Watermarking Three-Dimensional Polygonal Models

Ryutarou Ohbuchi, Hiroshi Masuda, Masaki Aono

ohbuchi@acm.org, masuda@trl.ibm.co.jp, aono@acm.org
IBM Tokyo Research Laboratory
1623-14 Shimo-tsuruma, Yamato-shi
Kanagawa, 242, Japan

Abstract

The advantages of digital media such as the Internet and CD-ROMs lie in the fact that their contents are easy to duplicate, edit, and distribute. These advantages, however, are double-edged swords, because they also facilitate unauthorized use of such contents. Data embedding, which places information into the contents themselves, is an approach to address this issue. Embedded information can be used, for example, for copyright protection, theft deterrence, and inventory.

This paper discusses our work on embedding data into three-dimensional (3D) polygonal models of geometry. Given objects consisting of points, lines, polygons, or curved surfaces, the data embedding algorithms described in this paper produce polygonal models with data embedded. Data are placed into 3D polygonal models by modifying either their vertex coordinates, their vertex topology (connectivity), or both.

A brief review of related work and a description of the requirements of data embedding is followed by a discussion of where, and by what fundamental methods, data can be embedded into 3D polygonal models. The paper then presents data-embedding algorithms, with examples, based on these fundamental methods.

Additional Keywords: three-dimensional geometrical modeling, three-dimensional graphics, data hiding, digital watermarking, steganography, copyright protection, digital fingerprinting.

1. Introduction

The advantages of digital media, such as the Internet and CD-ROMs lies in the fact that the duplication, distribution, and modification of contents are much easier than the older media, e.g., printed media. These advantages, however, are double-edged swords. Digital media made unauthorized duplication, distribution, and modification of their valuable contents easier.

One way of addressing this problem is to add *watermarks* to the objects in which contents are stored. Watermarks are structures containing information that are embedded in a data object (e.g., an image) in such a way that they do not interfere with its intended use (e.g., viewing). Watermarks can be used, for example, to deter theft, to notify users of how to contact the copyright owner for payment of licensing fees, to discourage unauthorized copying, or to take inventory. The technology associated with watermarks in this sense is called *steganography*, *data hiding*, *(digital) watermarking*, *data embedding*, or *fingerprinting* [Tanaka90, Walton95, Cox95, Braudway96, O'Ruanaidh96, Smith96, Zhao96, Hartung97]. The technology has been studied for still images, movie images, audio data, and texts in the past, but not for 3D model of geometry.

This paper presents fundamental techniques and algorithms for embedding data into 3D models of geometry. Realistic applications of data embedding would require more sophisticated features, for example, security provided by using encryption. However, a set of basic data embedding techniques can serve as a foundation for more specialized methods adapted for specific applications.

Figure 1. Three-dimensional models of dinosaurs in this scene are embedded with copyright notices and contact addresses, which can be extracted and displayed on demand.

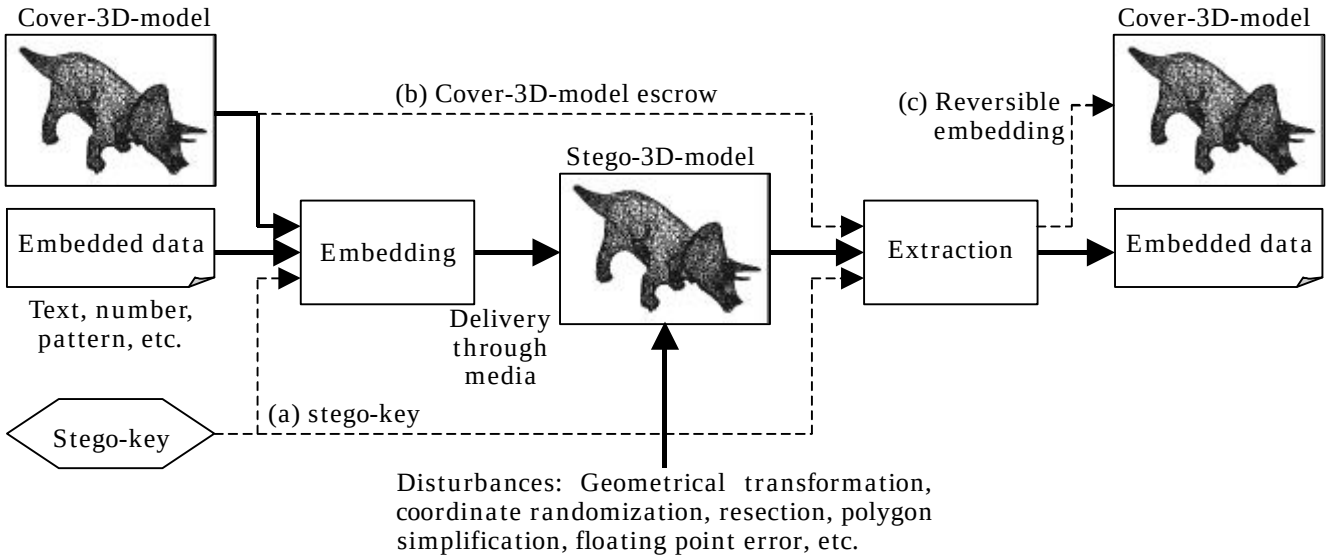


Figure 2. Data embedding into 3D polygonal models. Solid arrows indicate data flow in a basic embedding scheme. Dotted arrows indicate additional paths of information used in variations of the basic data embedding scheme: (a) stego-key, (b) cover-3D-model escrow, and (c) reversible embedding.

Figure 1 shows an example of embedding data into polygonal models of dinosaurs using an algorithm described in Section 3.2. Each dinosaur models are marked with a distinct message. The message embedded in the triceratops model, "Copyright (C) DINOSAURS INC. <CR> Model #triceratops003 <CR> Contact http://www.dinosaurs.org/.", can be extracted and displayed by clicking on the model while using a 3D model browser enhanced with the extracting capability. Such messages can be used, for example, to automatically connect the browser to a license server on the Internet to collect license fees.

Throughout the paper, the act of adding watermark is called *embedding (or watermarking)*, and retrieving the information encoded in the watermark for perusal is called *extraction*, as illustrated in Figure 2. The object in which the information is embedded is called *cover-<datatype>*, the object with watermark is called *stego-<datatype>*, and the information embedded is called *embedded-<datatype>*, following recommendations in [Pfitzmann96]. Suffix "<datatype>" varies with data types such as image, text, or 3D model. For example, an embedded-text is embedded in a cover-3D-model to produce a stego-3D-model with embedded-text.

Figure 2 illustrates the flow of data in a most basic of the data embedding scenarios, the subject of this paper, with solid arrows. In addition, three possible variations of the basic embedding schemes are illustrated with dotted arrows. In the first example, a stego-key is used to encrypt embedded data so that the embedded data are secure from the third party. The encryption scheme used can be either public-key or private key [Shneier96], although the illustration indicates a private-key scheme. In the second example, escrowing the original cover-3D-model makes extraction much easier and robust. However, the need for escrow makes the scheme unsuitable for many application scenarios. In the third example, the original cover-3D-model is recoverable by extraction, by using a reversible data embedding algorithm.

The remaining part of this paper is organized as follows. In next section, we present a brief review of related work and a discussion on requirements for data embedding into 3D models. We then details fundamental methods for data embedding into 3D models, namely, basic units of embedding and method to arrange a collection of one of these units to embed a meaningful amount of data. These fundamental methods are combined to create several embedding algorithms described in Section 3. We conclude in with a summary and some remarks on possible directions for future work.

2. Fundamentals of watermarking 3D models

2.1. Related work

In case of 3D model of geometry, the comment and annotation capabilities of scene description formats, such as the Virtual Reality Modeling Language (VRML) [ISO96], have been the primary means of adding information. However, these comments and annotations can be easily removed, either intentionally or unintentionally. For example, programs for converting between 3D model formats often remove comments and annotations. Consequently, comments and annotations cannot meet most of the requirements of watermarks.

2.2. Embedding requirements

While each application of data embedding into 3D models has its own set of requirements, the following three requirements are common to a majority of application scenarios.

Unobtrusive: The embedding must not interfere with the intended use of a model, such as viewing. One published example of image watermarking [Braudway96] used the visibility of the watermarks to its advantage, but in most applications, the watermarks must be unnoticeable in terms of the model's intended use.

Robust: Robustness is crucial to the success of data embedding. Making the watermark indestructible is not a trivial problem. Note that this is different from making message in the watermark unreadable.

With complete knowledge of how watermarks are embedded, any watermarks can theoretically be removed. With partial knowledge (e.g., the knowledge of the basic algorithm), the removal must be difficult enough so that it either interferes with the intended use of the models, or the cost of removal is greater than the value of the model.

Assuming the kind of use a VRML model has to expect, watermarks in a 3D model have to expect the following kinds of alterations during day-to-day use. Data format conversion is a common practice, which scrambles orders of points, polygons and other geometrical primitives. Data format conversion often introduces floating-point-number representation errors. Models are geometrically transformed to construct a scene. While the geometrical transformations are often limited to rotation, uniform scaling, and translation, more general transformations, e.g., affine transformations are common. Local deformations are occasionally applied to reshape a part of a model. Topological alterations, such as resection of a desired part of a model and polygon simplification, may also be performed.

Watermarks in a model may also be attacked with an intent to destroy or alter them. Possible means of intentional attack include addition of random (or systematic) values to vertex coordinates and polygonal simplification. It is not feasible to list every possible means of attacks.

If the degree of modification is limited so that the utility of the model is not compromised, watermarks in 3D models should *ideally* withstand all of these and other possible alterations, regardless of whether they are intentional or otherwise.

Space efficient: A data-embedding method should be able to embed a non-trivial amount of information into models.

In general, above three requirements are at odds. For example, if one needs more robust embedding, the amount of data that can be embedded is reduced. The best trade-off depends on each application.

2.3. Embedding targets

A 3D model may contain diverse range of data objects. For example, a VRML 2.0 file includes geometry of objects defined by polygons, lines, or predefined shapes (e.g., cylinders, spheres, or cones). These objects have attributes, such as shininess, per-surface or per-vertex colors, per-surface or per-vertex normal vectors, per-vertex texture coordinates, texture images, and others. The file may also contain Universal Resource Locator links, pointers to sound data files, behavioral scripts written in a programming language, and others. In case of non-VRML models, geometry of 3D objects may be represented by solids bounded by curved surfaces (e.g., Bezier patches), by voxel enumeration, and many others means.

We argue that geometry is the best candidate for data embedding among the data objects types that could exist in 3D scene descriptions, since it is by definition the least likely to be removed.

Among the many possible representations of 3D geometry, we chose, for the study reported in this paper, *polygonal models* as the target (output) of embedding (see Figure 2). A "polygonal model" in this paper may include one or more of the following geometrical primitives: points, lines, polygons, connected polygons, polyhedrons, and connected polyhedrons. While some data embedding algorithms require topology (connectivity) among points, topology can be added, for example, by Delaunay triangulation [O'rourke94].

Inputs to the embedding algorithm may either be polygonal models or *curved surface* models. An embedding may be performed either during or after a tessellation of the curved surfaces. For example, the algorithm that will be described in Section 3.5 accepts curved surfaces as input and embed patterns as it tessellates the surfaces. Embedding watermarks during a tessellation can be advantageous to data embedding. This is because the embedding algorithm could exploit a large degree of freedom it has in choosing the number, position, and topology of vertices produced by the tessellation.

Other components of 3D scene descriptions can also be used for data embedding. Images for texture mapping and sounds are obvious targets of embedding. Per-vertex normal vectors, per-vertex texture coordinates, per-vertex colors, or even face colors can also become targets of data embedding. These non-geometrical components, however, are less crucial to 3D scenes and have higher chances of alteration or removal than geometry.

2.4. Embedding primitives

There are two attributes in a polygonal model of geometry that can be modified in order to add watermarks. One is the *geometry* of the geometrical primitives (e.g., points or triangles) and the other is the *topology* among these primitives. Units of alteration, either geometrical or topological, are called *embedding primitives* in this paper.

2.4.1. Geometrical embedding primitives

Geometrical values - specifically, the coordinates of points and vertices - can be modified to embed data. Notice, however, that information encoded directly in coordinate values is vulnerable to almost any geometrical transformations. It is thus advantageous to employ geometrical embedding primitives that are *invariant* to certain classes of geometrical transformations. The following lists examples of embedding primitives derived

from vertex coordinates that are invariant to increasingly larger class of geometrical transformations.

1. Altered by all the transformations listed below
   a. Coordinates of a point.
2. Invariant to translation and rotation
   a. Length of a line.
   b. Area of a polygon.
   c. Volume of a polyhedron.
3. Invariant to rotation, uniform-scaling, and translation
   a. Two quantities that define a set of similar triangles (e.g., two angles).
   b. Ratio of the areas of two polygons.
4. Invariant to affine transformation
   a. Ratio of the lengths of two segments of a straight line.
   b. Ratio of the volumes of two polyhedrons.
5. Invariant to projection transformation.
   a. Cross-ratio of four points on a straight line [Farin96].

Upon embedding, the quantity of the primitive is modified, typically by very small amount, so that subsequent displacements of vertices do not affect the intended uses of the model.

The algorithms that will be described in Section 3.1 and Section 3.2 uses the primitive 3a and 4b, respectively.

2.4.2. Topological embedding primitives

Watermarks can be embedded by changing the topology of a model. The change may also involve change in geometry as a side effect (e.g., inserting or displacing vertices), but information is embedded mainly in the topology.

An example of topological embedding primitive is the connectivity of triangles in a triangle strip. A geometry compression algorithm described in [Taubin96] encoded the topology of a triangle strip in a bit string. (To be more precise, the algorithm described in [Taubin96] is more general, for it encoded a topology of a triangle strip with branches, i.e., *tree* of triangles.) This technique can be used in reverse, by letting a bit string control growth of a triangle strip. This approach is used in the algorithm that will be described in Section 3.3.

Another example of topological embedding primitives simply cuts out holes in an input mesh to encode a pair of symbols. This approach is used in the algorithm that will be described in Section 3.4. Yet another example of topological embedding primitives encodes a pair of symbols by using two different mesh sizes, □ and ⊞.

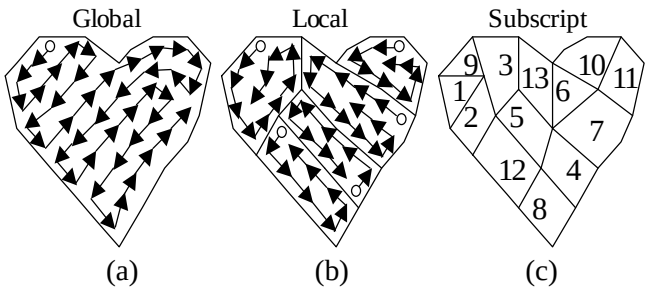


Figure 3. Illustrations of examples of global, local, and subscript arrangements defined on a geometrical object.

This method is used in the algorithm that will be described in Section 3.5.

2.5. Embedding primitive arrangements

For a practical data embedding, multiple embedding primitives must be arranged so that a collection of embedding primitives functions as a watermark to store a substantial amount of information.

Data objects such as image and audio data already have regular implicit ordering of embedding primitives. For example, an image has rectangular 2D array of pixels. In case of 3D geometrical models, arrangement of embedding primitives is somewhat more involved.

An example of arrangements of embedding primitives for 3D geometrical models is a 1D arrangement generated by sorting triangles according to their areas. Another example is a 2D arrangement of triangles based on the connectivity of triangles in an irregularly-tessellated triangular mesh.

Arrangements of embedding primitives can be established for 3D polygonal models by the following two methods.

a. Topological arrangement employs topological adjacency, such as the adjacency of vertices, to arrange embedding primitives. Topological arrangement is applicable to both topological and geometrical embedding-primitives. It can survive a geometrical transformation, but is not resistant to a topological modification.

b. Quantitative arrangement employs inequality relations among the quantities, such as volumes of polyhedrons, associated with embedding primitives to sort those primitives.

In both arrangement methods, it is often necessary to find an *initial condition* - for example, the first primitive of a one-dimensional arrangement - in order to initiate an

arrangement. Obviously, both arrangement and initial condition must be robust against expected disturbances, such as geometrical transformations, or the watermarks will be lost.

In this paper, arrangements of embedding primitives are classified by their locality into *global*, *local*, and *subscript arrangements.* Figure 3 shows illustrations of examples of these three types of arrangements based on topological adjacency.

a. Global arrangement arranges a set of all the embedding primitives in an embedding target.

b. Local arrangement arranges each one of multiple disjoint subsets of every embedding primitive in an embedding target.

c. Subscript arrangement is similar to local arrangement, but with a very small (e.g., a few primitives) subset, which is called a *Macro-Embedding-Primitive (MEP)*. Each MEP is associated with a special kind of data, a unique subscript. Subscripts map a set of embedding primitives into a sequence.

A global arrangement tends to have higher information density than the other two methods. However, by repeatedly embedding a message, local and subscript arrangements can be more robust against partial disruption of arrangements due, for example, to resection of model .

Arrangements of embedding primitives are used to embed data in two alternative ways, by means of what we call *symbol-sequence-embedding* and *pattern-embedding.*

a. Symbol-sequence-embedding method embeds an ordered sequence of symbols, such as a character string. Symbol-sequence-embedding typically employs a 1D arrangement of embedding primitives.

b. Pattern-embedding method embeds patterns that are visually recognizable if presented to human beings. For example, shapes of letters can be cut into a triangular mesh as a visible watermark which are visible if displayed by using a wire-frame rendering. Not all watermarks produced by pattern-embedding are visible, however.

The mapping from embedded data (either a symbol sequence or a pattern) and an arrangement of embedding primitives does not have to be straightforward. Scrambling the mapping, for example by using a pseudo random number sequence generated from a stego-key, could increase security of the embedded data. In this paper, however, we will not discuss this and other methods of scrambling watermark any further.

3. Embedding algorithms

This section describes, along with execution examples, algorithms that are created by combining fundamental methods discussed in the previous section. In developing the following algorithms, we assumed viewing of models by 3D model browsers (e.g., a VRML browser) as the intended use of models.

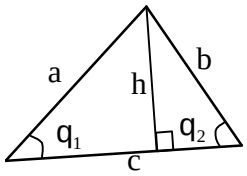


Figure 4. Examples of dimension-less quantities that defines a set of similar triangles.

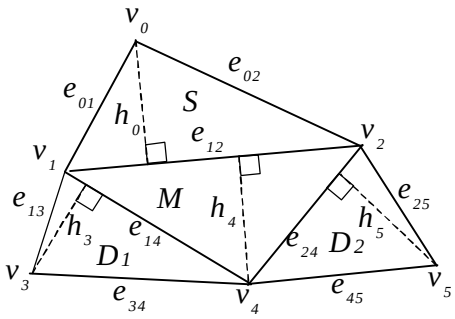


Figure 5. A macro-embedding-primitive. In the figure, v_I are vertices, e_{ij} are lengths of the edges, and h_I are heights of the triangles.

All the algorithms are implemented by using a kernel for a non-manifold modeler [Masuda96]. The system employs *radial edge* structure [Weiler86] to represent the topological relationship among vertices, edges, faces, and regions.

3.1. Triangle similarity quadruple embedding

A pair of dimensionless quantities, for example, {b/a, h/c} or $\{q_1,q_2\}$ in Figure 4, defines a set of similar triangles. The algorithm described in this section, which is called Triangle Similarity Quadruple (TSQ) algorithm, uses such dimensionless quantity pair as the geometrical embedding primitive to watermark triangular meshes. In order to realize subscript ordering, the algorithm uses a quadruple of adjacent triangles that share edges in the configuration depicted in Figure 5 as a Macro-Embedding-Primitive (MEP). Each MEP stores a quadruple of values *{Marker, Subscript, Data1, Data2}*. A marker is a special value (in this case a pair of values) that identifies MEPs. In Figure 5, the triangle marked *M* stores a marker, *S* stores a subscript, and *D1* and *D2* stores data values. While each MEP is formed by topology, a set of MEPs are arranged by quantity of subscript.

The TSQ extraction algorithm does not require escrowed original cover-3D-model for extraction. However, it does require a pair of values that identifies marker triangles. Watermarks produced by the TSQ algorithm withstand translation, rotation, and uniform-scaling transformations of the stego-3D-models. The watermarks are resistant to resection and local deformation since subscript arrangement and repeated embedding are

employed. The watermarks are destroyed, among other disturbances, by a randomization of coordinates, by a more general class of geometrical transformation, or by an extensive topological alteration such as re-meshing.

The TSQ algorithm embeds a message according to the following steps.

(1) Traverse the input triangular mesh to find a set of four triangles to be used as a MEP. In doing so, avoid vertices that have already been used for the watermark, or triangles that are unfit for stable embedding, e.g., triangles whose dimension-less quantities are too small.
(2) Embed the marker value in the center triangle of the MEP by changing its dimensionless quantity pair $\{e_{14}/e_{24}, h_4/e_{12}\}$, hence coordinates of its vertices v_1, v_2, and v_4, by small amounts (See Figure 5).
(3) Embed a subscript and two data symbols in the remaining three triangles of the MEP by displacing vertices v_0, v_3, and v_5 that are not shared with marker triangle in the center. Subscript is embedded in the pair $\{e_{02}/e_{01}, h_0/e_{12}\}$, and two data symbols are embedded in the pairs $\{e_{13}/e_{34}, h_3/e_{14}\}$ and $\{e_{45}/e_{25}, h_5/e_{24}\}$. For each the three triangles, the algorithm first modifies the ratio h_i/e_{ij} by changing h_i only. Then the algorithm modifies the ratio e_{ij}/e_{kl} while keeping the height h_i constant.
(4) Repeat (1) to (3) above until all the data symbols of the message are embedded.

In order to embed multiple copies of the message, steps (1) to (4) are repeated many times. Figure 7 shows triangles that formed MEPs in darker gray. Due to the mutual exclusion rule described in the step (1) above, MEPs do not share vertices.

In the steps (2) and (3) above, the magnitude of modification of the quantities must be larger than the expected noise. At the same time, it must be small enough so that the watermarks are not noticeable by human beings when displayed by using a model browser. These minimum and maximum magnitudes of modifications are chosen as a result of a trade-off between robustness, space efficiency and noticeability.

(a) (b) (c) (d)

Figure 6. A model of a Beethoven's bust (4889 triangles) resected repeatedly by arbitrary planes.

	Number of triangles	Data
(a)	4889	6 copies, 132 bytes each
(b)	2443	132/132 bytes
(c)	1192	102/132 bytes
(d)	399	85/132 bytes

Table 1. Data loss due to resection in the example shown in Figure 6.

Model	Number of triangles	Data capacity per model [bytes]	Embedding execution timing [sec]
Cow	5804	1062	19.8
Triceratops	5604	966	9.9
Beethoven	4889	873	9.7
IBM mesh	2996	486	9.8
Face	1406	210	2.3
Stegosaur	1023	180	2.5
Sphere	959	132	1.4

Table 2. Model size, embedding data capacity, and execution timings for embedding operation for the Triangular Similarity Quadruple (TSQ) embedding algorithm.

Given a watermarked mesh and two numbers that identify marker triangles, extraction proceeds according to the following steps.

(1) Traverse a given triangular mesh and find a triangle

with the marker, thereby locating a MEP.

(2) Extract a subscript and two data symbols from the triangles in the MEP.

(3) Repeat (1) to (2) above for all the marker triangles on a given triangular mesh.

(4) Sort the extracted symbols according to their subscripts.

The TSQ algorithm performs a simple error correction by majority voting if multiple copies of a message are embedded.

Figure 6a shows a model of Beethoven's bust (4889 triangles, 2655 vertices) in which six identical copies of a message, each message consisting of 132 bytes, have been embedded by using the TSQ algorithm. The message was gradually lost when the model was resected by arbitrary planes (Figure 6b-c). As shown in Table 1, cutting

Figure 7. Macro embedding primitives, each of which consists of four adjacent triangles, are shown in dark gray.

the model in half left the entire message intact, and quartering the model left 102 out of 132 bytes intact. Since a subscript arrangement was used, intact characters still tended to be in the correct positions within the message string.

Table 2 shows, for various models, their data capacities and execution timings for embedding in the case of the TSQ algorithm. Timings for extraction were not listed since they are about the same as those of embedding. The timings were measured by using IBM AIX™ 4.1.4 operating system and a xlC c++ compiler on a 100 MHz PowerPC 604 processor.

Space efficiency seen in these examples is adequate for many practical applications. It should be noted, however, that these examples pushed space efficiency by somewhat sacrificing robustness. Increasing robustness, for example by increasing number of repetition of a message and by using an error-correcting code would reduce effective data capacity.

Execution timings are roughly proportional to the number of triangles in the models. The embedding algorithm was prototyped on a full-fledged non-manifold modeler kernel, which has many more features than necessary for the embedding algorithm Timings will improve significantly if the code is designed for the embedding algorithm.

3.2. Tetrahedral volume ratio embedding

A ratio of volumes of a pair of tetrahedrons is the embedding primitive for the Tetrahedral Volume Ratio (TVR) embedding algorithm described in this section. The algorithm is designed to accept triangular meshes as its input. It arranges the embedding primitives topologically into either global or local one-dimensional arrangement for symbol sequence embedding.

The TVR algorithm does not require cover-3D-model for extraction. The watermarks produced by the TVR algorithm survive affine transformation. The watermarks are destroyed, among other disturbances, by topological modifications such as re-meshing, randomization of vertex coordinates, and geometrical transformations more general than affine transformation (e.g., a projection transformation). A variation of the TVR algorithm discussed at the end of this section is resistant to resection and local deformation through the use of local arrangement and repeated embedding.

The TVR algorithm embeds data in accordance with the following steps. The crux of the algorithm is establishing global one-dimensional ordering of embedding primitives. Details of step (1) below will be explained later.

(1) Find a spanning tree of vertices *Vt*, called *vertex tree*, on the input triangular mesh *M*, given an initial condition *Ivt* for *Vt*. Convert *Vt* into a sequence of triangles *Tris*, called a *triangle sequence.*

(2) Convert *Tris* into a sequence of tetrahedrons *Tets*, called a *tetrahedron sequence.* To do this, compute a common apex as the centroid of the coordinates of a few triangles selected from the triangle sequence (e.g., first three). The selected triangles are removed from the triangle sequence so that their coordinates are not modified by embedding of symbols.

(3) Convert *Tets* into a sequence of ratios of volumes *Vrs*. To do this, a volume of a tetrahedron (e.g., the first one) in *Tets* is selected as a common denominator of all the ratios, and volumes of the remaining tetrahedrons are used for numerators.

(4) Embed a symbol into each ratio by displacing vertices of numerator-tetrahedrons. The vertex displacements for the current symbol must not interfere with modifications of the previously embedded symbols. (In Figure 8, triangles that are used for embedding, which are colored dark gray, do not share edges because of this constraint.)

We now explain details of the first step, starting with the method to create a triangle sequence, and later come back to explain how to find an initial condition *Ivt.*

Generating vertex tree *Vt* from an input triangular mesh requires that the input mesh is an orientable manifold. To generate *Vt*, traverse vertices from a given initial condition *Ivt*, that is, *{initial vertex, initial traverse direction}* pair, starting with the *Vt* initialized to empty. At each vertex, by scanning the edges in counter-clockwise order, find an edge that is not a member of *Vt* and does not loop back to any of the vertices covered by *Vt*. If such an edge is found, add it to *Vt*.

In the example shown in Figure 9, the vertex tree has a root (and a branching point) at the vertex numbered 1. After passing through vertices 1 through to 10, the traverse backtracks to vertex 1 and adds two vertices, 11 and 12.

The vertex tree *Vt* is converted into the triangle sequence *Tris* as a set of edges *Tbe*, called a Triangle Bounding Edge (TBE) set is constructed. The *Tbe* is initialized to a set of edges that connect vertices in the *Vt*. To add an edge to the *Tbe*, vertices are traversed according to the *Vt*, starting from the root. At each vertex, all the edges adjacent to the vertex are scanned clockwise, and the scanned edge is added to the *Tbe* if it is not a member of the *Tbe*. A new triangle is added to the *Tris*, which started as an empty sequence, if all three edges of the triangle are in the *Tbe* for the first time, and the triangle is not already in the *Tris*. In the example shown in Figure 9, edges (except the initial entries of the *Tbe*) are marked by alphabets in the order of addition to the *Tbe* using alphabetical ordering. In the figure, members of the *Tris* are marked by the numbers in the circles according to the sequence each triangle is added to the *Tris*.

As the initial condition, the TVR algorithm selects an

Figure 8 Triangles used for embedding by the TVR algorithm are shown in dark gray.

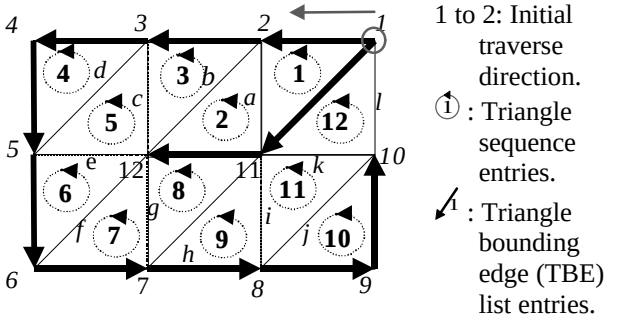


Figure 9. An example showing a vertex tree, a triangle bounding edge list, and a triangle sequence.

initial *edge*, instead of an {initial vertex, initial traverse direction} pair mentioned before. To select the initial edge, the algorithm computes, for every edge in the model, the volume of the tetrahedron subtended by the two triangles that are adjacent to the edge (Figure 10). The algorithm selects, as the initial edge, the edge for which tetrahedron's volume is the largest. This method works since affine transformation preserves inequality among volumes of tetrahedrons. (Note that these tetrahedrons are different set of tetrahedrons from the ones used to embed symbols.)

The TVR extraction algorithm employs the trial-and-error method to find a correct initial edge. The algorithm tries multiple candidate edges until it extracts a correct predetermined lead-in symbol sequence. This is because the edge found to have the largest volume may be incorrect due to noise and other reasons.

Using an edge as initial condition leaves two equally possible alternatives in starting a traversal to construct a vertex tree. This ambiguity is resolved, again, by using the trial-and-error method. The TVR extraction algorithm

extracts two sequences of symbols by using both alternatives. It then choose the direction that yielded correct predetermined lead-in symbol sequence.

Table 3 shows, for various models, data capacity per model and execution time for embedding operation. Conditions for timing measurements are the same as for Table 2. As with the TSQ algorithm, the amount of data that can be embedded by using the TVR algorithm appears to be adequate for many applications. Execution timings in the same table show similar tendency as the TSQ algorithm; they are roughly proportional to the size of the model. These timings also leave room for a significant improvement by designing an optimized code.

As mentioned before, the watermarks by TVR algorithm can be made resistant to resection and local deformation by using a local or subscript arrangement method combined with repeated embedding. The TVR algorithm strengthened by a local arrangement method, called TVR Cluster (TVRC) embedding algorithm, was used in the examples shown in Figure 11. In order to create sub-domains for local arrangement, this algorithm simply split the model into subsets (i.e., multiple disconnected meshes). Boundaries of subsets have duplicate vertices and edges so that the crack will not become visible. In this example, the message embedded in a model of a cow survived affine transformations. It also survived, to some extent, resections of a part of the model.

3.3. Triangle strip peeling symbol-sequence-embedding

The embedding algorithm presented in this section, called *Triangle Strip Peeling Symbol sequence (TSPS)* embedding algorithm, peels off triangle strips from a given triangle mesh in order to embed symbol sequences. The embedding primitive is an adjacency of a pair of triangles in a triangle strip, each of which encodes a bit of information. The adjacency also induces 1D arrangement of embedding primitives in the triangle strip. Since the algorithm employs a topological embedding primitive and topological arrangement, the watermarks produced by the algorithm are robust against practically every geometrical transformation. By repeating the embedding, the watermark as can be made resistant to resection. The algorithm does not require cover-3D-model escrow for extraction.

A disadvantage of this algorithm is its relatively low space efficiency. The embedding can be destroyed, for example, by polygon simplification algorithms that employ edge swapping operations.

Inputs to this embedding algorithm are an orientable triangular mesh and a sequence of binary digits. The basic TSPS embedding algorithm embeds data according to the following steps. (See Figure 12.)

(1) Starting from an edge *e* selected from the input mesh *M*, grow a triangular strip *S* on *M* by using the message bit-string to determine the connectivity of triangles on the strip.

Observe that a triangle at the end of (current) strip has two “free” edge, i.e., edges that are not adjacent to

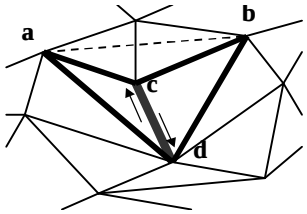
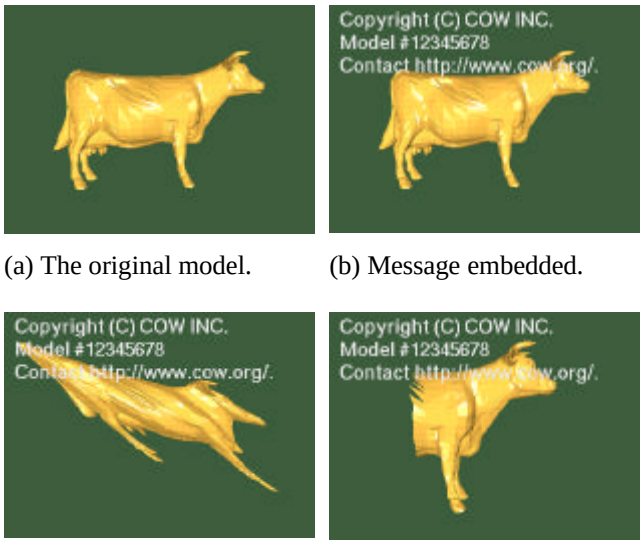


Figure 10. The volume of the tetrahedron a-b-c-d, subtended by two triangles a-d-c and b-c-d that are adjacent to the edge c-d, is computed. The arrows show two possible initial traverse orders from the edge c-d.

Model	Number of triangles	Data capacity per model [byte]	Embedding execution timing [sec]
Cow	5804	1027	20.2
Triceratops	5604	650	7.2
Beethoven	4889	324	9.6
IBM mesh	2996	652	9.9
Face	1406	116	2.2
Stegosaur	1023	225	2.4
Sphere	959	216	1.5

Table 3. Model size, data capacity, and execution timings for embedding operation for the Tetrahedral Volume Ratio (TVR) embedding algorithm.



(a) The original model. (b) Message embedded.

(d) Affine-transformed. (c) Cut in half.

Figure 11. (a) Three-dimensional model of a cow (5804 triangles). (b) A message is embedded by using the TVR algorithm enhanced with local arrangement and repeated embedding.. The message survives (c) resection or (d) affine transformation.

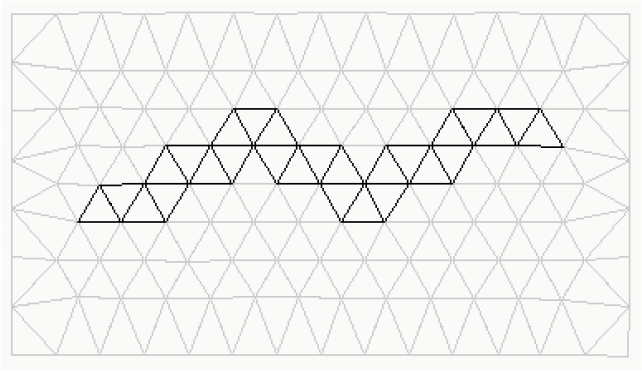


Figure 14. A triangle strip consisting of 27 triangles was cut out a triangle mesh (214 triangles). The triangle strip, displayed in darker gray, encodes 13 data bits interleaved with 13 steering bits.

triangles of the triangle strip. Since *M* is orientable, these two edges can be ordered on the triangle by traversing the edges in counterclockwise (or clockwise) order. Label the first free edge '0' and the second (i.e., last) free edge '1'. Then, depending on the bit to be embedded, choose one of the free edges to add the next triangle of the triangle strip. (See Figure 13.)

(2) "Peel off" the triangle strip *S* from *M* by splitting all the edges on the boundary of *S* except the initial edge *e*. (The strip *S* is connected to the rest of the mesh only by the edge *e*.) Since the peeled strip caps the hole completely, presence of the watermark is visually unnoticeable.

The mesh with watermark is called M^+, and the mesh M^+ minus the triangle strip *S* is called a *stencil mesh R*. The edge *e* serves as the initial condition for finding the triangle strip. Ordering of primitives is induced naturally by the connectivity of embedding primitives in the triangle strip. Termination condition of the arrangement is the open end of the strip.

Figure 13 shows an example of a triangle strip that starts at edge *e* and embeds a message bit string "10101101011" in a sequence of 12 triangles. Each bit of the bit string steers the direction of the growth of the triangle strip.

Note that such steering may produce strips whose shape may not fit in a given mesh, depending on the lengths and arrangement of 0s and 1s of message bit strings. For example, if a sequence of 1 continue in a bit string, the strip will keep steering to the left. This strip will either hit a boundary of the mesh *M* or circle back to itself, most likely before it is long enough to have embedded all the message bits. To avoid such problems, shapes of triangles strips must be manipulated, and locations and orientations of the strips in the mesh *M* must be chosen carefully.

The shape of a triangle stripe can be manipulated by introducing *steering symbols*. A steering symbol is a bit that do not carry information but simply steer direction of

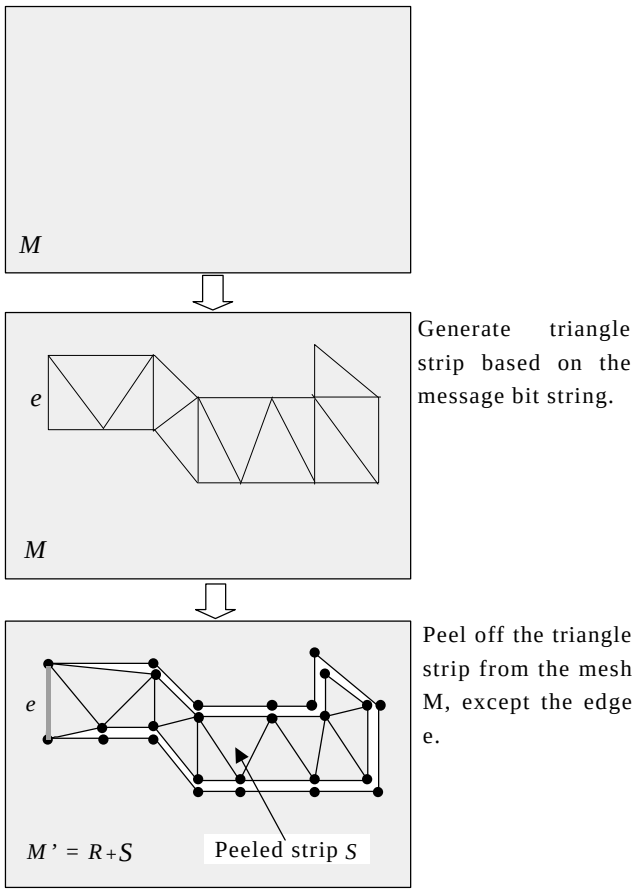


Figure 12. Triangle strip peeling symbol sequence embedding algorithm. Given a mesh *M*, a bit string is embedded into connectivity of triangles in a triangle strip *S*. *S* is then peeled off from the mesh *M*, except for edge *e*. (Cracks around the strip in the bottom figure are for illustration purpose only.)

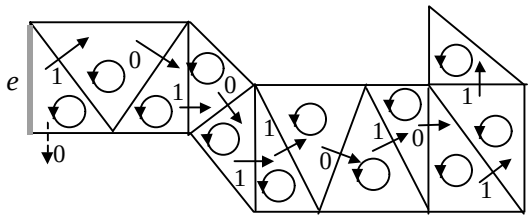


Figure 13. Connectivity of 12 triangles in a triangle strip encodes the bit string "10101101011" (11 bits).

growth of a triangle strip. Steering symbols are interleaved with data symbols (i.e., symbols that encodes embedded data) in order to actively steer the triangle strip into a desired shape. A drawback of using steering symbols is that the triangle strip becomes less space efficient. To pick locations and orientations of triangle strips on a mesh, our algorithm currently employs a simple-minded trial-and-error approach.

Extraction of message can be carried out according to the following steps.

(1) Traverse the given mesh M^+ and find an edge with topological features that starts a triangle strip of known length that is attached to the stencil mesh by an edge.

(2) Traverse the triangle strip to the open end as embedded bits are extracted.

Although rare, it is conceivable for the meshes *M* to have topological structures identical to that of peeled off triangle strip. Data extracted from such triangle strips are meaningless. These erroneous extraction results can be rejected, for example, by using a "signature" bit sequence.

Figure 14 shows a simple example of TSPS embedding, in which a triangle strip of length 27 is peeled off of a mesh that consisted of 214 triangles. The triangle strip encodes 13 data bits and 13 steering bits. Selection of steering bits in this case was done manually.

3.4. Polygon stencil pattern-embedding

Given a mesh *M*, a pattern can be embedded by simply cutting out a polygonal strip *S* in a desired pattern, as illustrated in Figure 15. *S* is attached to the embedded-mesh mesh by an edge, for example; It is easier to find and remove *S* if it is totally disconnected from the rest of the mesh. Since the strip completely caps the hole, watermarks are visually unnoticeable.

Watermarks produced by this algorithm, called Polygon Stencil Pattern (PSP) embedding, are robust against most of the polygonal simplification algorithms, since vertices on the boundary of polygonal strips and stencil meshes are preserved by these algorithms. (Unlike the TSP algorithm, connectivity of vertices on the strip does not matter.) If vertices on the boundary are removed or displaced forcefully, cracks will most likely appear in the model, diminishing the value of the model.

Figure 16 shows a stenciled polygonal mesh, a cut out triangle strip, and effects of polygon simplification algorithm on them. The vertices on the boundary of the stencil mesh are preserved despite a polygon simplification, which reduced the number of triangles from 1815 down to 459. Number and coordinates of vertices on the boundary of the triangle strip did not change after simplification. Topology of the edges did change after simplification, however, due to edge swapping employed by one of the simplification algorithms we have tried. (One of the polygonal "simplification" algorithm that employed edge-swapping actually increased the number of triangles from 207 triangles to 213 triangles while it kept the number of vertices unchanged.)

3.5. Mesh density pattern embedding

Another simple pattern embedding algorithm, named Mesh Density Pattern (MDP) embedding, generates polygonal mesh models given curved surface models as inputs. As the algorithm tessellates given curved surfaces, it embeds visible pattern by modulating the sizes of triangles

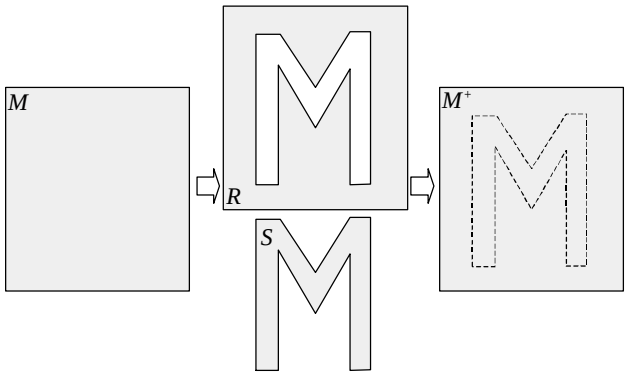
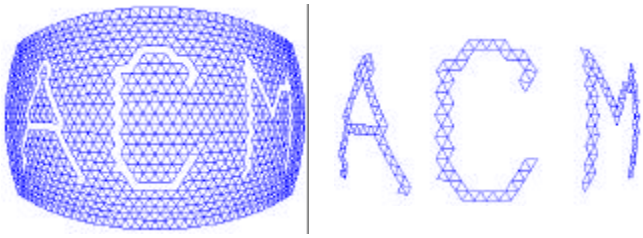
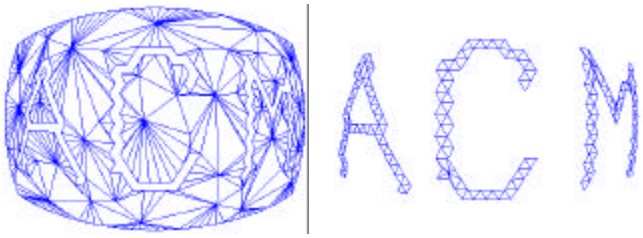


Figure 15. Pattern is embedded by peeling off a polygonal strip *S* in a desired pattern out of a given mesh *M*.



(a) The stencil mesh (1815 triangles) left after peeling off the strip (right) from the original mesh.

(b) The triangle strips cut in patterns of three letters (207 triangles total).



(c) The stencil mesh after polygonal simplification (459 triangles).

(d) Triangle strips after polygonal "simplification" (213 triangles).

Figure 16. A mesh generated from a curved surface has been stenciled with three letters (a). Letters are cut off as triangular strips (b). Boundary vertices and edges are preserved in the stenciled mesh and in the polygonal strips after polygonal simplification.

in the output mesh (Figure 17a). This pattern is hardly visible if displayed with a smooth shading (e.g., Gouraud shading) using proper vertex normal vectors calculated form the original curved surfaces. The pattern becomes visible when the data is displayed by using wire mesh rendering.

This watermark withstands practically every geometrical transformations. The algorithm is resistant but not immune to polygonal simplification and other topology

manipulations. Figure 17b shows an example of effects of polygonal simplification in which the number of triangles in the original mesh has been less than halved by a polygonal simplification algorithm. The pattern will be destroyed eventually due to the simplification. However, a visible watermark may be just enough to deter unauthorized use of the data.

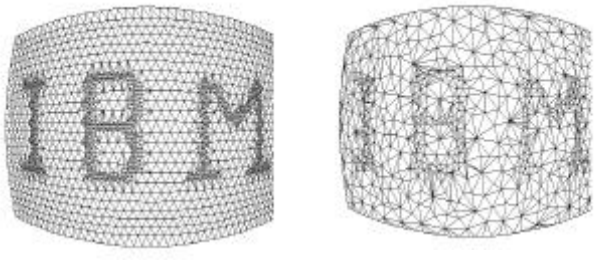
Note that many topological embedding methods can be combined with geometrical embedding methods. In fact, the "IBM mesh" model used in the examples of Table 2 and Table 3 is the mesh shown in Figure 17a. Combining multiple embedding methods, each with its own strength and weakness, is a possible approach to increase utility of the data embedding.

4. Summary and future Work

In this paper we presented fundamental techniques and example of algorithms for embedding information into 3D polygonal models. A review of related work and a discussion on requirements and on selection of target objects for embedding data into 3D models of geometry were given. Next, we presented fundamental methods for embedding data into a polygonal model, namely, geometrical and topological embedding primitives, and methods for introducing order into a set of embedding primitives. Finally, we described some simple data embedding algorithms and results from their implementations to demonstrate that a data embedding into 3D polygonal model of geometry is a practicable technique.

Some of the algorithms described in this paper may be useful for inventory of 3D models or for notification of copyrights to cooperative users. However, these algorithms lack many qualities desired for realistic applications. Probably the most important deficiency of the algorithms presented in this paper is the lack of robustness.

Development of more robust data embedding algorithms will be a major focus of our future work. We would also like to broaden the class of target objects for watermarking, for example, to include parameters defining curved surfaces or vertex normal vectors. Adaptation of embedding algorithms to realistic application models, for example, by adding security through stego-keys, is another area to be examined. Data embedding for 3D models whose intended purpose is not simple viewing is another interesting topic for investigation.



(a) Pattern is embedded by reducing mesh size by 1/4 (2996 triangles).

(b) The mesh after polygonal simplification (1374 triangles).

Figure 17. Pattern embedding and effect of a mesh simplification algorithm on the embedded pattern.

Acknowledgements

The authors would like to thank Mei Kobayashi and Shuichi Shimizu for helpful discussions. The authors also would like to thank anonymous reviewers for valuable comments.

References

[Braudway96] G. Braudway, K. Magerlein, and F. Mintzer, Protecting Publicly-Available Images with a Visible Image Watermark, *IBM Research Report*, TC-20336 (89918), January 15, 1996.

[Cox95] I. J. Cox, J. Kilian, T. Leighton, and T. Shamoon, Secure Spread Spectrum Watermarking for Multimedia, *Technical Report 95-10*, NEC Research Institute, 1995.

[Farin96] G. Farin, *Curves and Surfaces for CAGD: a Practical guide*, Fourth edition, Academic Press, 1996.

[Hartung97] F. Hartung, B. Girod, Copyright Protection in Video Delivery Networks by Watermarking of Pre-Compressed Video, *Lecture Notes in Computer Science*, Vol. 1242, pp.423-436, Springer, 1997.

[ISO96] ISO/IEC JTC1 SC24/N1596 CD #14772 Virtual Reality Model Language (VRML 2.0)

[Masuda96] H. Masuda, Topological operations for non-manifold geometric modeling and their applications, *Ph. D dissertation*, Department of Precision Machinery Engineering, University of Tokyo, 1996 (in Japanese).

[O'Rourke94] J. O'Rourke, *Computational Geometry in C*, Cambridge University Press, 1994.

[O'Ruanaidh96] J. J. K. O'Ruanaidh, W. J. Dowling, F. M. Boland, Watermarking Digital Images for Copyright Protection, *IEE Proc.-Vis. Image Signal Process.*, Vol. 143, No. 4, pp. 250-256, August 1996.

[Pfitzmann96] B. Pfitzmann, Information Hiding Terminology, in R. Anderson, Ed., Lecture Notes in Computer Science No.1174, pp347-350, Springer-Verlag, 1996.

[Schneier96] B. Schneier, *Applied Cryptography*, Second edition, John Wiley & Sons, Inc., New York, 1996.

[Smith96] J. R. Smith and B. O. Comiskey, Modulation and Information Hiding in Images, in R. Anderson, Ed., *Lecture Notes in Computer Science* No.1174, pp. 207-296, Springer-Verlag, 1996.

[Tanaka90] K. Tanaka, Y. Nakamura, and K. Matsui, Embedding Secret Information into a Dithered Multilevel Image, *Proc. 1990 IEEE Military Communications*

Conference, pp. 216-220, 1990.

[Taubin96] G.Taubin, Geometric Compression Through Topological Surgery, *IBM Research Report*, RC-20340 (89924), January 16, 1996.

[Walton95] S. Walton, Image Authentication for a Slippery New Age, *Dr. Dobb's Journal*, pp. 18-26, April 1995.

[Weiler86] K. Weiler, The Radial Edge Structure: A Topological Representation for Non-Manifold Geometric Boundary Modeling, *Geometric Modeling for CAD Applications*, North Holland, pp. 3-36, May 1986.

[Zhao96] J. Zhao and E. Koch, Embedding Robust Labels into Images for Copyright Protection, *'l Congress on Intellectual Property Rights for Specialized Information, Knowledge, and New Technologies* August 1995.