Permutation Steganography for Polygonal Meshes Based on Coding Tree

Shihchun Tu, Hungwei Hsu and Wenkai Tai

Department of Computer Science and Information Engineering, National Dong Hwa University, Taiwan, Republic of China

***Abstract*—We propose a coding tree for permutation steganography in polygonal meshes. Instead of conceptually building a complete binary tree for remaining embedding/extracting primitives to encode/decode bitstream, our method lengthens bitstream by exploiting the coding tree of the primitives, a binary tree which is a little bit skewed.**

The coding tree has higher levels than the complete binary tree so that as experimental results show, the average capacity of our approach is more about 0.39 bit/vertex than that of Bogomjakov et al. [5], and the improvement of the maximal capacity comes up to 40% of Bogomjakov et al. [5]. Still, our method has the minimal capacity same with Bogomjakov et al. [5], and is simple and easy to implement as well. This ordering is obtained based on the mesh connectivity [14], making our method robust to geometric affine transformations. Also, the primitive rearrangement does not distort the cover media and cause suspicion of the hidden message.

***Index Terms*—Coding Tree, Permutation Steganography, Data Hiding.**

I. INTRODUCTION

Steganography for 3D polygonal meshes was pioneered by Ohbuchi et al. [11], who introduced watermarking on 3D polygonal meshes. Since then several ideas of steganography have been proposed to modify and/or improve to previous work. Some methods, [1, 3, 6, 7, 8, 11, 16, 18], work in the spatial domain, while others, [9, 12, 13], work in the spectral domain. Transforming the model to the spectral domain and applying the data embedding and extraction operations in this domain improve the robustness of the resultant stego model. The processing time of these methods for embedding and extraction are relatively high, due to the domain transformation process. Furthermore, these methods have limited capacity. Therefore, these methods are more appropriate for data protection applications, such as watermarking, than for data hiding. In contrast, methods that work on the spatial domain have better capacity and efficiency but with weaker robustness.

Most current methods work for 3D polygonal meshes. Several methods, such as [9, 13, 17], utilize 3D models defined as point sets. Polygonal meshes provide fewer vertices than point set models, but have face information that can be used as the alternative medium. Higher vertex numbers allow a model to hide more information, but require more space and computing hide more information, but require more space and

Manuscript received on 20 June 2010
Email: wktai@mail.ndhu.edu.tw

computing power to handle. Therefore, many applications use polygonal meshes rather than point set models.

Permutation steganography, [2, 10, 15], giving optimal capacity, hides information by exploiting the difference between the arrangement of embedding primitives of a set and a known reference ordering of a set. Permutation steganography is of the optimal capacity, up to O(log(n!)) = O(nlogn) bits, which is much better than the results of the previous work for 3D polygonal meshes but at the expense of computation time $\Omega(n^2 \log^2 n \log\log n)$. Recently, one proposed method, Bogomjakov et al. [5], is simple to implement and perform efficiently, O(n). The method guarantees the minimal capacity, one bit per element less than the theoretical optimum, and is robust and resistant to any kind of attacks on the polygonal mesh because the reference ordering is obtained by using the traversal of Edgebreaker mesh compression algorithm [14] based on the mesh connectivity alone.

The idea of Bogomjakov et al. [5] is to maximize the bitstream length while encoding each embedding primitive using as the larger index of an embedding primitive in the reference ordering as possible. The number of the maximal embedding bits is about k + 1, $k = \lfloor \log_2 n \rfloor$ where n is the number of the embedding primitives. Namely, for those embedding primitives in the reference ordering with index not less than 2k but less than n could be embedded k + 1 bits. Eventually, there are 2k+1 different values in k + 1 bits. Hence, the chance of encoding an embedding primitive in maximal bits is $(n - 2^{\lfloor \log_2 (n-1) \rfloor}) / 2^{\lceil \log_2 n \rceil}$ for n > 2. It is clearly that all bitstream with maximal length are of the most significant bit "1" in Bogomjakov et al. [5]. Eventually, the method could be treated as making use of the binary search tree to encode and decode bitstream for primitives. There are two kinds of nodes: branch nodes and leaf (embedding primitive) nodes which are conceptually built for the remaining primitives in the reference ordering while encoding and decoding. If we interpret a zero as a left branch and a one as a right branch, then the binary bits peeked from the embedding message determine the branching needed at each level of the tree to reach the embedding primitive when searching the tree. On the contrary, given an extracting primitive we can trace the branching and determine the binary bits when decoding.

In this paper, we propose a binary tree as the coding tree, but the tree is a little bit skewed. When branching to the skewed end, the encoded and decoded bitstream will be longer in length than the corresponding complete binary tree of n remaining primitive nodes. Still, our method is of the same minimal capacity with Bogomjakov et al. [5], but the average capacity of our method is larger than that of Bogomjakov et al. [5], and the maximal capacity is much improved by the proposed coding tree, showing the improvement of the maximal capacity up to 40% of Bogomjakov et al. [5].

Our method is robust to attacks, geometric affine transformations, on the geometry because the reference ordering of the embedding primitives is derived from a canonical traversal of the connectivity [14]. The imperceptibility of the hidden message is one of the key requirements in steganography. As permutation steganography permutes embedding primitives relative to a canonical reference order, the stego polygonal mesh is nothing different from the input mesh except for the primitives present in a different order. Others will not discern the presence of the hidden message. Furthermore, the primitive rearrangement also makes our method distortion-free.

The rest of this paper is structured as follows. Section II describes the proposed method. Capacity analysis is specified in Section Ⅲ. Experimental results are shown in Section Ⅳ. Finally, we conclude this work and point out possible future work in Section Ⅴ.

II. PROPOSED METHOD

2.1 Reference Ordering

Permutation steganography hides the message in a cover media by rearranging the order of embedding primitives in the media. A canonical reference ordering is so essential that a permutation is well-defined with respect to the reference ordering. Edgebreaker algorithm proposed by Rossignac [14] is exploited to obtain the unique reference ordering from triangle meshes by giving the initial vertex and edge before embedding and extraction proceeds. Note that there are alternatives in Bogomjakov et al. [5] for general polygonal meshes to obtain the unique reference ordering.

Same as Bogomjakov et al. [5] the first edge of the first polygon in the cover polygonal mesh is selected as the initial vertex and edge, and the polygon is excluded from embedding. And then, we take the first edge of the first polygon in the stego polygonal mesh to perform the traversal and obtain the same reference ordering. In general, the file of polygonal mesh contains two kinds of primitives. The vertex primitive describes the coordinates of vertices and the polygon primitive specifies the mesh connectivity. Two reference orderings, one for each type of primitive, are required because both primitives are used to hide message.

2.2 Coding Tree

A binary tree could be treated as a coding tree. Each bit in a bitstream is used to branch (traverse) the coding tree, and the bitstream is encoded in the reached leaf node. The number of the encoded bits is the length of the path from the root to a leaf node. The coding tree containing n leaf nodes is recursively defined as

$$T(n) = T(l) + T(r)$$

where $T(n)$ represents the tree T having n leaf nodes, and $T(l)$ and $T(r)$ represent the left subtree having l leaf nodes and the right subtree having r leaf nodes respectively. $T(l)$ and $T(r)$ are of course coding trees. Also the number of leaf nodes in the tree is constrained as follows:

$$n = l + r, r = 2^{\lfloor \log_2 n \rfloor - 1}, n \geq 2$$

For example, one sample coding tree with 7 leaf nodes is shown as Figure 1 where T(7) = T(5) + T(2), T(5) = T(3) + T(2), T(3) = T(2) + T(1), and T(2) = T(1) + T(1). The coding tree defined in this way is meant to have the following properties:

- The left subtree with respect to its parent is a sort of skewed tree that helps lengthen the bitstream, namely increasing the maximal capacity as far as possible.
- The right subtree with respect to its parent is a full binary tree that keeps the minimal capacity of embedding message the same as the complete binary tree.
- The shortest path length of the left subtree is not less than the longest path length of the corresponding right subtree.
- The number of the leaf nodes in the left subtree is not less than that of the nodes in the corresponding right subtree.
- The left and right subtrees have at least one leaf node.

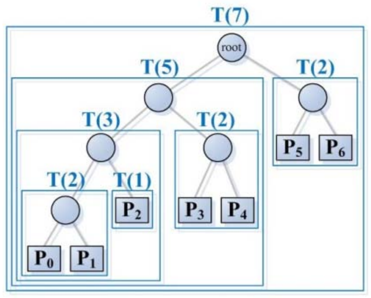


Fig. 1. One sample coding tree with 7 leaf nodes

2.3 Embedding Message

The embedding message is divided into two parts. Our embedding procedures hide one part of message in the vertex primitive by rearranging the vertices and another part of message in the polygon primitive by rearranging the polygons. The same embedding procedures apply to both of the vertex and face primitives.

A primitive at position p in the reference ordering is selected to embed the next h bits in the embedding message if p, non-negative integer, is equal to the integer value of the h bits and the extended binary traversal stops at the leaf node p according to the next h bit-stream. In Fig. 2, the procedures of the Encoder and RecursiveEncoder are applied in embedding message stage. The Encoder procedure is to rearrange (output) primitive according to the position (leaf node) visited by the recursive traversal of the RecusiveEncoder procedure.

Assume that there are n primitives in the initial reference ordering. At each step i in the Encoder procedure, we choose a primitive at position p, which is the returned value of the RecursiveEncoder procedure, from the remaining n − i primitives of the reference ordering and output it as the next primitive of the permutation. The output primitive is removed by replacing it with the current last primitive in the reference order so that the remaining n – i − 1 primitives are sequentially stored in the array. The position p is the index of the leaf node which is determined by the result of our coding tree traversal according to the next h bits where h is the external path length from root to a leaf node. From left to right each bit of the next h bits is used to branch the traversal. If the message bit is zero, the left child will be traversed, otherwise the right child will be traversed. The traversal continues until the number of the leaf nodes is only one left. After stopping the recurrence,
the position of the visited leaf node is computed level by level all the way back to the root. If the visited node is the right child with respect to its parent, then its local position must plus the number of the leaf nodes in left subtree to obtain the local position in each level. Therefore, position p is the index of the leaf node in the tree with respect to the next h bits in the embedding message. Fig. 2 shows the proposed embedding algorithm where n is the number of embedding primitives in the embedding primitive set P, k is the index of the message bitstream. Also, we illustrate the computation of position p as shown in Fig. 3.

```
Procedure Encoder
Input: a bitstream embedding message and a list of embedding primitives.
Output: the permutation array perm[].
begin
    $n = ||P||$;
    $k = 0$;
    // compute the reference ordering of embedding primitives
    ref[] = compRefOdr(P);
    for i = 0 to $n – 1$
        p = RecusriveEncoder($n – i$, k);
        perm[i] = ref[p];  // output primitive at p to permutation
        ref[p] = ref[$n – i – 1$];  // replace with last primitive
    end
end

Procedure RecursiveEncoder(n, k)
Input: the number of the embedding primitives.
Output: the position p, the index of the leaf node.
begin
    // the condition, $n = 1$, stops the recurrence call.
    if ($n = 1$) then return(0) end
    // nr and nl are the number of nodes in
    // left and right subtrees respectively
    $nr = 2^{\lfloor \log_2 n \rfloor - 1}$;
    $nl = n – nr$;
    if (msg[k] = 1) then
        $k = k + 1$;    // advance one bit
        p = RecursiveEncoder(nr, k);
        return($p + nl$);
    else
        $k = k + 1$;
        p = RecursiveEncoder(nl, k);
        return(p);
    end
end
```

Fig. 2. Embedding algorithm

2.4 Message Extraction

Assume that there are n primitives in the initial permutation in the stego polygonal mesh. At each step i, we choose the next primitive from the permutation to extract the embedded message bits. The position p from the remaining n – i primitives of the reference ordering that has the same primitive with the primitive from the permutation is the value of being output bitstream. Inspired by Bogomjakov et al. [5] again, to efficiently speedup the position computation, we extend the primitive record with a ref field that keeps the current position of the primitive in the reference ordering.

```
Input: the permutation of primitives in the stego polygonal mesh.
Output: the extracted bitstream.
begin
    perm[] = P;    // place P into perm[] array
    $n = ||P||$;
    // compute the reference ordering of primitives
    ref[] = compRefOdr(P);
    // initial position in ref array
    for $i = 0$ to $n – 1$
        ref[i].ref = i;
    end
    for $i = 0$ to $n – 1$
        // get primitive position in ref array
        p_orig = p = perm[i].ref;
        $nnode = n – i$;
        while ($nnode > 1$)
            // nr and nl are the number of leaf nodes in
            // subtrees  respectively
            $nr = 2^{\lfloor \log_2 nnode \rfloor - 1}$;
            $nl = nnode – nr$;
            if ($nl < (p + 1)$)     // traverse the right subtree
                output(1);          // output message bit 1
                $p = p – nl$;
                $nnode = nr$;
            else                    // traverse the left subtree
                output(0);          // output message bit 0
                $nnode = nl$;
            end
            // update reference ordering
            ref[p_orig] = ref[$n – i – 1$];
            ref[p_orig].ref = p_orig;
        end
    end
end
```

Fig. 3. Extraction algorithm

The embedded bits with respect to the remaining primitives at position p are extracted level by level from root to the leaf node p, and one level extracts one message bit. If the number (position) p is less than the number of the leaf nodes in the left subtree, position p is a leaf node in the left subtree. Hence we output message bit '0' and traverse the left child, otherwise position p is a leaf node in the right subtree and we output message bit '1'. Then, we update the number p by subtracting the number of the leaf nodes in the corresponding left subtree from it to have the local relative position of p in the right subtree. The extraction process recursively traverses the tree until the number of the leaf nodes is only one node left. Fig. 4 shows the proposed extraction algorithm where P is the set of primitves in the stego polygonal mesh, n is the number of primitives in the initial permutation. We illustrate the bitstream extraction of position p as shown in Fig. 5.

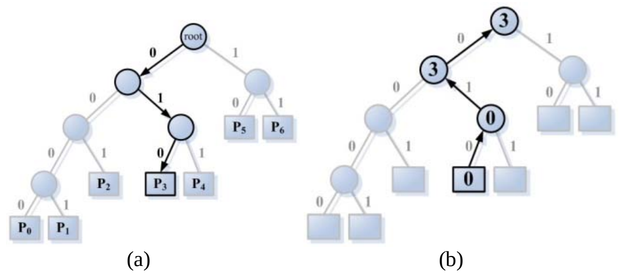


Fig. 4. Illustration of the computation of position p in embedding message stage. Assume the next message bitstream is "010". (a) The black lines represent the order of calling RecursiveEncoder that is RecursiveEncoder(5), RecursiveEncoder(2) and RecursiveEncoder(1), and the reached leaf node is P3. (b) The black lines represent the order of evaluating the positions of visited nodes in the tree, that is 0, 0 + 3 = 3 and 3

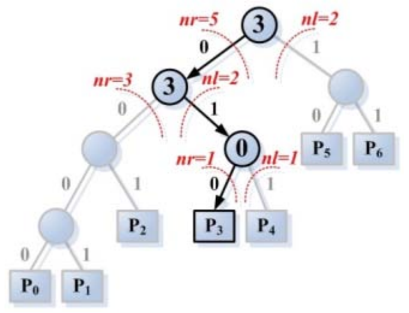


Fig. 5. Illustration of the computation of position p in extraction message stage. When extracting message, the black lines represent the order of extracting message: (1) p = 3 < 5, outputs "0". (2) p = 3 ≥ 3, outputs "1", p = 3 − 3 = 0. (3) p = 0 < 1, outputs "0"

III. CAPACITY

There are at least $\lfloor \log_2 (n-i) \rfloor$ bits embedded at each step i of the embedding procedure. Hence, given n embedding primitives, our method has the same minimal capacity C(n) with Bogomjakov et al. [5], C(n) ≥ log2n! – n + 1. Namely, our method is one bit per primitive less than the theoretical optimum, log2n!, of the standard permutation steganography.

Upon the best case for embedding message, the maximal possible capacity of Bogomjakov et al. [5] is about $\sum_{i=1}^{n} \lceil \log_2 i \rceil > \log_2 n!$. However, in our method, the height of the left subtree of the coding tree is not less than that of the right subtree. The maximal capacity of our method is

$$\sum_{i=1}^{n} H(T(i)) = \sum_{i=1}^{n} Max(H(T(i-2^{\lfloor \log_2 i \rfloor -1})), H(T(2^{\lfloor \log_2 i \rfloor -1}))) + 1$$
$$= \sum_{i=1}^{n} H(T(i-2^{\lfloor \log_2 i \rfloor -1})) + 1$$

where function H(T(i)) is the height of the tree T(i), Max() returns the larger one, $T(i-2^{\lfloor \log_2 i \rfloor -1})$ is the left subtree and $T(2^{\lfloor \log_2 i \rfloor -1})$ is the right subtree. As a comparison, the measurement of the maximal capacity is shown in Fig. 6. In Fig.6, as you can see, our method has the largest maximal capacity. Also note that the more embedding primitives the even more message bits embedded in by our method.

On average, the expectation capacity is

$$E\{C(n)\} = \sum_{i=1}^{n} \{p_i \times h_i + (1-p_i)(h_i - 1)\}$$

where h_i is the height of the coding tree regarding to the remaining primitives and pi is the possibility that hi bits can be embedded in the primitive. In Bogomjakov et al. [5], the number of remaining primitives that can be embedded hi bits is $i - 2^{\lfloor \log_2 (i-1) \rfloor}$, the possibility p_i is

$$p_i = \frac{i - 2^{\lfloor \log_2 (i-1) \rfloor}}{2^{h_i}}, i > 2$$

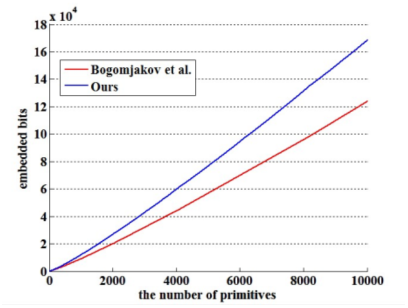
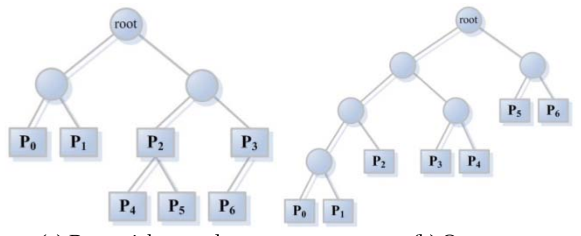


Fig. 6. The measurement of the maximal capacity of Bogomjakov et al. [5] and our method



(a) Bogomjakov et al. (b) Ours

Fig. 7. One sample of average capacity computation for 7 embedding primitives. The average capacity of Bogomjakov et al. [5] and our method are E{C(7)} = {3 × 3 / 8 + 2 × 1 / 8 + 2 × 2 / 4} + E{C(6)} = 10.025, and E{C(7)} = {4 × 2 / 16 + 3 × 3 / 8 + 2 × 2 / 4} + E{C(6)} = 11.875 respectively.

In our method, the possibility pi is $1/2^{h_i}$ where hi is the path length from root to the leaf node. Without loss of generality, Fig.

7 illustrates one sample of average capacity computation when given 7 embedding primitives. We will also show the experimentally measured capacity in the next section.

TABLE 1: THE COMPARISON OF THE MINIMAL CAPACITY OF BOGOMJAKOV ET AL. [5] AND OUR METHOD FOR POLYGONAL MESHES WITH DIFFERENT SIZES

Polygonal mesh			Minimal Capacity [bits]		
name	#verts	#faces	Bogomjakov et al.	Ours	bpv
Cow	2,904	5,804	89,331	89,331	30.76
Fandisk	6,475	12,946	221,451	221,451	34.20
Horse	48,485	96,966	2,082,158	2,082,158	42.94
Armadillo	172,974	345,944	8,381,157	8,381,157	48.45
Isis	187,644	375,284	9,158,667	9,158,667	48.81

TABLE 2: THE COMPARISON OF THE MAXIMAL CAPACITY OF BOGOMJAKOV ET AL. [5] AND OUR METHOD

Polygonal mesh			Maximal Capacity bits(bpv)		
name	#verts	#faces	Bogomjakov et al.	Ours	Improved %
Cow	2,904	5,804	98,014(33.75)	131,952(45.44)	34.63%
Fandisk	6,475	12,946	240,845(37.20)	328,099(50.67)	36.23%
Horse	48,485	96,966	2,227,576(45.94)	3,094,635(63.83)	38.92%
Armadillo	172,974	345,944	8,900,038(51.45)	12,444,121(71.94)	39.82%
Isis	187,644	375,284	9,721,558(51.81)	13,610,847(72.54)	40.01%

TABLE 3: THE COMPARISON OF THE AVERAGE CAPACITY. NOTE THAT OVER 1000 RANDOMLY GENERATED EMBEDDING MESSAGES ARE USED TO MEASURE THE AVERAGE CAPACITY AND THE LAST COLUMN SHOWS THE IMPROVED BPV

Polygonal mesh			Average Capacity bits(bpv)		
name	#verts	#faces	Bogomjakov et al.	Ours	Improved
Cow	2,904	5,804	91,134(31.38)	92,252(31.77)	0.39
Fandisk	6,475	12,946	225,513(34.83)	228,082(35.23)	0.40
Horse	48,485	96,966	2,112,572(43.57)	2,130,975(43.95)	0.38
Armadillo	172,974	345,944	8,489,806(49.08)	8,556,428(49.47)	0.39
Isis	187,644	375,284	9,275,981(49.43)	9,347,650(49.81)	0.38

TABLE 4: TIMING STATISTICS IN MILLISECONDS FOR EMBEDDING AND EXTRACTION ALGORITHMS OF OURS, AND THAT OF BOGOMJAKOV ET AL. [5] FOR DIFFERENT POLYGONAL MESHES

Polygonal mesh			Timings [msecs]			
			Embbeding		Extraction	
name	#verts	#faces	Bogomjakov et al.	Ours	Bogomjakov et al.	Ours
Cow	2,904	5,804	2	25	1	26
Fandisk	6,475	12,946	4	62	2	65
Horse	48,485	96,966	33	601	20	616
Armadillo	172,974	345,944	126	2439	82	2482
Isis	187,644	375,284	139	2642	90	2695

IV. EXPERIMENTAL RESULTS

All experiments were performed with several polygonal meshes of different sizes on a PC with an Intel Core 2 1.87GHz processor and 2GB main memory to verify and evaluate our embedding and extraction algorithms. The comparison of the minimal, maximal and average capacity and timing statistics of Bogomjakov et al. [5] and our method are summarized in Table 1, 2, 3 and 4 respectively.

As Table 1 shows, the minimal capacity of our method is the same with that of Bogomjakov et al. [5] because the minimal capability of all methods is guaranteed, one bit per element less than the theoretical optimum, and is computed from the lower bound as $\sum_{i=2}^{n}\lfloor \log_2 i \rfloor + \sum_{i=2}^{m}\lfloor \log_2 i \rfloor$ where n and m are the number of vertices and faces respectively.

As you can see in Table 2, the maximal capacity of our method is greater than that of Bogomjakov et al. [5]. Also note that the larger the mesh size the higher percentage of the bit/vertex improved by our approach. For instance, the largest testing polygonal mesh, Isis, achieves capacity over 72.54 bit/vertex and improved percentage of the bit/vertex up to 40.01%.

Table 3 shows the comparison of the average capacity of Bogomjakov et al. [5] and our method for polygonal meshes with different sizes. Over 1000 randomly generated embedding messages are used to measure the average capacity. As you can see, the average capacity of our method is also larger than that of Bogomjakov et al. [5]. Actually, the improved average bit/vertex is about 0.39 bit/vertex over Bogomjakov et al. [5].

Table 4 shows timing statistics in milliseconds for embedding and extraction algorithms of ours and those of Bogomjakov et al. [5]. Still over 1000 runs are used to measure the timing. As you can see, the running time of our approach is slower than that of Bogomjakov et al. [5], because the time complexity of our approach is O(nlogn) and the time complexity of Bogomjakov et al. [5] is O(n).

V. CONCLUSION

We have presented a method for permutation steganography in polygonal meshes based on exploiting the binary tree representation to the position of the primitives in the reference ordering. The proposed coding tree with skew property helps lengthen the embedding bitstream. The average capacity of our approach is more about 0.39 bit/vertex than that of Bogomjakov et al. [5]. Besides, the maximal capacity of our method comes up to over 72.54 bit/vertex, greatly improved up to 40% of Bogomjakov et al. [5].

Permutation steganography is to rearrange the primitives in the cover media. Although the rendering performance might be affected by the rearrangement [4], the geometry and file size keep the same in stego media. In this way, the polygonal meshes will not be distorted and the arrangement of primitives in the stego media will not make anyone suspicious of hidden message. Moreover, the reference ordering from the polygonal mesh traversal [14] is based on the mesh connectivity, making our method robust to any geometric affine transformations.

In the future, we would like to investigate the possibility of embedding more than one permutation in the cover media. Namely, the primitive arrangement represents more than one primitive permutation. The optimal capacity of permutation steganography is log2n!. Hopefully, we can achieve klog2n!, k > 1.

REFERENCES

[1] N. Aspert, E. Drelie, Y. Maret and T. Ebrahimi. Steganography for Three-Dimensional Polygonal Meshes, In: *SPIE 47th Annual Meeting, Seattle,* WA, July 7–11, pp. 705–708, 2002.

[2] D. Artz. Digital Steganography: Hiding Data within Data, *IEEE Internet Computing*, 5(2), pp. 75–80, 2001.

[3] O. Benedens. Geometry-based watermarking of 3D models. *IEEE Computer Graphics and Applications*, 19(1), pp. 46–55, 1999.

[4] A. Bogomjakov and C. Gotsman. Universal rendering sequences for transparent vertex caching of progressive meshes. *Computer Graphics Forum*, 21(2), pp. 137—148, 2002.

[5] A. Bogomjakov, C. Gotsman and M. Isenburg. Distortion-Free Steganography for Polygonal Meshes. *Computer Graphics Forum*, 27(2), pp. 637—642, 2008.

[6] J. W. Cho, R. Prost and H. Y. Jung. An Oblivious Watermarking for 3-D Polygonal Meshes Using Distribution of Vertex Norms. *IEEE Transactions on Signal Processing*, 55(1), pp. 142—155, 2007.

[7] Y. M. Cheng and C. M. Wang. A High-Capacity Steganographic Approach for 3D Polygonal Meshes. *The Visual Computer*, 22(9), pp. 845—855, 2006.

[8] Y. M. Cheng and C. M. Wang.: An Adaptive Steganographic Algorithm for 3D Polygonal Meshes. *The Visual Computer*, 23(9-11), pp. 721—732, 2007.

[9] D. Cotting, T. Weyrich, M. Pauly and M. Gross. Robust Watermarking of Point-Sampled Geometry. In: *Proc. of International Conference on Shape Modeling and Applications, Genova, Italy*, June 7--9, pp. 233—242, 2004.

[10] M. Kwan. Gifshufflee. *http://www.darkside.com.au/gifshuffle/*

[11] R. Ohbuchi and M. Aono. Watermarking Three-Dimensional Polygonal Model. In: P*roceedings of the fifth ACM international conference on Multimedia, Seattle, WA,* Nov. 9--13, pp. 261—272, 1997.

[12] R. Ohbuchi R, A. Mukaiyama and S. Takahashi. A Frequency-Domain Approach to Watermarking 3D Shapes. *Computer Graphics Forum*, 21(3), pp. 373—382, 2002.

[13] R. Ohbuchi, A. Mukaiyama and S. Takahashi. Watermarking Three-Dimensional Polygonal Model. In: 3*rd International Conference on Cyberworlds, Tokyo, Japan,* Nov. 18--20, pp. 392—399, 2004.

[14] J. Rossignac. Edgebreaker: Connectivity Compression for Triangle Meshes. *IEEE Transactions on Visualization and Computer Graphics*, 5(1), pp. 47—61, 1999.

[15] D. Glaude. Hiding data into the palette of a GIF file. *http://users.skynet.be/glu/sgpo.htm.*

[16] C. M. Wang and Y. M. Cheng. An Efficient Information Hiding Algorithm for Polygon Models. *Computer Graphics Forum*, 24(3), pp. 591—600, 2005.

[17] C. M. Wang and P. C. Wang. Steganography on Point-Sampled Geometry. *Computers & Graphics*, 30(2), pp. 244—254, 2006.

[18] S. Zafeiriou, A. Tefas and I. Pitas. Blind Robust Watermarking Schemes for Copyright Protection of 3D Mesh Objects. *IEEE Transactions on Visualization and Computer Graphics*, 11(5), pp. 596—607, 2005.

Shihchun Tu is currently a doctoral student of Computer Science and Information Engineering department at National Dong Hwa University, Taiwan, R.O.C. He received his B.S. in Applied Mathematics from the Chinese Culture University in 1989 and his M.S. degree in Applied Mathematics from National Sun Yat-sen University in 1995. His research interests include data hiding, 3D modeling, and terrain rendering.

Hungwei Hsu is currently a doctoral student of Computer Science and Information Engineering department at National Dong Hwa University, Taiwan, R.O.C.

Wenkai Tai is currently an associate professor of Computer Science and Information Engineering department at National Dong Hwa University, Taiwan, R.O.C. He received his M.S. and Ph.D. degree in Computer Science from National Chiao Tung University in 1989 and 1995 respectively. His current research interests include real-time rendering, visibility, and techniques for gaming.