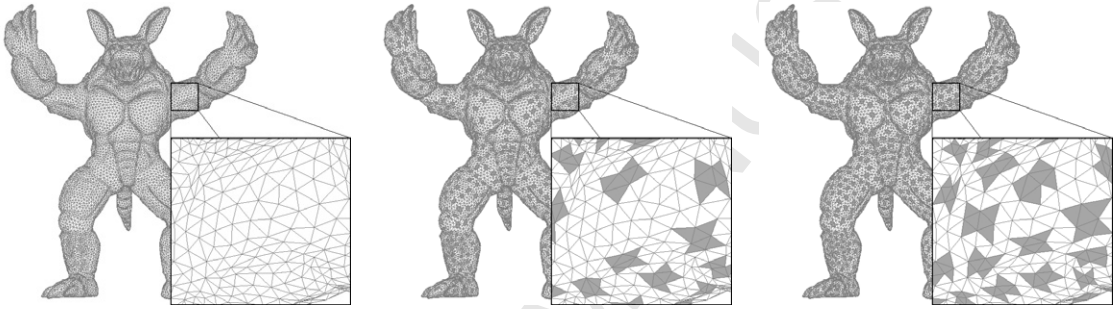
4.3. Results using the IIE algorithm

Table 3 shows comparative results, and figure 8 shows the visual results for the Bunny model, using the IIE algorithm. In (b) and (c), the shaded vertices were modified by watermarking.

htb]

Table 2: Optimization using AIE.

Model	Maxplanck	Armadillo	Bunny
No.	815	1710	2876
Opt No.	1049	2232	3444
Improvement	28.7%	30.5%	19.7%
Time	1.80s	5.04s	8.98s
Dis	0.00391	0.00663	0.001028
Opt Dis	0.00335	0.00513	0.000737
Improvement	14.3%	22.6%	28.3%
Time	7.75s	26.5s	80.4s



(a) Original model (Armadillo) (b) Watermarked model (un-optimized algorithm) (c) Watermarked model (optimized algorithm)

Figure 7: The original model and the watermarked models (using AIE).

As for the IIE algorithm figure 8 shows that the optimized algorithm tightens the arrangement of invariant units. Table 3 shows that the overall invisibility is increased by the second optimization step.

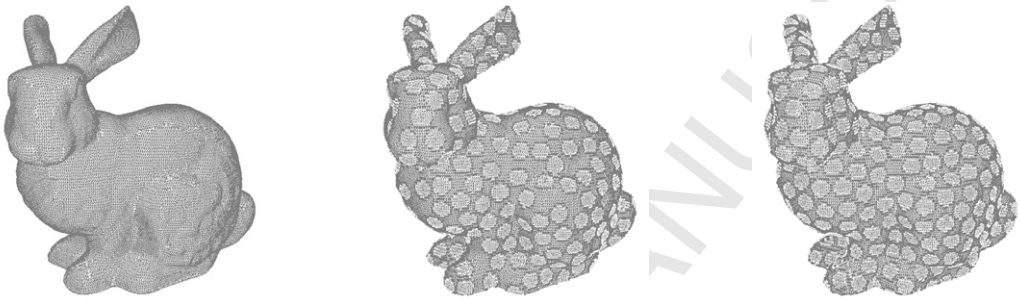
5. Conclusion

We have presented an optimization framework with two steps that can improve the capacity and invisibility of watermarking algorithms using indexed localization schemes. It can easily convert any watermarking algorithm of this kind into an optimized algorithm. By transforming the problem to a maximum independent set problem and solving it, we obtain an optimal layout of invariant units which improves the capacity of the watermarking algorithm. By solving an assignment problem, we obtain an optimal watermarking strategy under which the invisibility of watermarking is improved.

When working with existing algorithms, the number of vertices in an invariant unit greatly influences the execution time of the two optimization steps. If the number is small, the first step is quick while the second step may take more time. However, if the number is large, the first step may take more time while the second step is quick. If execution

Table 3: Optimization using IIE.

Model	Max Planck	Armadillo	Bunny
No.	129	208	331
Opt No.	140	287	424
Improvement	8.5%	38.0%	28.1%
Time	36.9s	34.7s	40.7s
Dis	0.01414	0.01057	0.000549
Opt Dis	0.01310	0.00847	0.000481
Improvement	7.4%	19.9%	12.4%
Time	23ms	278ms	1123ms



(a) Original model (Bunny) (b) Watermarked model (un-optimized algorithm) (c) Watermarked model (optimized algorithm)

Figure 8: The original model and the watermarked models (using IIE).

time is crucial, we could omit the most expensive of these two steps in order to improve performance.

We using a coarse invisibility evaluation function. If in future, and standard is agreed upon for this function, it could easily be used to provide optimal results.

Furthermore, every watermarking algorithm has parameters. When deciding a set of parameters, there is a trade-off between invisibility, capacity and robustness. Improved capacity and invisibility allows algorithms to change the parameters to get better robustness. Therefore the overall performance of watermarking algorithms could be improved, so we believe that our method is a useful step for watermarking algorithms employing indexed localization schemes.

Acknowledgements

This work was supported by the National Basic Research Project of China (Project Number 2006CB303104) and the Natural Science Foundation of China (Project Number 60673004). The models used in this paper are the courtesy of Stanford University and AIM@SHAPE shape repository.

References

Alekseyev, S., 2003. Volumetric Surface Reconstruction with Level Set Methods. Diploma Thesis, RWTH Aachen University, Germany.

Aspert, N., Santa-Cruz, D., Ebrahimi, T., 2002. Mesh: Measuring error between surfaces using the hausdorff distance. In: Proceedings of the IEEE International Conference on Multimedia and Expo. Vol. 1. pp. 705–708.

Benedens, O., 2000. Affine invariant watermarks for 3d polygonal and nurbs based models. In: Proceedings of the 3rd International Workshop Information Security, Wollongong. pp. 15–29.

Benedens, O., Busch, C., 2000. Towards blind detection of robust watermarks in polygonal models. In: Proceedings of Eurographics 2000. pp. 199–208.

Besl, P., McKay, J., 1992. A method for registration of 3-d shapes. IEEE Trans. Patt. Recog. Mach. Intell. 14 (2), 239–255.

Bian, Z., Hu, S.-M., Martin, R. R., 2009. Evaluation for small visual difference between conforming meshes on strain field. Journal of Computer Science and Technology 24 (1), 65–75.

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

Chen, Y., Medioni, G., Apr. 1992. Object modelling by registration of multiple range images. Image and Vision Computing 10 (3), 145–155.

Chou, C.-M., Tseng, D.-C., 2006. A public fragile watermarking scheme for 3d model authentication. Computer-Aided Design 38 (11), 1154–1165.

Cignoni, P., Rocchini, C., Scopigno, R., 1998. Metro: Meaturing error on simplified surfaces. Computer Graphics Forum 17 (2), 167–174.

Corsini, M., Gelasca, E. D., Ebrahimi, T., Barni, M., 2007. Watermarked 3-d mesh quality assessment. IEEE Transactions on Multimedia 9 (2), 247–256.

Craver, S., Memon, N., Yeo, B.-L., Yeung, M.-M., 1998. Resolving rightful ownerships with invisible watermarking techniques: Limitations, attacks, and implications. IEEE Journal on Selected Areas in Communications 16 (4), 573–586.

Kanai, S., Date, H., Kishinami, T., 1998. Digital watermarking for 3d polygons using multiresolution wavelet decomposition. In: Proceedings of International Workshop on Geometric Modeling. pp. 296–307.

Lavoué, G., Gelasca, E. D., Duport, F., Baskurt, A., Ebrahimi, T., 2006. Perceptually driven 3d distance metrics with application to watermarking. In: Proceedings of SPIE. Vol. 6312.

Leguizamon, G., Michalewicz, Z., 2001. A ant system for the maximum independent set problem. In: Proceedings of the VII Argentinian Congress on Computer Science, Santa Cruz, Argentina. Vol. 2. pp. 1027–1040.

Li, L., Zhang, D., Pan, Z., 2004. Watermarking 3d mesh by spherical parameterization. Computers & Graphics 28 (6), 981–989.

Nielson, G., Foley, T., 1989. A Survey of Applications of an Affine Invariant Norm. Academic Press Professional, Inc. San Diego, CA, USA.

Ohbuchi, R., Masuda, H., Aono, M., 1997a. Embedding data in 3d mmodels. In: Proceedings of the European Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services 97. pp. 1–10.

Ohbuchi, R., Masuda, H., Aono, M., 1997b. Watermarking three dimensional polygonal models. In: Proceedings of ACM Multimedia. pp. 261–272.

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

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

Praun, E., Hoppe, H., Finkelstein, A., 1999. Robust mesh watermarking. In: ACM SIGGRAPH on Computer Graphics Proceedings. pp. 325–334.

Uccheddu, F., Corsini, M., Barni, M., 2004. Wavelet-based blind watermarking of 3d models. In: Proceedings of the 2004 Multimedia and Security Workshop. pp. 143–154.

Wang, Y.-P., Hu, S.-M., 2009. A new watermarking method of 3d models based on integral invariants. IEEE Transactions on Visualization and Computer Graphics 15 (2), 285–294.

Wang, Z., Bovik, A., Sheikh, H., Simoncelli, E., 2004. Image quality assessment: from error visibility to structural similarity. IEEE Transactions on Image Processing 13 (4), 1C14.

Wu, J.-H., Kobbelt, L., 2005. Efficient spectral watermarking of large meshes with orthogonal basis functions. Visual Computer 21 (8-10), 848–857.

Yang, Y.-L., Lai, Y.-K., Hu, S.-M., Pottmann, H., 2006. Robust principal curvatures on multiple scales. In: Proceedings of 4th Eurographics Symposium on Geometry Processing, Eurographics Association. pp. 223–226.

Yeo, B.-L., Yeung, M.-M., 1999. Watermarking 3d objects for verification. IEEE Computer Graphics and Applications 19 (1), 36–45.