To be published in *Computer Communications*, Elsevier Science B. V.

Data Embedding Algorithms for Geometrical and Non-Geometrical Targets in Three-Dimensional Polygonal Models

Ryutarou Ohbuchi[1], Hiroshi Masuda[2], and Masaki Aono[1]

[1] IBM Tokyo Research Laboratory, 1623-14 Shimo-tsuruma, Yamato-shi, Kanagawa, 242-8502, Japan

[2] The University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo, 113-8656, Japan.

ohbuchi@acm.org, masuda@nakl.t.u-tokyo.ac.jp, aono@acm.org

Abstract

Three-dimensional (3D) graphics is about to become a full-fledged multimedia data type, prompted by increasing popularity of Virtual Reality Modeling Language (VRML) [15] and imminent standardization of MPEG-4 [16].

This paper presents several algorithms for embedding data in triangular meshes, arguably the most important component in both VRML and MPEG 4 in defining arbitrary shapes. A topology-modifying algorithm described in this paper embeds bit string in connectivity of triangles, while another algorithm cuts out patterns from a mesh. Yet another algorithm modifies texture coordinate, a non-geometrical quantity, for embedding.

Watermarks embedded in 3D graphics contents could be used as a tool in managing intellectual property and other issues associated with these contents.

Keywords

Three-dimensional computer graphics, information security, digital watermark.

1. Introduction

Techniques to embed data into digital contents are referred to by such names as *steganography*, *information hiding*, *visible watermarking*, *invisible watermarking*, *fingerprinting*, or *fragile watermarking* depending on their characteristics and application scenarios. These techniques put structures called *watermarks* into digital contents (e.g., images) in such a way that the structures do not interfere with intended use (e.g., viewing) of the contents. We will use a neutral term *data embedding* throughout this paper since our algorithms simply embed data into three-dimensional (3D) polygonal meshes without assuming specific application scenarios.

Embedded watermarks can be classified by its *visibility* (or, more generally, *perceptibility*) and *robustness*, as suggested by Mintzer, et al. [1]. A *visible watermark* is made intentionally visible to serve their purposes, for example, to deter a third party from unauthorized sales of contents. An *invisible watermark* is imperceptible without processing by using computer program and other mechanical means. A *robust watermark* should resists both intentional and

unintentional modifications of the watermarked content. A *fragile watermark*, on the other hand, is designed to be destroyed by intentional modifications so that it could detect tampering of the content. Here, *unintentional modifications* are the kind a content should expect during a course of its intended use, while *intentional modifications* are the kind that are applied with an intention of destroying or altering the watermark**.**

Data embedding has many potential applications. Robust, visible, yet unobtrusive

new embedding target type. In the next section, fundamentals of data embedding in 3D polygonal meshes are discussed. For completeness, Section 3.1 describes two coordinate modifying algorithms that appeared in our previous paper [19]. In Section 3.2, this paper will present two algorithms that target vertex topology (i.e., connectivity of vertices) of 3D polygonal meshes. These algorithms compliment our previous paper that emphasized algorithms that modifies vertex coordinates of polygonal meshes. In section 3.3, this paper presents an algorithm that modifies non-geometrical quantity of polygonal meshes, namely, texture coordinate, for embedding. This algorithm expands the type of embedding targets. We will conclude this paper in Section 4.

2. Data Embedding Fundamentals

While a 3D model could contain diverse data object types, we argued in [17, 19] that definitions of shapes are the best target for data embedding since it is the least likely component to be removed from a 3D model. Of many representations of shapes [20], we chose 3D polygonal mesh as the embedding target, since it is the most important component in both VRML and MPEG 4 for defining shapes of objects.

Shape of a 3D polygonal mesh is defined by two components, *vertex coordinates* and *vertex topology*. Vertex coordinate combined with vertex topology defines more complex geometrical primitives, that are, lines, polygons, and polyhedrons. These geometrical primitives have their own quantities such as length of a line segment and volume of a polyhedron. We call these *geometrical quantities*. These geometrical primitives also have their own topology, which are, for example, connectivity of vertices and triangles. These topology and geometrical quantities are the most important targets for embedding in 3D polygonal meshes.

These geometrical primitives typically have additional *non-geometrical quantity* associated with them. Per-vertex color, per-face normal vector, per-vertex texture coordinate, per-face transparency, or per-volume refractive index, are examples of these non-shape-defining attributes. While less crucial than the former two shape-defining attributes, these non-geometrical quantities could be a good target for embedding.

Our data embedding algorithms presented in this paper modify one of these three attributes of polygonal meshes, that are, *geometrical quantity*, *topology*, or *non-geometrical quantity* to embed data. In the following, a unit of such modification is called an *embedding primitive*. These embedding primitives are arranged in some manner so that an ordered set of embedding primitives encodes a significant amount of information.

2.1 Modification Primitives

(1) Geometrical quantity

A geometrical-quantity embedding primitive consists of quantity or quantities derived from vertex coordinate. One of these quantities is modified to embed data. Vertex displacement resulting from the modification is typically very small so that displacements of vertices do not affect the intended uses of the model.

The simplest primitive of this kind is vertex coordinate. However, information encoded by directly modifying vertex coordinate can be destroyed by almost any geometrical transformation. Many applications of watermark demand robustness against certain class of unintentional disturbances. In case of polygonal meshes, their watermarks must often withstand a limited class of geometrical transformations, e.g., affine transformations.

In order to make watermarks robust against a given class of geometrical transformations, we proposed to use quantities derived from coordinate values that are *invariant* to the given class of geometrical transformations. For example, a ratio of volume of two tetrahedrons is invariant to affine transformation. This quantity is used in the algorithm detailed in [17, 19]. (This algorithm is described also in Section 3.1.2 of this paper.) As another example, cross-ratio of four points on a straight-line [21] is invariant to projection transformation, which is a more general class than affine transformation.

(2) Topology

Watermarks can be embedded by changing topology, that is, connectivity, of a set of vertices, polygons, and polyhedrons. When topology is modified for embedding, coordinates of geometrical primitives involved may change as a side effect. For example, vertices (and edges) may be displaced, inserted, or deleted. However, in topological embedding, information is embedded primarily in topology.

An example of topological embedding primitive is connectivity of triangles in a triangle strip. This primitive is used in the algorithm that will be described in Section 3.2.1. Another example of topological embedding 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 3.2.2. Yet another example of algorithm that employ a topological embedding primitive, which is described in [17, 19], encodes patterns by using two different mesh sizes, e.g.,  and  .

(3) Non-geometrical quantity

Non-geometrical quantities of polygonal meshes do not define shape but are associated with geometrical primitives. In VRML models, non-geometrical quantities are defined either *per-vertex* (point) or *per-face*. In non-VRML models, non-geometrical quantities could be defined *per-line (-segment)* or *per-volume* bounded by polygons. Non-geometrical quantities associated with vertex, line, face, or volume include color, 2D and 3D texture coordinates, normal vector, and refractive index. All of these attributes, essentially a set of numerical values, can be modified to embed data symbols. By introducing arrangement among such modulated values, significant amount of data can be embedded.

An algorithm that will be described in Section 3.3.1 modifies an attribute of this kind, texture coordinate, to embed data.

2.2. Embedding Primitive Arrangements

For a practical data embedding, multiple embedding primitives must usually be arranged so that a collection of embedding primitives functions as a watermark to store a substantial amount of information. An example of arrangements of embedding primitives for a 3D triangular mesh surface is a 1D arrangement of triangles on the mesh generated by sorting triangles according to their areas. Another example is a 2D arrangement of triangles of a (irregularly tessellated) triangular mesh surface based on the topology (adjacency) of triangles.

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 general 3D polygonal mesh surfaces, arrangement of embedding primitives is somewhat more involved.

Arrangements of both geometrical and non-geometrical embedding primitives can be established for 3D polygonal meshes by using one of the following; *topology*, *geometrical quantity*, or *non-geometrical quantity* of geometrical primitives. An arrangement derived from an attribute could be used to arrange primitives of another kind. For example, topological arrangement could be used to arrange face colors.

A *topological arrangement* method employs topological adjacency, such as the adjacency of vertices, to arrange embedding primitives. For example, generating a spanning tree of vertices on the mesh and traversing the tree from its root will create a one-dimensional ordering of vertices [22]. A *quantitative arrangement* method employs a set of quantities, either geometrical or non-geometrical, for arrangement. Hashing vertex coordinate values onto a finite interval of integer domain would create a simple, generally sparse, 1D arrangement. Alternatively, sorting the set of numerical values by using comparison operator will create a dense one-dimensional ordering.

A proper arrangement of embedding primitives often requires an *initial condition*. For example, to uniquely determine a topological arrangement of vertices by using vertex tree, an initial vertex and an initial traverse direction must be found. Obviously, both arrangement and initial condition must be robust against expected disturbances, such as geometrical transformations, or the watermarks will be lost.

Note that the mapping from embedded data (either a symbol sequence or a pattern) to an arrangement of embedding primitives does not have to be straightforward. For example, in order to secure the embedded data, the mapping can be intentionally scrambled by using a cryptographic technique with a stego-key.

3. Embedding Algorithms for Polygonal Meshes

In this section, we will present algorithms that target polygonal mesh surfaces for data embedding. All the algorithms in this section are implemented by using a kernel for a non-manifold modeler [23]. The system employs *radial edge* structure [24] to represent the topological relationship among vertices, edges, faces, and regions.

3.1. Algorithms Based on Geometrical Quantity Modifications

This section describes two algorithms that embed data in polygonal mesh surfaces by ultimately modifying their vertex coordinates. These two algorithms are describe in detail in [19], but are presented here for completeness.

3.1.1. Triangle similarity quadruple embedding

A pair of dimensionless quantities, for example, $\{e_{14}/e_{24}, h_4/e_{12}\}$ in Figure 2, defines a set of similar triangles. The algorithm described in this section, Triangle Similarity Quadruple (TSQ) algorithm, uses such dimensionless quantity pair as the geometrical embedding primitive to watermark triangular meshes.

The TSQ algorithm can be classified as a public watermarking scheme. Watermarks produced by the TSQ algorithm withstand translation, rotation, and uniform-scaling transformations of the stego-polygonal-meshes. An embedded message is resistant to resection and local deformation if it is repeatedly embedded over a mesh. The watermarks are destroyed, among other disturbances, by a randomization of coordinates, by a more general class of geometrical transformation, or by a topological modification such as re-meshing.

In order to realize subscript ordering, the algorithm uses a quadruple of adjacent triangles in the configuration depicted in Figure 2 as a Macro-Embedding-Primitive (MEP). Each MEP stores a quadruple of symbols *{Marker, Subscript, Data1, Data2}*. In Figure 2, the triangle marked *M* stores a marker, *S* stores a subscript, and D_1 and D_2 stores data values. A marker is

a pair of values that identifies MEPs. As mentioned above, this public watermarking scheme does not require cover-polygonal-mesh for extraction. However, the marker value pair is necessary for extraction. A watermarked mesh would contain multiple MEPs to embed a significant amount of data as shown in the example of Figure 3. While each MEP is formed by topology, a set of multiple MEPs is arranged by quantity of the subscript.

The TSQ algorithm embeds a message according to the following steps. (For the detailed explanation and execution examples, please refer [12].)

(1) Traverse the input triangular mesh to find a set of four triangles to be used as a MEP. MEPs must not share edges or vertices to avoid interference.

(2) Embed the marker value by changing a dimensionless quantity pair in the center triangle of the MEP. In Figure 2, it is $\{e_{14}/e_{24}, h_4/e_{12}\}$. This modifies positions of vertices v_1, v_2, and v_4.

(3) Embed a subscript and two data symbols in a similar manner by displacing vertices v_0, v_3, and v_5. 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 $\{e_{45}/e_{25}, h_5/e_{24}\}$.

(4) Repeat (1) to (3) above until all the data symbols of the message are embedded.

For each triangle, 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. In order to embed the message repetitively, steps (1) to (4) are repeated many times.

Figure 4 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.

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 embedded 210 bytes of data, that is, 0.15 byte/triangle, in the model of Figure 4, which consisted of 1406 triangles. Experiments using seven polygonal mesh models showed that the TSQ algorithm was able to embed 0.15-0.18 byte/triangle.

3.1.2. 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 a global one-dimensional arrangement on a triangular mesh in order to embed a sequence of symbols.

The TVR algorithm is a public watermarking scheme. The watermarks produced by the TVR algorithm withstand affine transformation of stego-polygonal-meshes. The watermarks are destroyed, among other disturbances, by topological modifications such as resection and re-meshing, randomization of vertex coordinates, and geometrical transformations more general than affine transformation (e.g., a projection transformation). However, repetitive embedding could make the watermark somewhat resistant to some of these disturbances. Repetitive embedding in a triangular mesh can be achieved, for example, by cutting the mesh

into multiple sub-meshes and embedding a message into each sub-mesh.

The crux of the TVR algorithm is establishing 1D arrangement of embedding primitives, whose process is illustrated in Figure 5 by using a simple example.

Given a triangular mesh, an initial vertex, and an initial traverse direction, a *vertex (spanning) tree* is generated on the mesh (Figure 5b). (A method to find the initial vertex and initial traverse direction will be explained later.) The initial vertex becomes the root of the vertex tree. The vertex tree is converted to a tree of triangles, *a triangle (spanning) tree*, which is then traversed in depth-first order to generate 1D *triangle sequence* (Figure 5e). These steps are similar to the one employed in [20] to compress triangular mesh data.

Once the sequence of triangles is obtained, it is converted to a sequence of tetrahedrons by finding a point to be used as a common apex of the tetrahedrons. For example, the centroid of three vertices of the first triangle in the triangle sequence is used as the apex. The triangle used to compute common apex must be put aside so that its vertices are not modified later due to embedding of data. Note that the vertices on the mesh should not be coplanar, or volumes of all the tetrahedrons would be zero. (The vertices on the mesh of Figure 5 look coplanar, because meshes that look coplanar are easier to draw.) In Figure 5, centroid of vertices of the triangle numbered 1 is used as the common apex in forming tetrahedrons numbered 2 to 12 in Figure 5f. (The example of Figure 5 seems to indicate the vertices on the mesh to be coplanar. This is for the ease of illustration only. Vertices must not be coplanar for the algorithm to work.)

It is now easy to generate a sequence of ratios of volumes of tetrahedrons. For example, volume of the first tetrahedron in the tetrahedron sequence is used as a common denominator and volumes of remaining tetrahedrons are used as numerators. In Figure 5f and in Figure 5g, volume of the tetrahedron 2 is used as the common denominator, and the remaining tetrahedrons, numbered 3 to 12, are used as the numerators. Each ratio is modified according to the symbol to be embedded in the ratio. Modification of these ratios subsequently alters vertex coordinates.

A pair of initial vertex and initial traverse direction is determined by finding an initial edge. To find the initial edge, the algorithm computes, for every edge in the mesh, the volume of the tetrahedron subtended by two triangles adjacent to the edge. The algorithm selects as the initial edge an edge whose tetrahedron has the largest volume. (The tetrahedrons used to select the initial edge are unrelated to those used to embed symbols.) Correct initial vertex and initial traverse direction pair is found by trial-and-error method; the pair that yielded correct lead-in symbols is chosen as the right pair.

Figure 6 shows a model watermarked with the TVR algorithm in which modified triangles are rendered with dark gray. Figure 7 shows an example of data embedding by using the TVR algorithm with repetitive embedding. Affine transformation (Figure 7c) and resection (Figure 7d) did not destroy the embedded message. The TVR algorithm embedded 0.08 byte/triangle in the face model of the Figure 6, and 0.18 byte/triangle in the cow model of Figure 7. An experiment with seven polygonal mesh models showed that the embedding data density ranged from 0.05 byte/triangle to 0.2 byte/triangle.

3.2. Algorithms Based on Topological Modifications

This section presents two new algorithms that embed data by using topology of polygonal meshes. [1]

[1] These two algorithms first appeared in [18].

3.2.1. Triangle strip peeling symbol-sequence-embedding

The *Triangle Strip Peeling Symbol sequence (TSPS)* embedding algorithm that will be presented in this section is a public watermarking scheme based on a topological embedding primitive. It employs, as its embedding primitive, an adjacency of a pair of triangles in a triangle strip, each of which encodes a bit of information. One-dimensional arrangement of embedding primitives is induced by the adjacency of triangles on the triangle strip. To recognize the triangle strip with watermark, the strip is peeled off from the original mesh.

Since both embedding primitive and arrangement are topological, watermarks produced by the algorithm are immune to geometrical transformation. Repetitive embedding makes the watermarks resistant to resection. The watermarks can be destroyed by topological manipulations, for example, by polygon simplification algorithms. A disadvantage of this algorithm is its low space efficiency compared to many algorithms based on geometrical primitives.

Inputs to this embedding algorithm are an orientable triangular mesh and a message bit string. The TSPS embedding algorithm embeds data according to the following steps. (See Figure 8.)

(1) Starting from an edge *e* selected from the input mesh *M*, grow a triangle strip *S* on *M* by using the message bit-string to determine the direction of growth of the strip. Observe that a triangle at the end of (current) strip has two "free" edges, i.e., edges that are not adjacent to triangles of the current triangle strip. Since *M* is orientable, these two edges can be ordered on the triangle by traversing the edges in a fixed order (either counterclockwise or clockwise). Depending on the data bit, choose one of the two free edges as the edge to be shared with the next triangle of the strip. (See Figure 9.)

(2) "Peel off" the triangle strip *S* from *M* by splitting all the edges and vertices 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*.

The edge *e* serves as the initial condition for finding the triangle strip. Arrangement of embedding primitives is induced naturally by the connectivity of triangles on the triangle strip. Since the peeled strip caps the hole completely, proper colors and vertex normal vectors make the watermark invisible.

Figure 9 shows an example of a triangle strip. The strip drawn with solid lines, which start at edge *e*, embeds a bit string "10101101011" in a sequence of 12 triangles. Each bit of the bit string steers the direction of growth of the triangle strip. If the last bit of the string is "0" instead of "1", the last triangle will become the one that is drawn with broken lines.

Steering by message bit strings produces strips whose shape may not fit in a given mesh, depending on a given bit string. In the example of Figure 9, a message bit string with all "1" would keep steering the strip to the left. If the message string is sufficiently long, the strip will either hit the boundary of the mesh or circle back to itself. To avoid this problem, shapes, locations and orientations of the strips must be controlled carefully. We manipulate the shape of the triangle by using *steering symbols*. A steering symbol is a bit that does not carry information but simply steer direction of growth of a triangle strip. Steering symbols are interleaved with data symbols, that are, symbols that encode embedded data, in order to control shape of triangle strips. Obviously, steering symbol halves the embedding data capacity. Our current implementation determines initial locations, directions of growth, and shapes of triangle strip manually.

Extraction of message is carried out according to the following steps.

(1) Traverse the watermarked mesh 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) Starting from the initial edge, traverse the triangle strip to the open end as embedded bits are extracted.

Figure 10 shows a simple example of TSPS embedding, in which a triangle strip of length 27 is peeled off from 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. As another example, a model of triceratops (499 triangles) in Figure 11 is marked with a triangle strip of length 19 triangles, which encodes 9 bits. (The colors of the strips in these examples are intentionally changed to show their location.)

Watermarks produced by the TSPS embedding algorithm can be erased if a geometrical "mending" program, for example the one similar to [25], which would stitch the triangle strip back to the stencil mesh. Such mending can be prevented to some extent by modifying topology of stego-polygonal-mesh in order to confuse mending algorithms. For example, vertices, edges, and polygons can be added into the stencil mesh R so that finding correspondence of edges and vertices to be stitched together is difficult.

3.2.2. 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 12. 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, given proper colors and normal vectors, watermarks are visually unnoticeable.

Watermarks produced by this algorithm, called Polygon Stencil Pattern (PSP) embedding, are robust against many polygonal simplification algorithms. This is because vertices on the boundary of polygonal strips and stencil meshes are preserved by many polygon simplification algorithms. If vertices on the boundary are removed or displaced forcefully, as some polygon simplification algorithms do, cracks will appear, diminishing the value of the model.

Figure 13 shows a stenciled polygonal mesh, a cut out triangle strip, and effects of a polygon simplification algorithm on them. The vertices on the boundary of the stencil mesh and the triangle strip are preserved despite a polygon simplification. On the stencil mesh, the simplification reduced the number of triangles from 1735 down to 413. Number (and coordinates) of vertices on the boundary of the triangle strip did not change after the simplification, but the number of triangles is reduces from 287 to 283. Topology of the edges did change after simplification, but the watermark remained. (Note the change at the upper-left corner of the letter "M", for example.)

3.3. An Algorithm Based on Non-geometrical Quantity Modifications

Data can be embedded in non-geometrical quantities, which include per-vertex color, per-vertex texture coordinate, per-face color, per-face normal vector, per-vertex normal vector, per-volume color, and per-volume refractive index. The approach to embedding using non-geometrical attributes is similar to those used for algorithms that employ coordinate embedding primitives; Values of the attributes are modified and the modifications are ordered to embed a significant amount of information.

As an example, a new algorithm that embeds data in texture coordinates of polygonal mesh

will be presented in the next section. A similar algorithm can be used to embed data in other per-vertex attributes, such as vertex colors. Data embedding into per-face attributes of a polygonal mesh surface is also possible; Modify per-face attributes and then arrange modifications.

3.3.1. Texture coordinate modification embedding

A set of texture coordinates associated with vertices of a polygonal model is a good target for data embedding. Proper texture coordinate is crucial for proper rendering of texture mapped objects, and a set of texture coordinates is difficult to regenerate once it is lost. An added benefit is that texture coordinate is not affected by coordinate transformation of vertices, which could make embedding algorithms that target texture coordinates simpler than those that target vertex coordinates.

The embedding algorithm modulates amplitude of texture coordinates based on message symbols. There are many ways to modify amplitudes of texture coordinate values for embedding. A simple method to embed a bit string is as follows.

Let s_i be ith bit of a bit string S. The embedding algorithm modifies a coordinate value x_i (e.g., either u or v) of a texture coordinates by the following steps, given a modulation amplitude A.

$r=x_i-x_i/A;$

if s_i='0' **then** $b=A/4$ **else if** s_i='1' **then** $b=A*3/4$;

$x_i= r+b;$

Alternatively, we can work in binary representation of floating point numbers. To modulate a number by a bit, simply replace selected bit in the number with the bit to be embedded. Regardless of the actual modulation methods, we can make two such modulations per 2D-texture coordinates. Thus, if we embed one bit per floating point number, we can embed $2N$ bits into N 2D-texture coordinates.

In our prototype implementation, we used the order of appearance of texture coordinates in the input file as the ordering for the embedding. This order is destroyed easily by shuffling the positions of the texture coordinates in the file. If this is a problem, there are alternative methods to introduce ordering into a set of texture coordinates. Since each texture coordinates is associated with a vertex, ordering vertices implies ordering of texture coordinates. In Section 3, we introduced several examples of methods to order vertices. For example, vertices can be ordered topologically by TVR embedding algorithm (Section 3.1.2) or TSPS embedding algorithm (Section 3.2.1). It is also possible to arrange texture coordinate by using a non-geometrical quantity, for example, texture coordinate or vertex color.

Note that watermarks by the TSQ, TVR, and TSPS embedding algorithms do not interfere with texture coordinates, face color, and other non-geometrical attributes of polygonal meshes. Thus, it is possible to combine an attribute-modifying algorithm (e.g., the one described here) with an algorithm that modifies geometry or topology (e.g., TSPS embedding algorithm).

The modulation amplitude A must be small enough so that the embedding does not affect the visual quality of the rendered model. Simultaneously, the amplitude A should be large enough so that the embedding won't be destroyed, for example, by floating-point number representation errors. We conducted experiments to determine relation between modulation amplitudes and quality of rendered images. Some of the results are shown in Figure 14 and Figure 15.

Texture images are a synthetic red-and-white stripe image (256 x 256 pixels), and a

photograph of a human face (512 x 512 pixels). These images were texture-mapped onto a model of a sphere tessellated into 1800 triangles, which contained 961 vertices (and thus 961 texture coordinate). If we modify four bits per single-precision floating-point number, we should be able to embed maximum of 961 bytes in 961 texture coordinates. In this experiment, we embedded a 358 byte long text in the texture coordinates.

In the figures, *Ar* is the modulation amplitude relative to the range of texture coordinate variation on the model. In these examples, the texture coordinates varied in the range [0,1] in both *u* and *v* coordinates so that the maximum variation range of texture was 1.0. Thus, *Ar*=0.1 % means amplitude of 0.001.

In Figure 14, in which the red-and-white stripe texture is used, distortion in the rendered image is perceptible in rendered images when *Ar*=0.5 % (Figure 14c) and *Ar*=1 % (Figure 14d). Complex, less geometrical, texture images reduced perceptibility of texture distortions. Distortions of the human face texture shown in Figure 15 were difficult to perceive. Even for the image with *Ar*=1 % (Figure 15d), a careful comparison with the original image (Figure 15a) was necessary to reveal distortions.

This experiment showed that, if modification amplitude is chosen appropriately, data embedding into texture coordinates is possible without noticeable change in the models rendered appearance.

4. Conclusion

Following an introduction and a description of fundamental approaches to data embedding in 3D polygonal meshes, this paper presented algorithms that embed data in polygonal mesh surfaces.

The paper presented two algorithms that modify topology of polygonal mesh surfaces. One of the algorithms embeds bit strings in connectivity of triangles and the other embeds visual patterns by cutting out the patterns. Watermarks produced by both of these algorithms are robust against any modification of vertex coordinate values. These watermarks are destroyed by topological modifications, such as polygon simplifications. This paper also presented an algorithm that modifies non-geometrical quantities. These quantities are associated with vertices, line segments, faces, or volumes but do not define shapes. Per-vertex texture coordinates, per-vertex and per-face colors, and per-volume transparency are example of this kind of quantities. In particular, an algorithm that modifies per-vertex texture coordinate is presented. With these algorithms, this paper complemented and expanded our previous papers [17, 18, 19] by adding new algorithms and by increasing target object types.

Data embedding algorithms described in this paper are not robust enough for many applications scenarios. However, some of them could be used as they are in some other applications. For example, we are planning to explore application of fragile watermarks for authentication or tamper detection. We will also look for more robust embedding algorithms and more diverse embedding target types.

Acknowledgements

The authors would like to thank Mei Kobayashi and Shuichi Shimizu for helpful discussions. The authors would like to thank Takaaki Murao for the use of his polygon simplification code. The authors also would like to thank anonymous reviewers for valuable comments.

References

[1] F. Mintzer, G. W. Braudway, and M. M. Yeung, Effective and Ineffective Digital Watermarks, Proceedings of the *IEEE International Conference on Image Processing (ICIP) '97*, Vol. 3, pp. 9-12, IEEE, Piscataway, NJ, USA, 1997.
[2] B. Pfitzmann, Information Hiding Terminology, in R. Anderson, Ed., *Lecture Notes in Computer Science* No.1174, pp. 347-350, Springer, Berlin, Germany, 1996.
[3] A. J. Menezes, P. C. van Oorshot, and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, Boca Raton, FL, USA, 1996.
[4] K. Tanaka, Y. Nakamura, and K. Matsui, Embedding Secret Information into a Dithered Multilevel Image, *Proc. 1990 IEEE Military Communications Conference*, pp. 216-220, IEEE, Piscataway, NJ, USA, 1990.
[5] S. Walton, Image Authentication for a Slippery New Age, *Dr. Dobb's Journal*, Miller Freeman, Inc., San Francisco, CA, USA, pp. 18-26, April 1995.
[6] I. J. Cox, J. Kilian, T. Leighton, and T. Shamoon, Secure Spread Spectrum Watermarking for Multimedia, *IEEE Trans. on Image Processing*, Vol. 6, No. 12, pp. 1673-1678, IEEE, Piscataway, NJ, USA, 1997.
[7] W. Bender, D. Gruhl, and N. Morimoto, Techniques for Data Embedding, *IBM Systems Journal*, Vol. 35, Nos. 3 & 4, International Business Machines Corporation, Armonk, NY, USA, 1996.
[8] 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.
[9] J. J. K. O'Ruanaidh, W. J. Dowling and F. M. Boland, Watermarking Digital Images for Copyright Protection, *IEE Proc.-Vis. Image Signal Process.*, Vol. 143, No. 4, pp. 250-256, August 1996.
[10] 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, Berlin, Germany, 1996.
[11] J. Zhao and E. Koch, Embedding Robust Labels into Images for Copyright Protection, Proc. of *the Int'l. Congress on Intellectual Property Rights for Specialized Information, Knowledge, and New Technologies*, Vienna, Austria, Aug. 21-25, R. Oldenbourg, Munich, Germany, 1995, pp.242-241.
[12] F. Hartung and 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, Berlin, Germany, 1997.
[13] M. M. Yeung, F. C. Mintzer, G. Braudway, and A. R. Rao, Digital Watermarking For High-Quality Imaging, Proceedings of the *First IEEE Workshop on Multimedia Signal Processing*, Princeton, NJ, USA, June, 1997, pp. 357-362, IEEE, Piscataway, NJ, USA.
[14] M. M. Yeung and F. Mintzer, An Invisible Watermarking Techniques for Image Verification, Proceedings of the *IEEE ICIP '97*, Vol. 2, pp. 680-683, IEEE, Piscataway, NJ, USA, 1997.
[15] ISO/IEC 14772-1 *Virtual Reality Model Language (VRML)*, International Standard Organization (ISO), 1997.
[16] ISO/IEC JTC1/SC29/WG11 *MPEG-4 Visual* and *MPEG 4 SNHC*, ISO, 1998.
[17] R. Ohbuchi, H. Masuda, and M. Aono, Embedding Data in 3D Models, in Steinmetz, et al. eds, *Lecture Notes in Computer Science* No. 1309, pp.1-11 (Proceedings of the *IDMS '97*, Darmstadt, Germany, September), Springer, Berlin, Germany, 1997.
[18] R. Ohbuchi, H. Masuda, and M. Aono, Watermarking Three-Dimensional Polygonal

Models, *Proceedings of the ACM Multimedia '97*, Seattle, Washington, USA, November 1997, pp. 261-272, Addison-Wesley Publishing, Reading, WA, USA.
[19] R. Ohbuchi, H. Masuda, and M. Aono, Watermarking Three-Dimensional Polygonal Models Through Geometric and Topological Modifications, *IEEE Journal on Selected Areas in Communications*, Vol. 16, No. 4, pp. 551-560, IEEE, Piscataway, NJ, USA, 1998.
[20] M. E. Mortenson, *Geometric Modeling*, John Wiley & Sons, New York, USA, 1997.
[21] G. Farin, *Curves and Surfaces for CAGD: a Practical guide*, Fourth edition, Academic Press, Chestnut Hill, MA, USA, 1996.
[22] G. Taubin, Geometric Compression Through Topological Surgery, *IBM Research Report*, RC-20340 (89924), January 16, 1996.
[23] 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).
[24] K. Weiler, The Radial Edge Structure: A Topological Representation for Non-Manifold Geometric Boundary Modeling, *Geometric Modeling for CAD Applications*, pp. 3-36, May 1986, North Holland, Amsterdam.
[25] A. Gueziec, G. Taubin, F. Lazarus, and W. Horn, Cutting and Stitching: Efficient Conversion of a Non-Manifold Polygonal Surface to a Manifold, *IBM Research Report* RC-20935 (92693), July, 1997.

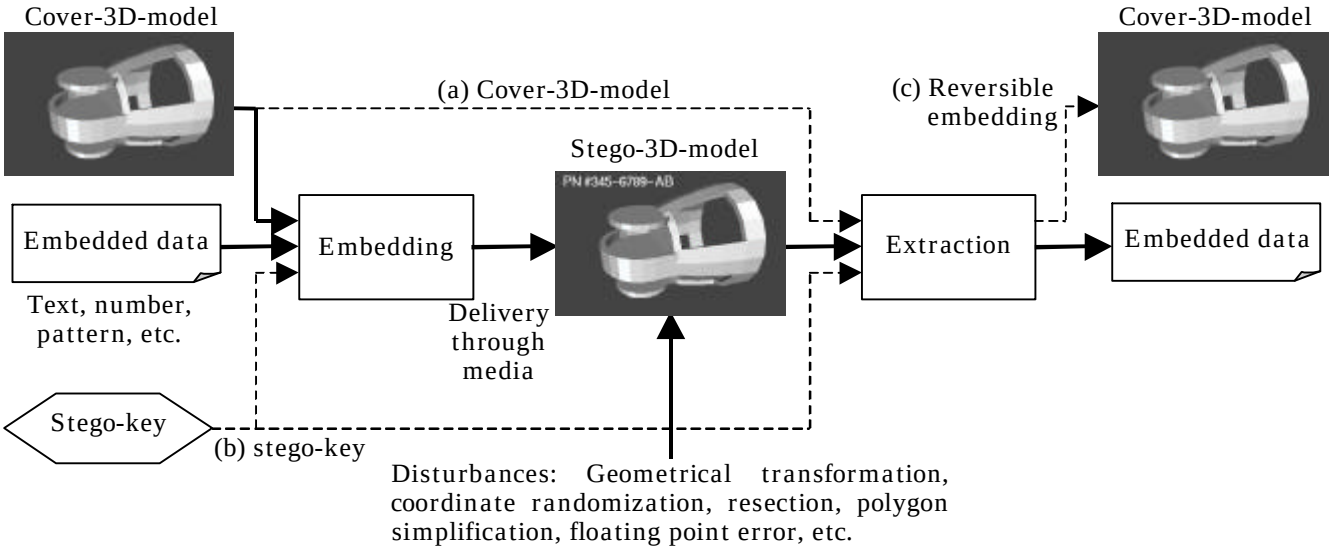
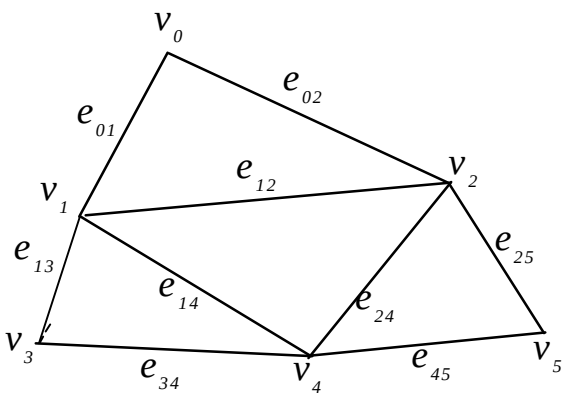


Figure 1. Data embedding in 3D polygonal models. Solid lines indicate data flow of a basic embedding scheme. Dotted lines indicate additional paths of information used in variations of the basic data embedding scheme: (a) extraction that requires cover-3D-model, (b) security provided by using stego-key, and (c) reversible embedding.


v_0
e_{02}
e_{01}
e_{12}
v_2
v_1
e_{13}
e_{25}
e_{14}
e_{24}
v_3
e_{34}
v_4
e_{45}
v_5

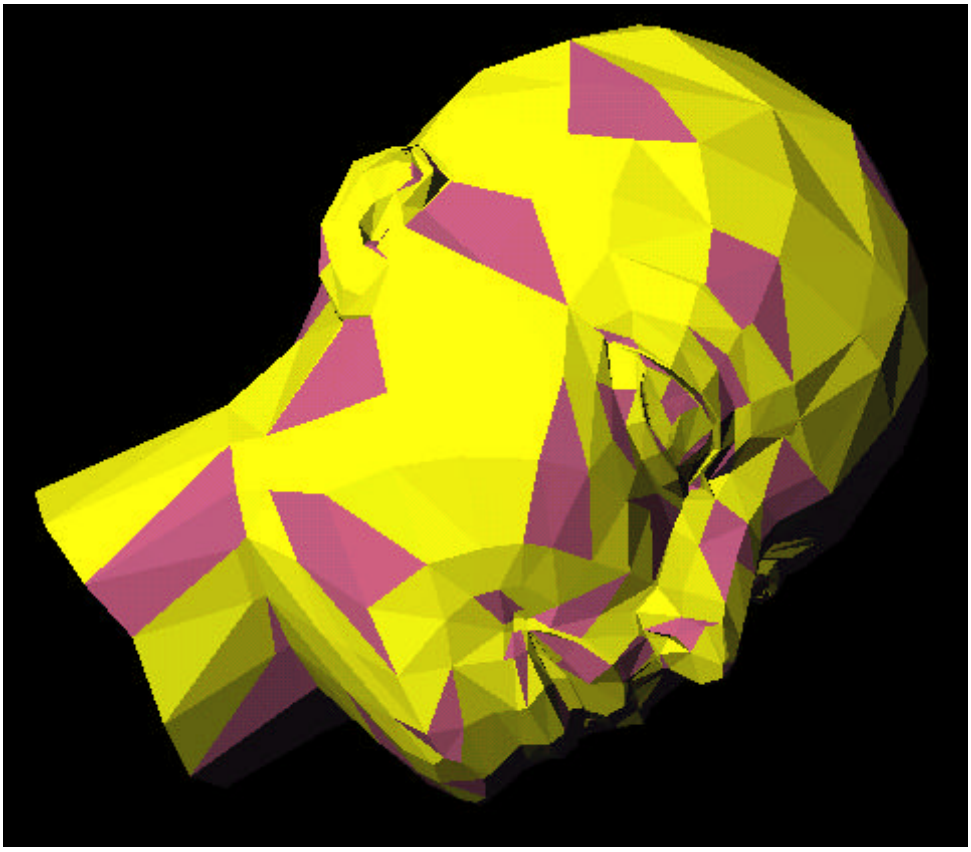
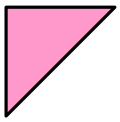


Figure 4. Macro embedding primitives, each of which consists of four adjacent triangles, are shown in dark gray.

To be published in *Computer Communications*, Elsevier Science B. V.



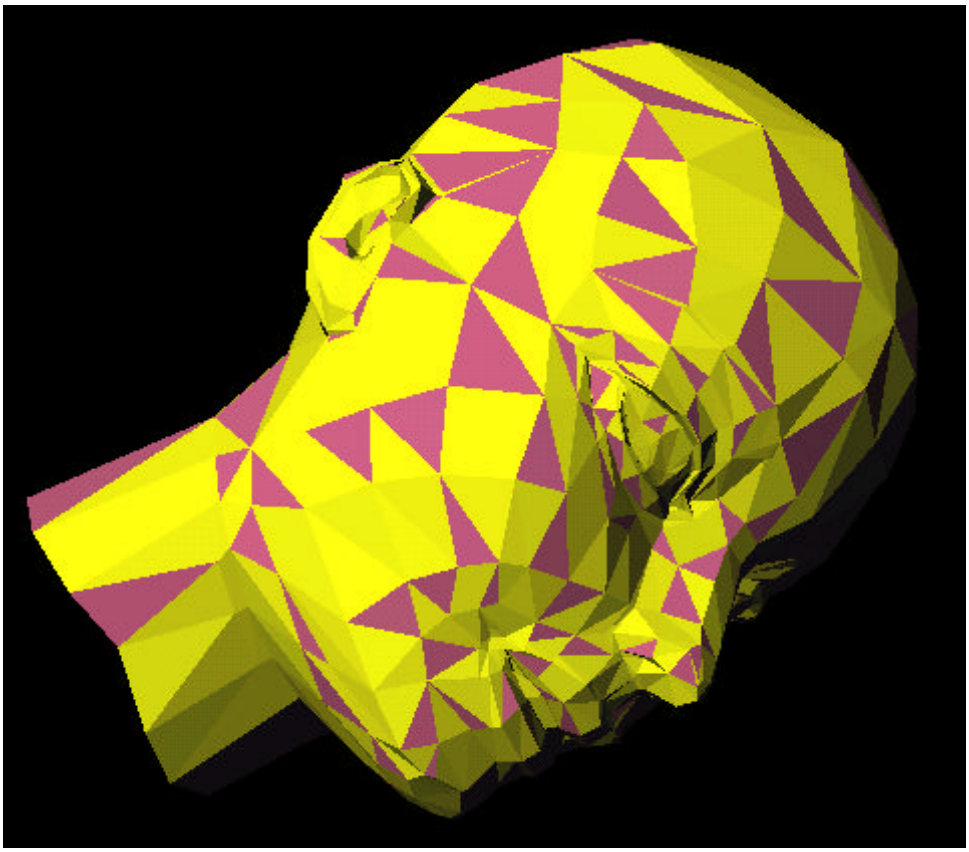
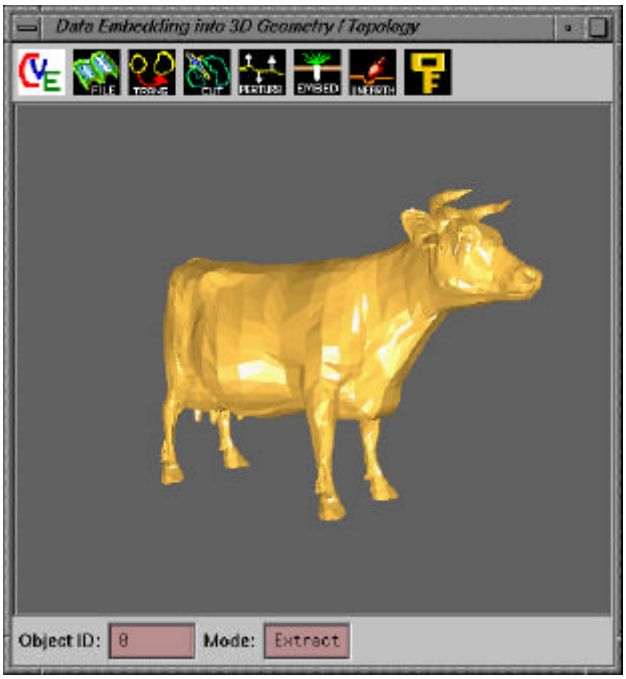


Figure 6. Triangles used for embedding by the TVR algorithm are shown in dark gray.



(a) The original model.

(b) Message embedded.

(c) Affine-transformed.

(d) Cut in half.

Figure 7. (a) Polygonal mesh 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. (The URL is fictitious.)

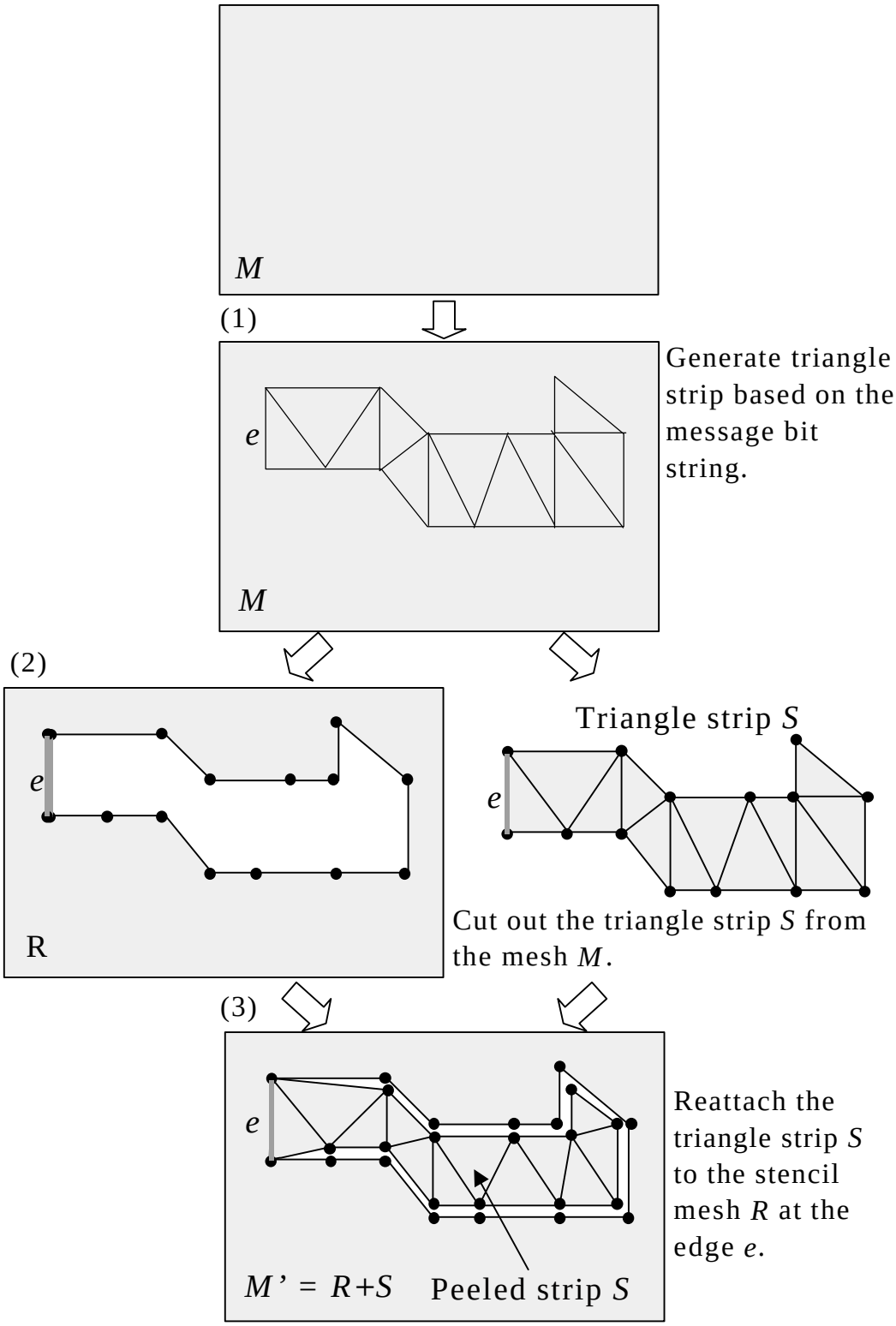


Figure 8. Triangle strip peeling symbol sequence embedding algorithm. (Cracks around the strip in the bottom figure are for illustration purpose only.)

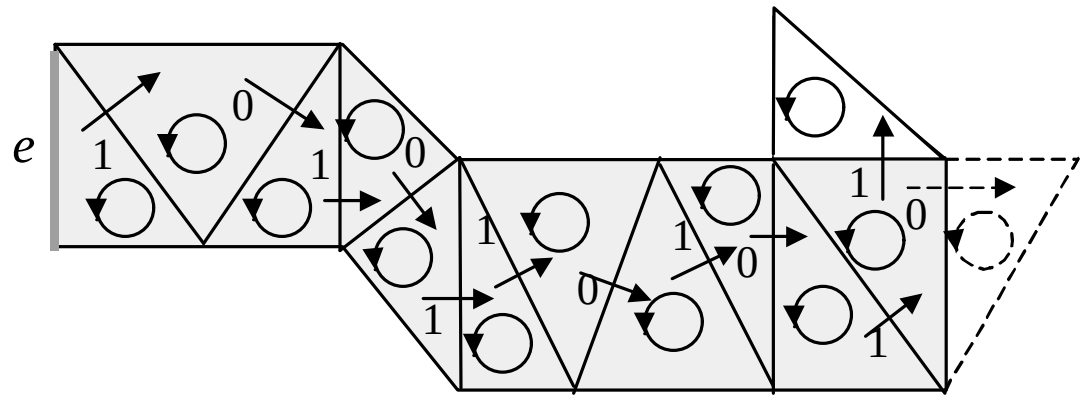


Figure 9. Connectivity of 12 triangles (drawn with solid lines) in a triangle strip encodes the bit string "10101101011" (11 bits). If the bit string is "10101101010" (change in the last bit), the last triangle will be the one drawn with broken lines.

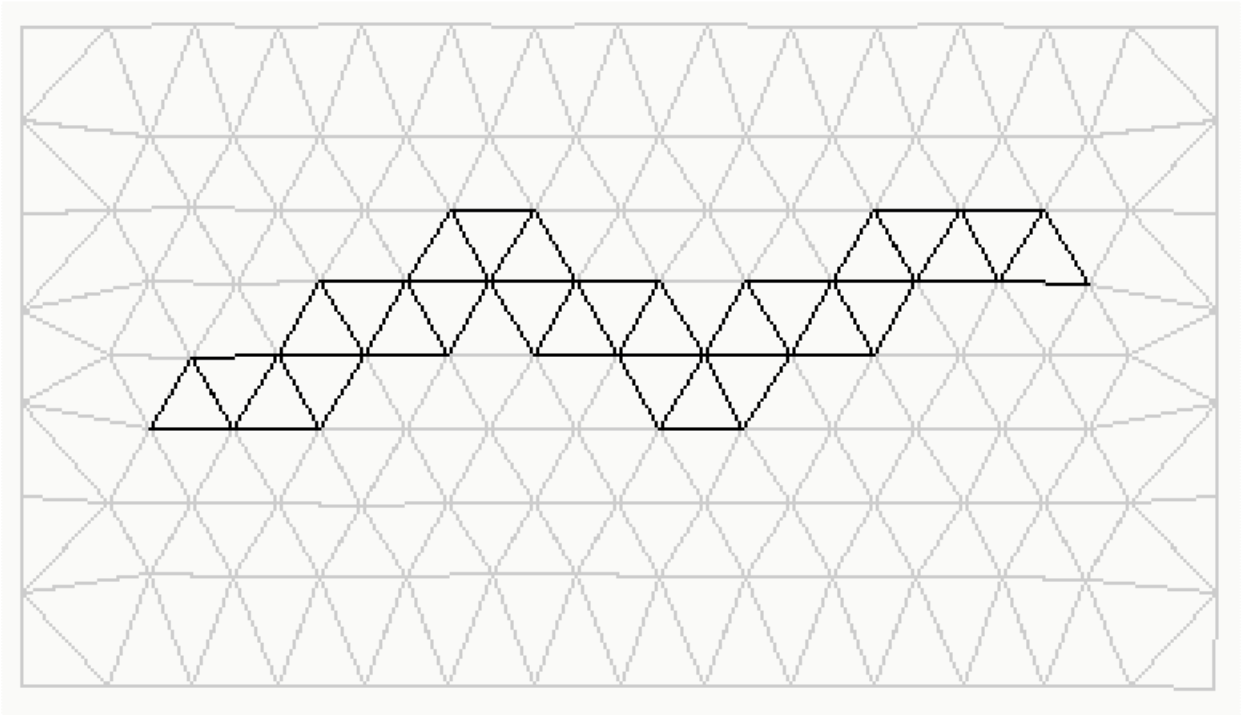


Figure 10. A triangle strip consisting of 27 triangles was cut out from a flat triangular mesh (214 triangles). The triangle strip, displayed in darker gray, encodes 13 data bits interleaved with 13 steering bits.

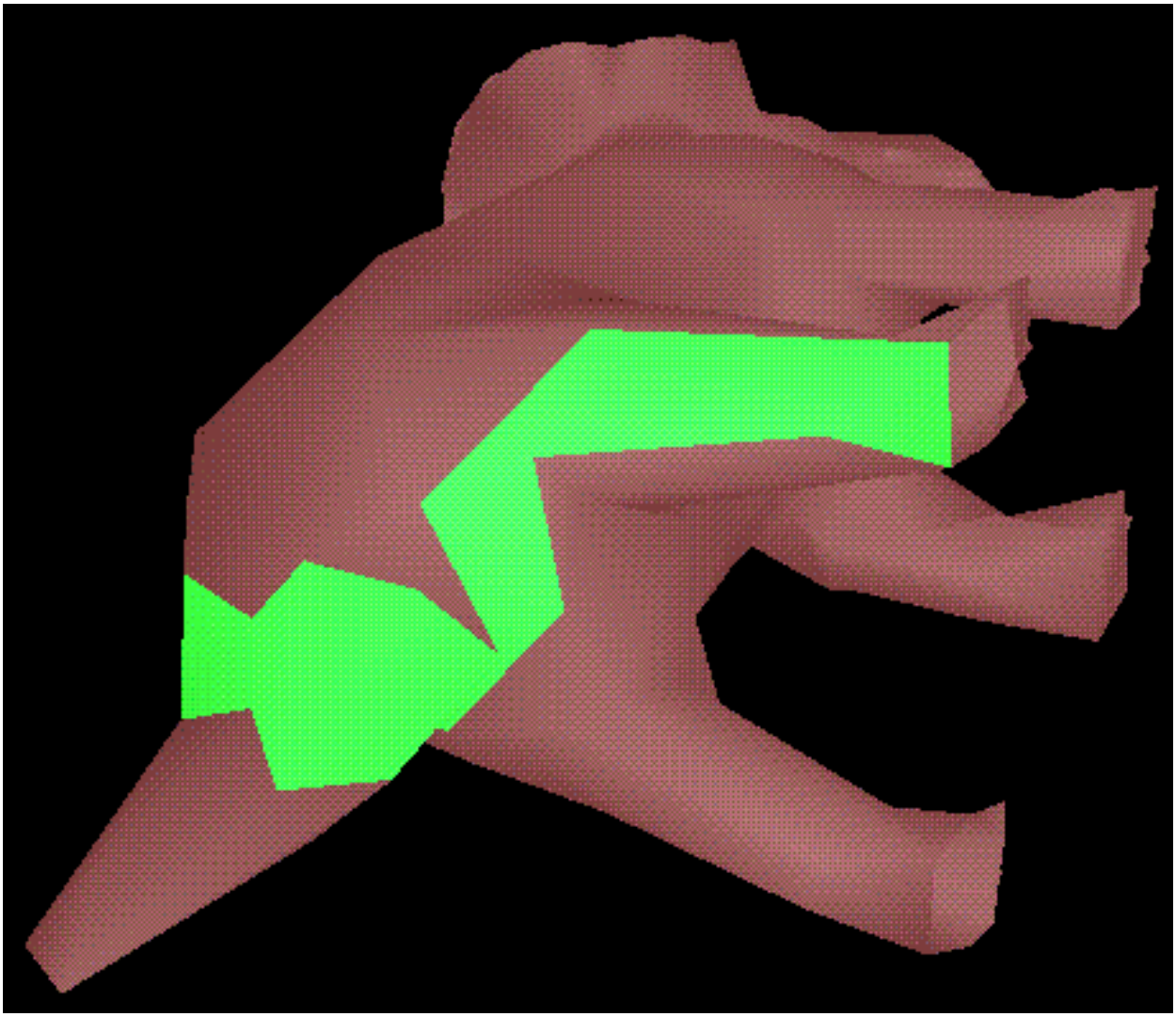
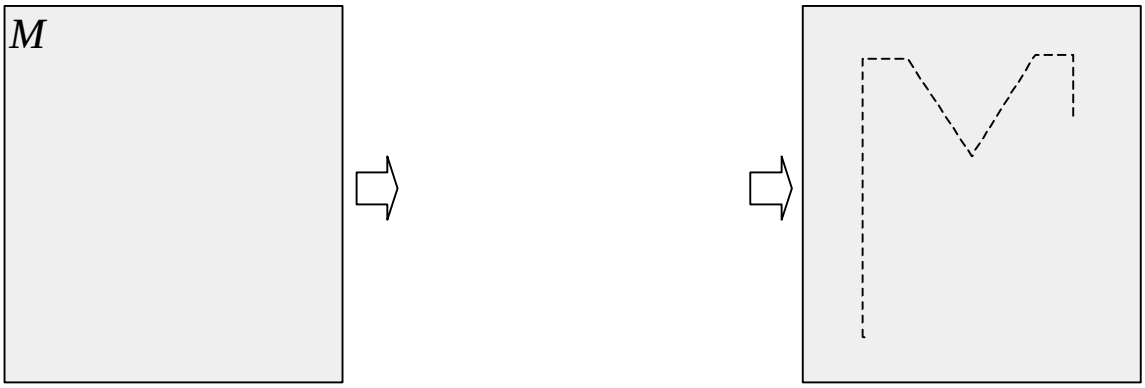
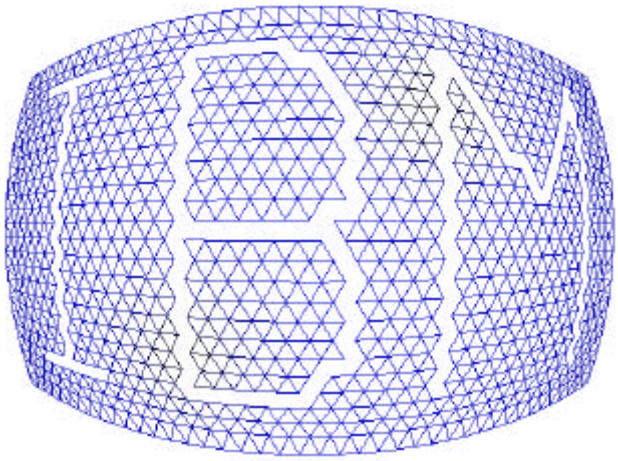
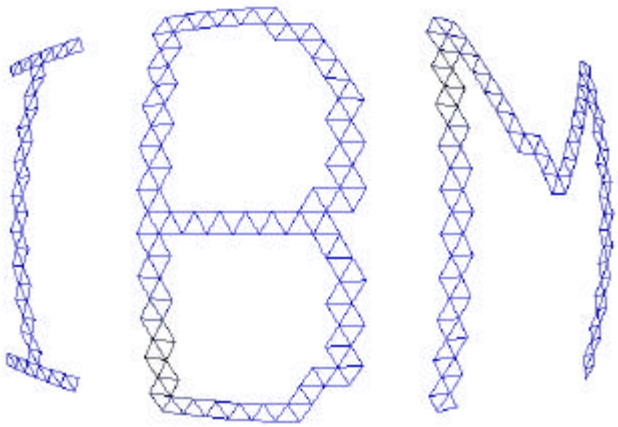


Figure 11. A triangle strip, 19 triangles long and shown in a light gray, is generated and peeled off from a model of a triceratops (499 triangles). (A part of the strip is not visible from this viewpoint.)

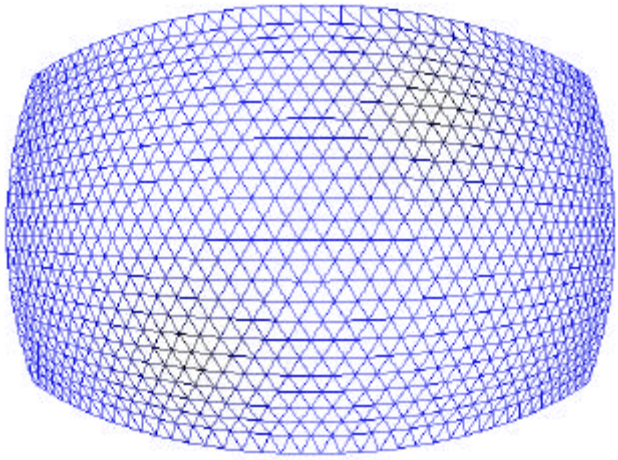

M



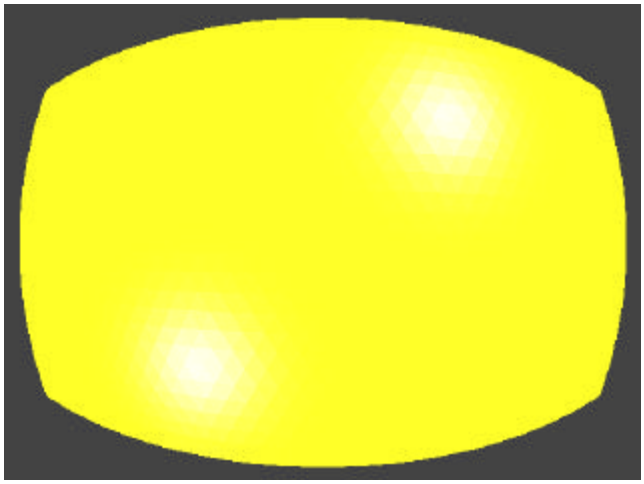
(a) The stencil mesh (1735 triangles) left after cutting out a strip (right) from the original mesh.



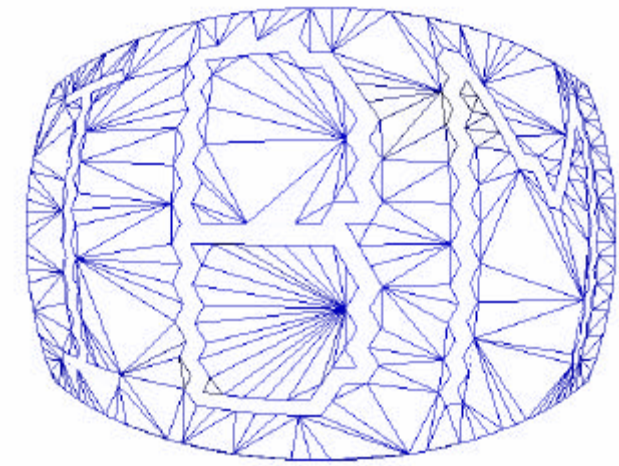
(b) The triangle strips cut in patterns of three letters “IBM” (287 triangles total).



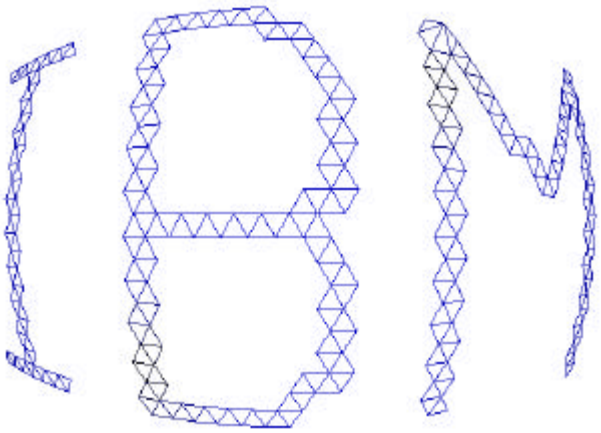
(c) The stencil mesh and the triangle strip displayed together (2022 triangles total.)



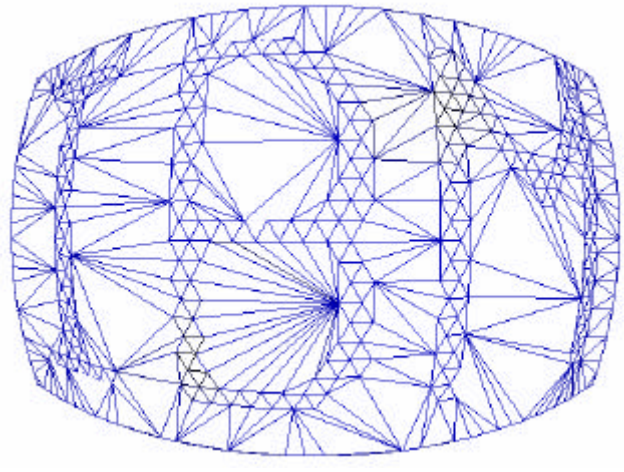
(d) If rendered together, the embedded watermark is not noticeable.



(e) The stencil mesh after polygonal simplification (413 triangles).



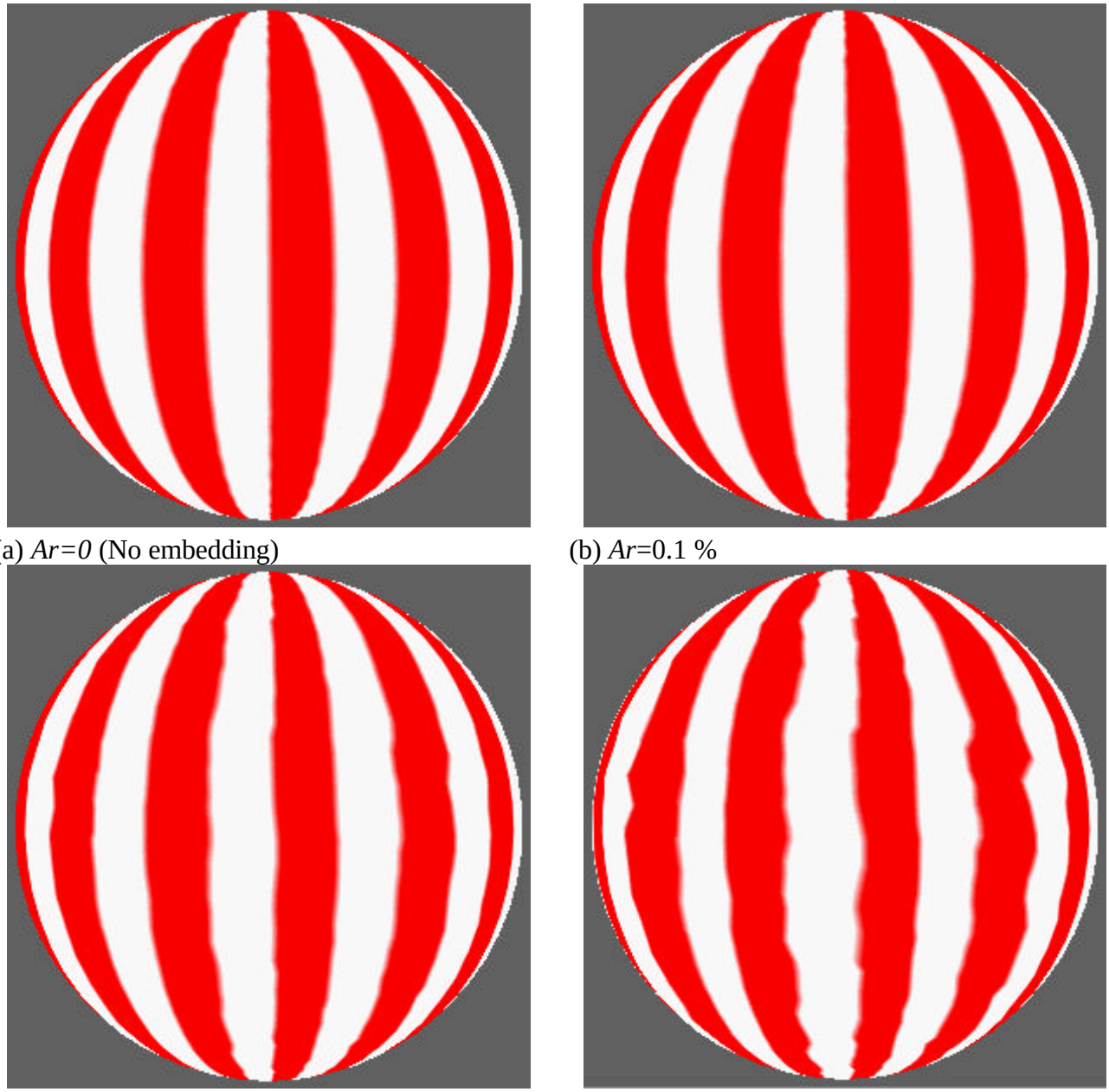
(f) The triangle strip after polygonal simplification (283 triangles).



(g) The stencil and the triangle strip combined, reduced to 696 triangles total.

(h) The mesh to the left rendered with shading. Quality degradation is apparent.

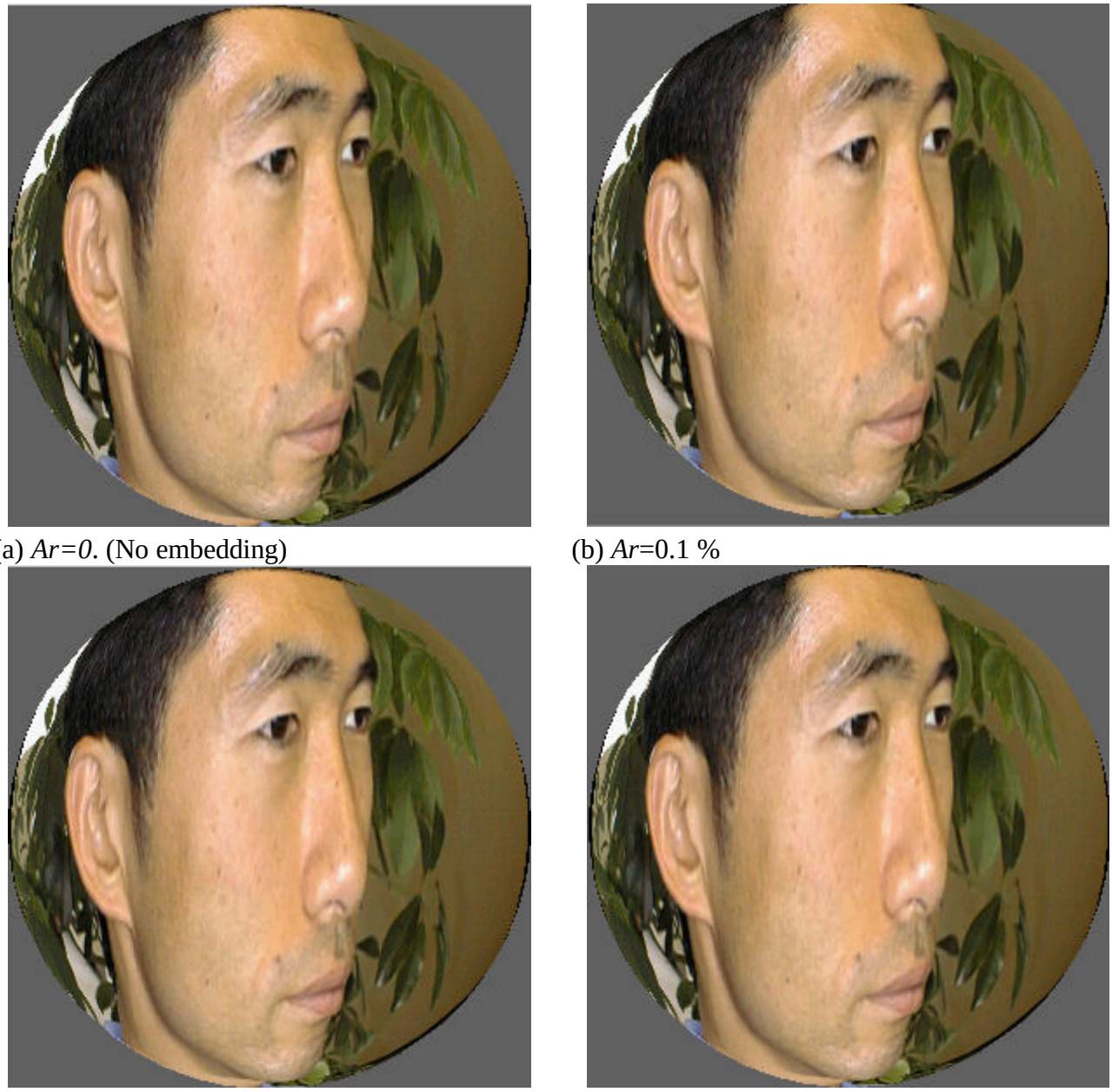
Figure 13. A simple pattern embedding by cutting patterns of a polygonal mesh. The watermark is resistant to many polygon simplification algorithms.



(a) *Ar=0* (No embedding) (b) *Ar*=0.1 %

(c) *Ar*=0.5 % (d) *Ar*=1 %

Figure 14. A red-and-white stripe image is mapped onto a sphere model (1800 triangles). Texture coordinates are modulated with several relative amplitudes *Ar*. Distortions are not perceptible at Ar=0.1 % but become quite noticeable if *Ar* is near or above 0.5 %.



(a) *Ar=0*. (No embedding) (b) *Ar*=0.1 %

(c) *Ar*=0.5 % (d) *Ar*=1 %

Figure 15. A photograph of a human face is mapped onto a sphere model (1800 triangles), whose texture coordinates are modulated with several relative amplitudes *Ar*. With such complex texture images, distortions are less noticeable than with a simpler, geometrical texture (e.g., that of Figure 2).