EUROGRAPHICS 2008 / G. Drettakis and R. Scopigno
(Guest Editors)

Volume 27 (2008), Number 2

Distortion-Free Steganography for Polygonal Meshes

Alexander Bogomjakov[1] Craig Gotsman[1] Martin Isenburg[2]

[1] Center for Graphics and Geometric Computing – Technion – Israel Institute of Technology
[2] Center for Applied Scientific Computing – Lawrence Livermore National Labs

Abstract

We present a technique for steganography in polygonal meshes. Our method hides a message in the indexed representation of a mesh by permuting the order in which faces and vertices are stored. The permutation is relative to a reference ordering that encoder and decoder derive from the mesh connectivity in a consistent manner. Our method is distortion-free because it does not modify the geometry of the mesh. Compared to previous steganographic methods for polygonal meshes our capacity is up to an order of magnitude better.

Our steganography algorithm is universal and can be used instead of the standard permutation steganography algorithm on arbitrary datasets. The standard algorithm runs in $\Omega(n^2 \log^2 n \log\log n)$ *time and achieves optimal* $O(n \log n)$ *bit capacity on datasets with n elements. In contrast, our algorithm runs in* $O(n)$ *time, achieves a capacity that is only one bit per element less than optimal, and is extremely simple to implement.*

Categories and Subject Descriptors (according to ACM CCS): I.3.m [Computer Graphics]: Miscellaneous

1. Introduction

Steganography (or, more simply, data-hiding) is the science of hiding messages in media in such a way that even the existence of the message remains undetected to all but the recipient. This is in contrast with cryptography, where the fact that a message is hidden in the data is not disguised, but it may be retrieved only by the use of a secret key, typically known only to the recipient. Thus, steganographic messages do not attract attention to themselves, to messengers, or to recipients. A classic example is invisible ink that turns brown when the paper is heated. An inconspicuous cover message is important as a blank sheet of paper can arouse suspicion.

The digital equivalent of invisible ink is software that embeds secret data inside another file - the carrier. A silly snapshot we receive from a colleague via unsecure email, for example, could easily hide the root password to his computer. Image, video, and audio data have been the predominant carrier for digital steganography—especially in the context of digital watermarking and copyright protection. In recent years researchers have turned their attention to the emerging media of 3D datasets. This is the subject of our paper.

Once it has been established that a particular media can carry steganographic content, the issue is capacity and complexity - how large a message can be hidden in a given data set and at what runtime cost (for encoding and decoding).

Traditional steganographic methods try to hide information in the noise of the data. They distort the original data just enough to embed a message but without this distortion being noticable. They may, for example, modify the least significant bits of individual samples in an image, a video, or an audio track. A common approach for 3D data is to slightly perturb the vertex positions. Such a geometric modification can be done directly in the spatial domain [OMA97, CDSM04, ADME02, ME04, WC05, WW06, PHF99], which is simple and efficient but also prone to robustness issues. Better robustness can be achieved by spectral methods [KDK98, OTMM01, OMT04, CWPG04]. However, computing a spectral representation is expensive, and generally provides lower capacity. These steganographic techniques are lossy because they affect the integrity of the data.

Recently, researchers have started to exploit the flexibility in representing polygon meshes to gain additional capacity. Wang and Cheng [WC05] used alternating CW and CCW orders of face vertices to encode an additional bit per triangle. Cheng and Wang [CW06] describe a connectivity embedding scheme which increases the capacity by 6 bits per vertex. They rearrange vertices and faces relative to a reference ordering derived from the mesh geometry. The position of a vertex or a face in the new ordering encodes a single bit, depending on the parity of the position. They also use

cyclic vertex shifts within triangles to gain another 2 bits per vertex. These steganographic techniques are lossless because reordering mesh elements does not distort the geometry.

Surprisingly, the authors of these papers do not seem to be aware of a well-known steganographic method, called *permutation steganography*, that gives optimal capacity. Permutation steganography hides information in the ordering of elements of a set, for example, in the order of cards in a deck, of people in a picture, of cars parked along the street, or anything else. This assumes that the elements can be rearranged into any order. A message is then encoded as the difference between the arrangement of elements with respect to a known reference ordering. Given n elements, permutation steganography can encode messages of up to $O(\log(n!)) = O(n \log n)$ bits, which is optimal and much better than the results reported for 3D mesh data to date.

The standard permutation steganography algorithm is described and implemented on a number of Web sites [Pst, Deo, Sgp, Gif], but has received little mention in scientific literature [Art01]. There is software that hides information in a GIF picture by reordering the color palette or in Web pages by reordering the arguments of HTML tags. The drawback of the standard algorithm is that it is based on integer arithmetic of very large numbers and its runtime complexity of $\Omega(n^2 \log^2 n \log\log n)$ can be prohibitive for long messages.

We present here a different permutation steganography algorithm whose runtime complexity is only $O(n)$ and whose implementation is surprisingly simple. This comes at the price of a slightly lower (and therefore less than optimal) capacity of $\log n - 1$ instead of $\log n$ bits per element. Our algorithm is completely universal and may be used instead of the standard permutation steganography algorithm whenever run-time efficiency and simplicity of implementation are important, especially when hiding large messages.

We apply our algorithm to polygonal meshes and permute mesh vertices and faces relative to a canonical ordering that we derive from a systematic traversal of the mesh connectivity. Obviously, this approach is lossless: The mesh with the hidden message is identical to the mesh without the message in all ways except the order of its elements. We significantly improve over previous results in lossless hiding of data in polygonal meshes. Our capacity is much higher and grows with the size of the mesh. Our encoding and decoding algorithms are fast and extremely simple to implement.

2. Permutation Steganography

Permutation steganography deals with encoding messages in a dataset by rearranging the order of elements in the dataset. Since there are $n!$ possible permutations of n elements, it should be possible to encode a message of $\log_2(n!)$ bits. The standard permutation steganography algorithm [Pst, Deo, Sgp, Gif] treats the sequence of bits representing a message as a very long unsigned integer. This number is represented in an *alternative basis*, which is defined by the number of elements only. The new permutation of the elements – encoding the message – is defined by the alternative representation.

2.1. The Alternative Basis

An integer M is represented in the alternative basis of n elements as the linear combination:

$$M = \sum_{k=1}^{n} c_k b_k, \tag{1}$$

where $\{b_k\}$ is the basis and $\{c_k\}$ is the alternative representation. The basis is defined as: $b_k = \prod_{i=0}^{k-1} d_i$, where $d_0 = 1$ and $d_{i\geq 1}$ is the number of distinct values that may appear in digit i. Thus $c_i \in \{0,..,d_i - 1\}$. Note that d_i may be different for different digits. In conventional number systems all digits have the same number of values, e.g. $d_{i\geq 1} = 10$ and $b_k = 10^{k-1}$ in the decimal system. It turns out that a variable number of values per digit in the basis $\{b_k\}$, as defined above, still leads to a unique representation, at least for any non-negative integer.

2.2. The Optimal Capacity Algorithm

The optimal capacity permutation steganography algorithm for n elements transforms a number M representing a message into its representation using an alternative basis. The number of possible values in digit i is defined to be $d_i = i$. Thus the basis elements are $b_k = (k-1)!$.

The encoding algorithm proceeds in the reverse order $k = n,..,1$, at each step computing the new representation c_k, starting with $M_n = M$:

$$c_k = M_k \text{ div } b_k \tag{2}$$

and the number representing the remainder of the message:

$$M_{k-1} = M_k \bmod b_k \tag{3}$$

The alternative representation $\{c_k\}$ is then converted to a permutation of n elements by making the element at position c_n the first element of the permutation, the element at position c_{n-1} among the remaining $n-1$ elements the second element of the permutation and so on (see details in Section 3.2).

For decoding, conversion of a permutation into a representation $\{c_k\}$ is done by reversing this logic using an appropriate data structure (see details in Section 3.3). The original message M is then recovered by applying Equation (1).

2.3. Analysis

The division and modulo operations (2) and (3) at step k involve two integers whose magnitude is $\geq (k-1)!$. Integer division of two m-bit integers requires $O(m^2)$ operations using the most naive method, and $O(m \log m \log\log m)$

operations using the most sophisticated (and complicated) method [PFTV92, SS71]. At step k, the length of the integers is $m = O(\log(k-1)!) = O(k \log k)$, incurring runtime of $O(k \log^2 k \log\log k)$. The total runtime complexity of the algorithm is thus

$$T(n) = \sum_{k=1}^{n} (O(k \log^2 k \log\log k)) = O(n^2 \log^2 n \log\log n)$$

The decoding algorithm has the same time complexity because the multiplication operation in (1) also runs in time $O(m \log m \log\log m)$ for m-bit integers.

The capacity of this algorithm is $\log_2(n!) = O(n \log n)$ bits, because an n-element basis can uniquely represent the $n!$ integers $\{0, 1, ..n! - 1\}$. This is also optimal, because n elements can generate no more than $n!$ permutations.

3. Our Algorithm

A typical polygon mesh dataset (also known as an *indexed face set*) consists of two kinds of *elements*: a list of 3D vectors representing the *geometry* of the vertices, and a list of polygonal faces representing the mesh *connectivity*. Each such polygonal face is represented as a list of (integer) indices into the vertex list. There is inherent redundancy in this representation: the same mesh is represented even if either or both lists are permuted (note: after permuting the vertex list the indices within the face lists must be updated). Permutation steganography takes advantage of this.

A permutation of the mesh elements is well defined only relative to some "canonical" reference ordering. To be useful, this canonical ordering should be easy to compute by some deterministic algorithm given only a start element. We capitalize on a number of canonical ordering methods that have emerged over recent years as "side-effect" of mesh compression algorithms. Examples are those generated by the TG algorithm [TG98] and the Edgebreaker [Ros99] algorithm. Other canonical orderings have also been used as so-called "rendering sequences" [BG02].

The encoding and the decoding algorithm consists of two stages. In the first stage a reference ordering of the mesh vertices and faces is computed. In the second stage the message is encoded as a permutation of the mesh vertices and faces relative to their reference order, or decoded by comparing the ordering present in the dataset to the reference one. We can hide one part of the message by rearranging the vertices and the other part by rearranging the faces. The vertex and face reorderings are independent of each other and use the same encoding and decoding procedure.

The essence of the encoding procedure is as follows: At step i we pick element e_i from position b amongst the $n - i$ remaining elements of the reference ordering and output it as the next element of our permutation. The position b is defined by the next several bits of the input message. The number of input bits read at step i is such that their numerical value is in the range $\{0, .., n - i - 1\}$. Then element e_i is eliminated from the reference ordering so that we are left with $n - i - 1$ available positions to chose from when picking the next element e_{i+1} for our permutation.

The output of the encoding procedure is a mesh with the same connectivity and geometry as the original, but with vertices and faces rearranged according to the hidden message.

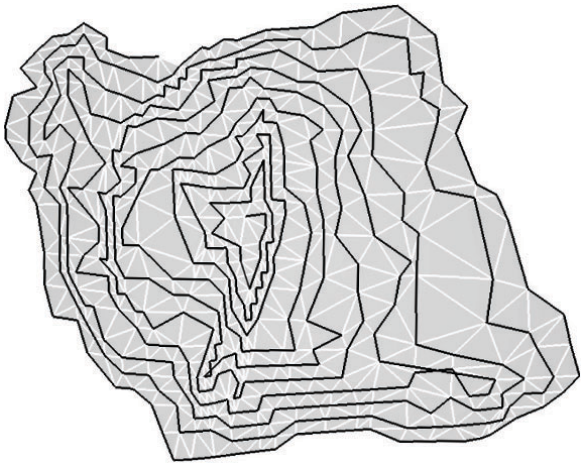

Figure 1: *An ordering of the vertices of an example mesh obtained from the Edgebreaker algorithm [Ros99].*

3.1. The Reference Ordering

To apply our data hiding method to 3D mesh datasets we need to obtain a "canonical" ordering of the mesh elements that can easily be computed by both the encoder and decoder. One possibility is to compute such an ordering based on the mesh geometry using some kind of spatial sorting [WC05, CW06], e.g. by sorting the vertices along one coodinate axis and breaking ties using the other two axii. This requires access to vertex geometry and is sensitive to geometric distortion. Instead we define a reference ordering based on the mesh connectivity alone. This makes our method immune to any kind of attack on the geometry.

We simultaneously compute a reference ordering of the mesh faces and vertices using the traversal performed by the Edgebreaker mesh compression algorithm [Ros99]. Once the initial vertex and edge are specified, the entire traversal is uniquely defined, see Figure 1. The only condition on the connectivity is that the mesh is manifold and orientable. Both the encoder and decoder compute this ordering without any information beyond the identity of the initial vertex and edge and the mesh connectivity. One possible way to ensure consistent selection of the initial vertex and edge during encoding and decoding is to exclude the first face from encoding. Then start with the first edge of the first face when constructing the reference ordering for decoding.

Note that any kind of well-defined deterministic mesh traversal could be used here, as long as it is independent of the element ordering in the input and is based only on the connectivity. The complexity of computing the traversal should not exceed $O(n)$. Possible alternatives are the traversal of the TG mesh compression algorithm [TG98], GPS traversal [GPS76] or universal rendering sequences [BG02]. The complexity of Edgebreaker and TG are $O(n)$.

```
Step 1:
Compute reference ordering of mesh vertices.
Place vertices into ref[] array

Step 2:
for  i = 0,..,n-1
   k ← ⌊log2(n-i)⌋
   b ← peek(k+1)      // peek at next k+1 bits
   if  2^k ≤ b  and  b < n-i
      advance(k+1)    // advance by k+1 bits
   else
      b ← peek(k)     // peek at next k bits
      advance(k)      // advance by k bits
   end
   perm[i] ← ref[b]          // vertex b to permutation
   ref[b] ← ref[n-i-1]       // replace with last vertex
end
```

Figure 2: *Pseudocode for encoding a message as a permutation of n vertices. The input is the message in form of a bitstream and the output is the permutation array perm[].*

3.2. Encoding

Encoding works exactly in the same way when hiding bits in the vertex or face orderings, thus we only describe how to reorder the vertices.

Let n be the number of vertices in the initial reference ordering and let i be the number of vertices that have already been reordered. At each step we pick a vertex at position b from the $n-i$ remaining vertices of the reference ordering and output it as the next vertex of the permutation. Since b is an integer in the range $\{0,..,n-i-1\}$, this encodes (at least) the next $k = \lfloor \log_2(n-i) \rfloor$ bits of the input message.

We peek at the next $k+1$ bits of the input message. If their value b is smaller than $n-i$ but also larger or equal to 2^k (i.e. if the first bit is set) we can encode $k+1$ bits by picking the vertex at position b. Otherwise we can only encode k bits and we use their value for picking the vertex. Figure 2 shows the pseudocode of the encoding algorithm.

After a vertex was picked for output, we remove it from the reference ordering and move the currently last vertex into its position. This means that the remaining vertices in the reference ordering are always stored consecutively in the array. Looking up a vertex at a given position can therefore be done in constant time. Hence, the entire encoding procedure has a time complexity of $O(n)$. This also means that the reference ordering is not fixed in its initial order but evolves over the course of the algorithm. This is fine as long as we assure that it evolves in the same manner during decoding.

```
Step 1:
Compute reference ordering of mesh vertices.
Place vertices into ref[] array
for  i = 0,..,n-1
   ref[i].ref ← i      // initial position in ref array

Step 2:
for  i = 0,..,n-1
   b ← perm[i].ref     // position in ref array
   k ← ⌊log2(n-i)⌋
   if  2^k ≤ b  then  k++
   output(b,k)         // output b using k bits
   ref[b] ← ref[n-1-i]// update reference ordering
   ref[b].ref ← b
end
```

Figure 3: *Pseudocode of decoding a message from a permutation of n vertices. The input is the permutation array perm[] and the output is the message in form of a bitstream.*

3.3. Decoding

The decoder enhances every vertex record with a `ref` field that stores the current position of the vertex in the evolving reference ordering. The decoder initializes the `ref` field of each vertex to the position it has in the initial reference ordering. The field is updated each time the last vertex replaces the eliminated vertex in the reference ordering.

Let n be the total number of vertices in the permutation and let i be the number of vertices that has already been processed. At each step we get the value b from the `ref` field of the next vertex in the permutation. This value corresponds to the position b that the vertex has in the reference ordering. It also represents the value of the next few bits in the output message. We still need to determine how many bits the value b represents. Let $k = \lfloor \log_2(n-i) \rfloor$. If $b < 2^k$ then b represents k bits of data. Otherwise $2^k \leq b < n-i$ must hold and b represents $k+1$ bits of data.

We then update the reference ordering. The currently last vertex of the reference ordering is copied to position b and the `ref` field of this vertex is updated to hold the correct new value b. Figure 3 shows the pseudocode of the decoding algorithm. Its time complexity is also $O(n)$.

4. Capacity

The capacity achieved by our method will not be optimal. By breaking up the very long input message into more manageable pieces which are encoded separately (as opposed to

encoding the entire input as one very long integer), we lose up to one bit per element. Luckily this is a very small loss compared to the optimal capacity of $\log n$ bits per element. More precisely, at step i of the encoding procedure at least $\lfloor \log_2(n-i+1) \rfloor$ bits are encoded. So for an ordering of n elements our algorithm achieves the following capacity:

$$C(n) \geq \Sigma_{i=1}^{n} \lfloor \log_2(n-i+1) \rfloor =$$

$$\Sigma_{i=1}^{n} \lfloor \log_2 i \rfloor \geq \Sigma_{i=1}^{n} (\log_2 i - 1) =$$

$$\log_2(n!) - n + 1 \qquad (4)$$

There are $n!$ possible permutations of n elements. Hence the maximal theoretical capacity of an n-element ordering is $\log_2 n!$ bits. From (4) we see that the difference between the capacity guaranteed by our algorithm and the theoretical optimum is less than one bit per element.

5. Experimental Results

We implemented the described algorithms in C++ and experimented with triangle meshes of different sizes on a laptop with a 2.1GHz Intel Centrino processor and 1GB of RAM. The results are summarized in Table 1. Computation times are nearly instant. For example, after loading the "isis" mesh from disk we need less than 0.3 seconds to hide a 1 MB message of random bits. Most of this time is spent on computing the reference ordering, which entails reconstructing connectivity and traversing it with Edgebreaker. Decoding is slower than encoding. This is due to the incoherent access to the `ref` field that hurts memory cache performance and the additional pass for initializing the `ref` field.

We provide three exact bit counts related to capacity. The minimum is the guaranteed capacity, computed from the lower bound as $\Sigma_{i=1}^{n} \lfloor \log_2 i \rfloor + \Sigma_{i=1}^{m} \lfloor \log_2 i \rfloor$. The maximum is the highest possible capacity for this model and is computed as $\lfloor \Sigma_{i=1}^{n} \log_2 i \rfloor + \lfloor \Sigma_{i=1}^{m} \log_2 i \rfloor$ (notice that here the floor operation is outside the sum). n is the number of vertices and m is the number of faces in the mesh. The third bit count is experimentally measured for a message with random bits.

An important observation is that the capacity per vertex in our method *increases* with the model size, as opposed to previous algorithms [WC05, CW06], for which the capacity in bits/vertex is independent of the mesh size. The smallest mesh used in our experiments achieved capacity of over 31 bit/vertex. This is more than 3.4 times greater than the fixed 9 bit/vertex of Cheng and Wang [CW06], which is the highest capacity algorithm so far. The largest mesh in our experiments was able to hide as much as 49.43 bit/vertex.

6. Summary and Discussion

We have presented a universal algorithm for permutation steganography, which may be applied to any data set which can tolerate an arbitrary permutation of its elements. The associated encoder and decoder run in $O(n)$ time on a dataset of n elements. This contrasts with standard permutation steganography techniques, which run in time $\Omega(n^2 \log^2 n \log\log n)$. The improved runtime complexity comes with a small penalty in capacity, which is one bit per element worse than the optimal $O(n \log n)$ bits.

We have shown that we can use this technique for distortion-free hiding of data within the connectivity component of a 3D mesh dataset. For this we derive the reference ordering of mesh elements—relative to which the permutation is computed—from a canonical traversal of the connectivity. The capacity we achieve is much higher than what has been reported in previous works [WC05, CW06].

Nowhere in the algorithm did we make any assumptions about degrees of the faces. In fact, the algorithm is not limited to triangular faces and can be used with any polygonal meshes. Care should be taken, however, that an appropriate traversal technique is chosen. For example, Edgebreaker and TG compression algorithms, whose traversals can be employed here, and were designed originally for triangle meshes, have versions that work on general polygonal meshes [KG01, KADS02, Ise02].

There is another independent medium for data hiding present in 3D mesh data. The order of the vertices *inside* faces can also be employed. Each face of d vertices has d equivalent possibilities for description, each a cyclic shift of the others. This can add another $\log_2 d$ bits per face. We did not describe this technique here because it has already been used by Cheng et al. [CW06] on triangular faces.

Finally, if an application requires higher capacity and some distortion can be tolerated, our algorithm can be combined with some other, geometry-based technique.

We should point out that while permuting vertices and faces does not affect the geometric quality of the mesh it can affect rendering performance—especially if the original mesh was ordered for cache efficient rendering [BG02]. However, in the past little attention has been given to the mesh element ordering at distribution time as evident in the highly incoherent mesh layouts of popular datasets [IL05].

Acknowledgements

Parts of this work were performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

References

[ADME02] ASPERT N., DRELIE E., MARET Y., EBRAHIMI T.: Steganography for three-dimensional polygonal meshes. In *SPIE 47th Annual Meeting* (2002), SPIE, pp. 705–708.

[Art01] ARTZ D.: Digital steganography: Hiding data

Mesh			Timings [msecs]					Capacity [bits]			
name	#verts	#faces	load	recon	trav	enc	dec	min	max	random	(b/v)
cow	2,904	5,804	15	0	0	0	0	89,331	98,037	91,141	31.38
fandisk	6,475	12,946	93	16	0	0	0	221,451	240,870	225,703	34.86
horse	48,485	96,966	171	31	16	15	16	2,082,158	2,227,607	2,112,433	43.57
armadillo	172,974	345,944	406	156	63	31	78	8,381,157	8,900,073	8,489,866	49.08
isis	187,644	375,284	468	203	47	32	93	9,158,667	9,721,593	9,274,892	49.43

Table 1: *Timing and capacity results for different meshes. Timings are reported in milliseconds for **load**ing the indexed mesh from disk, **recon**structing connectivity between triangles, **trav**ersing the mesh to create the reference orderings, **enc**oding a message that is a random stream of bits by permuting vertices and faces, and **dec**oding this message again. Capacity is reported in bits for the guaranteed **min**imum, the possible **max**imum, and the experimentally achieved message size for **random** input.*

within data. *IEEE Internet Computing 05*, 3 (2001), 75–80.

[BG02] BOGOMJAKOV A., GOTSMAN C.: Universal rendering sequences for transparent vertex caching of progressive meshes. *Computer Graphics Forum 21*, 2 (2002), 137–148.

[CDSM04] CAYRE F., DEVILLERS O., SCHMITT F., MAÎTRE H.: *Watermarking 3D triangle meshes for authentication and integrity*. Report 5223, INRIA, 2004.

[CW06] CHENG Y.-M., WANG C.-M.: A high-capacity steganographic approach for 3D polygonal meshes. *Visual Computer 22*, 9 (2006), 845–855.

[CWPG04] COTTING D., WEYRICH T., PAULY M., GROSS M. H.: Robust watermarking of point-sampled geometry. In *Proceedings of SMI* (2004), pp. 233–242.

[Deo] http://wandership.ca/projects/deogol/intro.html.

[Gif] http://www.darkside.com.au/gifshuffle/.

[GPS76] GIBBS N. E., POOLE W. G., STOCKMEYE P. K.: An algorithm for reducing the bandwidth and profile of a sparse matrix. *SIAM Journal on Numerical Analysis 13*, 2 (1976), 236–250.

[IL05] ISENBURG M., LINDSTROM P.: Streaming meshes. In *Visualization'05 Proceedings* (2005), pp. 231–238.

[Ise02] ISENBURG M.: Compressing polygon mesh connectivity with degree duality prediction. In *Graphics Interface'02 Proceedings* (2002), pp. 161–170.

[KADS02] KHODAKOVSKY A., ALLIEZ P., DESBRUN M., SCHROEDER P.: Near-optimal connectivity encoding of 2-manifold polygon meshes. *Graphical Models 64*, 3-4 (2002), 147–168.

[KDK98] KANAI S., DATE H., KISHINAMI T.: Digital watermarking for 3d polygons using multiresolution wavelet decomposition. In *Sixth International Workshop on Geometric Modeling: Fundamentals Applications Proceedings* (1998), pp. 296–307.

[KG01] KRONROD B., GOTSMAN C.: Efficient coding of non-triangular mesh connectivity. *Graphical Models 63* (2001), 263–275.

[ME04] MARET Y., EBRAHIMI T.: Data hiding on 3D polygonal meshes. *Proceeding of the 2004 workshop on multimedia and security* (2004), 68–74.

[OMA97] OBUCHI R., MASUDA H., AONO M.: Embedding data in 3D models. *Proc. of the 4th International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services* (1997), 1–10.

[OMT04] OHBUCHI R., MUKAIYAMA A., TAKAHASHI S.: Watermarking a 3D shape model defined as a point set. In *Proceedings of Cyberworlds* (2004), pp. 392–399.

[OTMM01] OHBUCHI R., TAKAHASHI S., MIYAZAWA T., MUKAIYAMA A.: Watermarking 3d polygonal meshes in the mesh spectral domain. In *Proceedings of Graphics Interface 2001* (2001), pp. 9–17.

[PFTV92] PRESS W. H., FLANNERY B. P., TEUKOLSKY S. A., VETTERLING W. T.: Numerical recipes in C: The art of scientific computing.

[PHF99] PRAUN E., HOPPE H., FINKELSTEIN A.: Robust mesh watermarking. In *Siggraph 1999, Computer Graphics Proceedings* (1999), pp. 49–56.

[Pst] http://www.wiredyne.com/software/pstego.html.

[Ros99] ROSSIGNAC J.: Edgebreaker: Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics 5*, 1 (1999), 47–61.

[Sgp] http://users.skynet.be/glu/sgpo.htm.

[SS71] SCHOENHAGE A., STRASSEN V.: Schnelle multiplikation großer zahlen. *Computing 7* (1971), 281–292.

[TG98] TOUMA C., GOTSMAN C.: Triangle mesh compression. In *Proceedings of Graphics Interface 1998* (1998), pp. 26–34.

[WC05] WANG C.-M., CHENG Y.-M.: An efficient information hiding algorithm for polygon models. *Computer Graphics Forum 24*, 3 (2005), 591–600.

[WW06] WANG C.-M., WANG P.-C.: Steganography on point-sampled geometry. *Computers & Graphics 30*, 2 (2006), 244–254.