TFAN stream description

Khaled Mamou, Titus Zaharia, Marius Preda, Francoise Preteux

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION
ORGANISATION INTERNATIONALE NORMALISATION

ISO/IEC JTC 1/SC 29/WG 11

CODING OF MOVING PICTURES AND AUDIO

ISO/IEC JTC 1/SC 29/WG 11/ **m15825**

Bursa, Korea, October 2008

Title: TFAN stream description

Authors Khaled Mamou, Titus Zaharia, Marius Preda, Francoise Prêteux

Source Institut TELECOM, TELECOM & Management SudParis, ARTEMIS Department

Status: Proposal

This contribution describes the Triangle Fan-based encode (TFAN) binary stream syntax.

1. TFAN stream

1.1 Overview

TFAN is a tool to compress an IndexFaceSet (IFS) node representing a triangular mesh by encoding the mesh geometry and connectivity. The mesh geometry information is defined by a subset of the following attributes:

- position coordinates,
- normals,
- texture coordinates,
- colours, and
- other attributes.

The mesh connectivity is composed of the subset of the following elements:

- position coordinates index,
- normal index,
- texture coordinates index, and
- others attributes index.

TFAN offers the functionality of preserving the vertices or the vertices and triangles orders. The data in a TFAN stream is structured into two components:

- The TFAN stream header: indicating general information about the IFS (number of coordinates, number of triangles …), and
- The TFAN data buffer containing the geometry and connectivity information.

The FAMC bitstream structure is illustrated in Figure XXX.1.

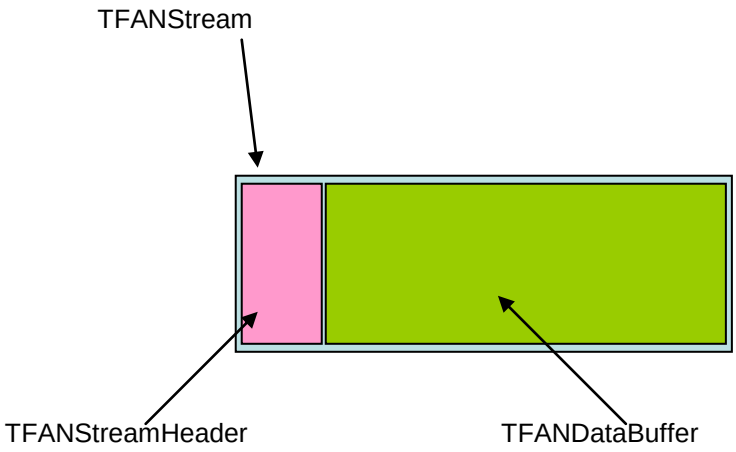

Figure XXX.1 — TFAN bitstream structure.

Figure XXX.2 illustrates the TFAN decoding process.

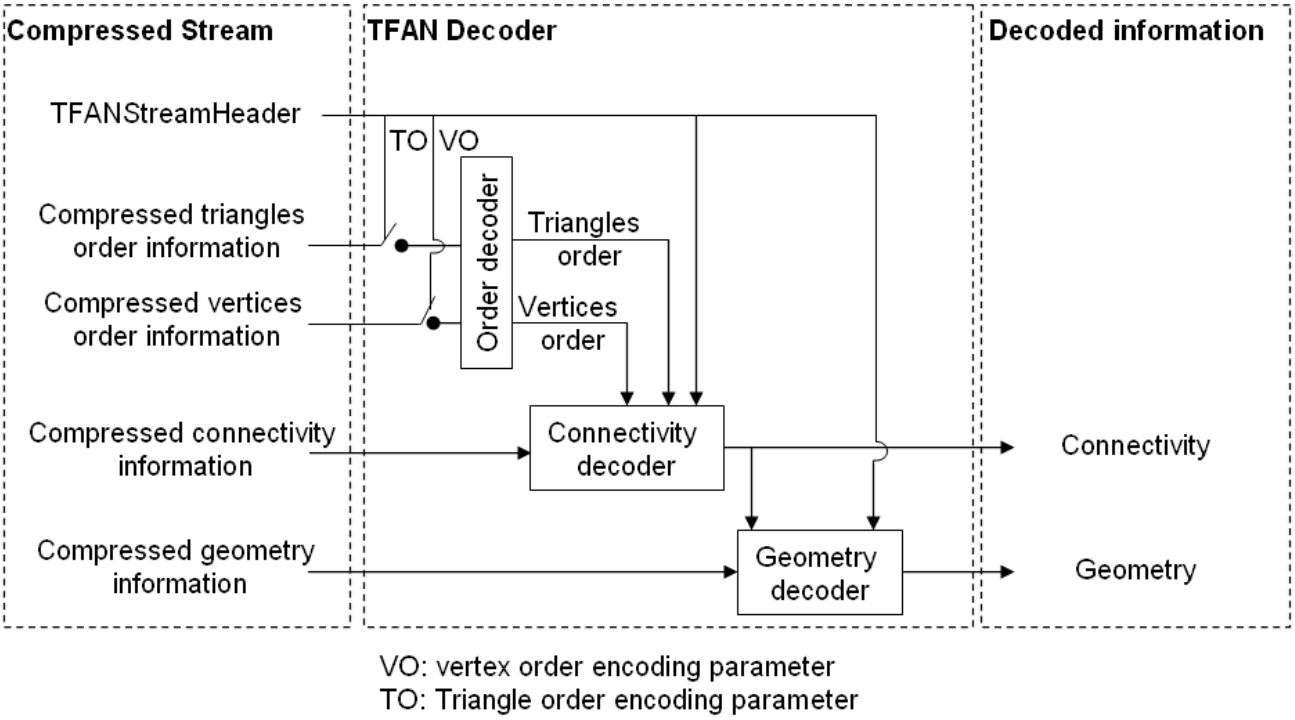

Figure XXX.2 — TFAN decoding process.

The following sections describe in detail the structure of the TFAN stream.

1.2 TFAN inclusion in the scene graph

TFAN stream is associated with an IndexedFaceSet by using the BitWrapper mechanism with value of field *type* equals to 3.

1.3 TFANStream class

1.3.1 Syntax

```
class TFANStream{
    TFANStreamHeader header;
    TFANDataBuffer data;
}
```

1.3.2 Semantics

TFANStreamHeader: contains the header buffer.

TFANDataBuffer: contains the data buffer.

1.4 TFANStreamtHeader class

1.4.1 Syntax

```
class  TFANStreamHeader {
    unsigned int (32) startCode;
    unsigned int (32) streamSize;
    float (32) creaseAngle;
    bit (1) ccw;
    bit (1) solid;
    bit (1) convex;
    bit (1) colorPerVertex;
    bit (1) normalPerVertex;
    bit (1) otherAttributesPerVertex;
    bit (1) isTringularMesh;
```

```
    bit (1) isOptimisedForParallelDeconding;
    unsigned int (32) numberOfCoord;
    unsigned int (32) numberOfTexCoord;
    unsigned int (32) numberOfNormal;
    unsigned int (32) numberOfColor;
    unsigned int (32) numberOfOtherAttributes;
    if (numberOfOtherAttributes >0) dimensionOfOtherAttributes;
    if (numberOfCoord>0) unsigned int (32) numberOfCoordIndex;
    if (numberOfTexCoord>0) unsigned int (32) numberOfTexCoordIndex;
    if (numberOfNormal>0) unsigned int (32) numberOfNormalIndex;
    if (numberOfColor>0) unsigned int (32) numberOfColorIndex;
    if (numberOfOtherAttributes >0) unsigned int (32) numberOfOtherAttributesIndex;
}
```

1.4.2 Semantics

startCode: a 32-bit unsigned integer equals to **TFANStreamStartCode**.

TFANStreamStartCode: a constant that indicates the beginning of a TFAN stream.

TFANStreamStartCode = 00 00 01 F0.

streamSize: a 32-bit unsigned integer describing the size in bytes of the current TFAN stream.

creaseAngle: a 32-bit float indicating the IFS *creaseAngle* parameter which controls the default normal generation process.

ccw: 1-bit describing the IFS *ccw* parameter which indicates whether the vertices are ordered in a counter-clockwise direction when the mesh is viewed from the outsides.

solid: 1-bit describing the IFS *solid* parameter which indicates whether the shape encloses a volume.

convex: 1-bit describing the IFS *solid* parameter which indicates whether all faces in the shape are convex (should be always 1 for triangular meshes).

colorPerVertex: 1-bit describing the IFS *colorPerVertex* parameter which indicates whether the colors are defined per vertex.

normalPerVertex: 1-bit describing the IFS *normalPerVertex* parameter which indicates whether the normals are defined per vertex.

otherAttributesPerVertex: 1-bit describing whether the other attributes are defined per vertex.

isTriangularMesh: 1-bit describing whether the mesh is triangular (should be always 1).

isOptimizedForParallelDecoding: 1-bit describing whether the stream is optimized for parallel decoding (should be always 0).

numberOfCoord: a 32-bit unsigned integer indicating the number of positions coordinates.

numberOfNormal: a 32-bit unsigned integer indicating the number of normals.

numberOfTexCoord: a 32-bit unsigned integer indicating the number of texture coordinates.

numberOfOtherAttributes: a 32-bit unsigned integer indicating the number of the other attributes.

dimensionOfOtherAttributes: a 32-bit unsigned integer indicating the dimension (i.e., number of attributes) of the other attributes.

numberOfCoordIndex: a 32-bit unsigned integer indicating the number of indices associated to the positions coordinates.

numberOfNormalIndex: a 32-bit unsigned integer indicating the number of the number of indices associated to the normals.

numberOfTexCoordIndex: a 32-bit unsigned integer indicating the number of the number of indices associated to the texture coordinates.

numberOfOtherAttributesIndex: a 32-bit unsigned integer indicating the number of the number of indices associated to the other attributes.

1.5 TFANDataBuffer class

1.5.1 Syntax

```
class  TFANDataBuffer {
    if (numberOfCoordIndex >0)
        TFANConnectivityDecoder(3, numberOfCoord, numberOfCoordIndex) coordIndex;
    if (numberOfTexCoordIndex >0)
        TFANConnectivityDecoder(3, numberOfTexCoord, numberOfTexCoordIndex) texCoordIndex;
    if (numberOfNormalIndex >0)
    {
        If (normalPerVertex == 1)
            TFANConnectivityDecoder(3, numberOfNormal, numberOfNormalIndex) normalIndex;
        else
            TFANConnectivityDecoder(1, numberOfNormal, numberOfNormalIndex) normalIndex;
    }
    if (numberOfColorIndex >0)
    {
        If (colorPerVertex == 1)
            TFANConnectivityDecoder(3, numberOfColor, numberOfColorIndex) colorIndex;
        else
            TFANConnectivityDecoder(1, numberOfColor, numberOfColorIndex) colorIndex;
    }
    if (numberOfOtherAttributesIndex >0)
    {
        If (otherAttributesPerVertex == 1)
            TFANConnectivityDecoder(3, numberOfOtherAttributes, numberOfOtherAttributesIndex)
otherAttributesIndex;
        else
            TFANConnectivityDecoder(1, numberOfOtherAttributes, numberOfOtherAttributesIndex)
otherAttributesIndex;
    }

    if (numberOfCoord >0) TFANGeometryDecoder(coordIndex, 0) coord;

    if (numberOfTexCoord >0)
    {
        If (numberOfTexCoordIndex >0) TFANGeometryDecoder(texCoordIndex, 3) texCoord;
        else TFANGeometryDecoder(coordIndex, 3) texCoord;
    }

    if (numberOfNormal >0)
    {
        If (normalPerVertex == 1)
        {
            If (numberOfNormalIndex >0) TFANGeometryDecoder(normalIndex, 3) normal;
            else TFANGeometryDecoder(coordIndex, 3) normal;
        }
        else
        {
            If (numberOfNormalIndex >0) TFANGeometryDecoder(normalIndex, 1) normal;
            else TFANGeometryDecoder(coordIndex, 1) normal;
        }
    }

    if (numberOfColor >0)
    {
        If (colorPerVertex == 1)
        {
```

```
            If (numberOfColorIndex >0) TFANGeometryDecoder(colorIndex, 3) color;
            else TFANGeometryDecoder(coordIndex, 3) color;
        }
        else
        {
            If (numberOfColorIndex >0)
                TFANGeometryDecoder(colorIndex, 1) color;
            else
                TFANGeometryDecoder(coordIndex, 1) color;
        }
    }

    if (numberOfOtherAttributes >0)
    {
        If (otherAttributesPerVertex == 1)
        {
            If (numberOfOtherAttributesIndex >0)
                TFANGeometryDecoder(otherAttributesIndex, 3) otherAttributes;
            else
                TFANGeometryDecoder(coordIndex, 3) otherAttributes;
        }
        else
        {
            If (numberOfOtherAttributesIndex >0)
                TFANGeometryDecoder(otherAttributesIndex, 1) otherAttributes;
            else
                TFANGeometryDecoder(coordIndex, 1) otherAttributes;
        }
    }
}
```

1.5.2 Semantics

coordIndex: bitstream describing the IFS *coordIndex* field.

texCoordIndex: bitstream describing the IFS tex*CoordIndex* field.

normalIndex: bitstream describing the IFS *normalIndex* field.

colorIndex: bitstream describing the IFS *colorIndex* field.

otherAttributesIndex: bitstream describing the indices associatd to the other attributes of the mesh.

coord: bitstream describing the IFS *coord* field.

texCoord: bitstream describing the IFS tex*Coord* field.

normal: bitstream describing the IFS *normal* field.

color: bitstream describing the IFS *color* field.

otherAttributes: bitstream describing the other attributes associated to the mesh.

1.6 TFANConnectivityDecoder class

1.6.1 Syntax

```
class  TFANConnectivityDecoder(dim, nV, nT) {
    if (dim == 3)
    {
        TFANConnectivityDecoderNTFans(nV) nTFans;
        TFANConnectivityDecoderDegrees degrees;
        TFANConnectivityDecoderCases cases;
        TFANConnectivityDecoderVerticesIndices verticesIndices;
```

```
        TFANConnectivityDecoderOps ops;
        If (vertexOrderPres == 1) {
            TFANConnectivityDecoderOrder  vo;
        }
        If (triangleOrderPres == 1) {
            TFANConnectivityDecoderOrder to;
        }

    }
}
```

1.6.2 Semantics

dim: parameter indicating the dimension the connectivity array to be decoded (should be either 1 or 3).

nV: parameter indicating the number of vertices for the connectivity array to be decoded.

nT: parameter indicating the number of triangles for the connectivity array to be decoded.

nTFans: bitstream of type TFANConnectivityDecoderNTFans describing for each vertex the number of triangle fans incidents to it.

degrees: bitstream of type TFANConnectivityDecoderDegrees describing for each triangle fan the number of its triangles.

cases: bitstream of type TFANConnectivityDecoderCases describing for each triangle fan its topological configuration.We refer to Annex A and Annex B for a description of the TFAN encoding process and a detailed specification of the 10 topological configurations (or cases) considered by the TFAN codec.

verticesIndices: bitstream of type TFANConnectivityDecoderVerticesIndices describing for each triangle fan the indices of the vertices composing it. We refer to Annex A and Annex B for a description of the TFAN encoding process.

ops: bitstream of type TFANConnectivityDecoderOps describing for each vertex of a triangle fan if it should be created by the decoder (i.e., new vertex) of an old one (i.e., already created). We refer to Annex A and Annex B for a description of the TFAN encoding process.

vo: bitstream of type TFANConnectivityDecoderOrder describing for each vertex its original order (i.e., position in the original IFS).

to: bitstream of type TFANConnectivityDecoderOrder describing for each triangle its original order (i.e., position in the original IFS).

1.7 TFANGeometryDecoder class

1.7.1 Syntax

```
class  TFANGeometryDecoder(dim, nV) {
    unsigned char (8) q;
    unsigned char (8) decodingMode;
    unsigned char (8) predMode;
    unsigned char (8) M;
    unsigned char (8) K;
    float (32) minValues[dim];
    float (32) maxValues[dim];

    if (predMode == 3)
        unsigned int (32) nbrPredModes;

    unsigned int(32) streamSize;

    if (predMode_ == 3) {
        for(int v = 0; v < nbrPredModes; v++) {
            vertexPredMode[v]= arithmetic_decoder.Decode();
```

```
        }
    }

    if (decodingMode == 0) {
        for(int v = 0; v <nV; v++) {
            for (int i = 0; i < dim; i++) {
                dV = arithmetic_decoder.Decode();
                if (dV <= 2*M) {
                    errTable[v*dim+i] = dV - M;
                }
                else {
                    if (dV == 2*M+1) {
                        errTable[v*dim+i] = - (arithmetic_decoder.ExpGolombDecode(K) + M + 1);
                    }
                    else {
                        errTable[v*dim+i] = arithmetic_decoder.ExpGolombDecode(K) + M + 1;
                    }
                }
            }
        }
    }
    else {
        for(int v = 0; v < nV; v++) {
            for (int i = 0; i < DIM; i++) {
                bit (1) rep = arithmetic_decoder.Decode();
                if (rep == 0 ){
                    errTable[v*DIM+i] = errTable[(v-1)*dim+i];
                }
                else {
                    dV = arithmetic_decoder.Decode();
                    if (dV <= 2*M) {
                        errTable[v*dim+i] = dV - M;
                    }
                    else {
                        if (dV == 2*M+1) {
                            errTable[v*dim+i] = - (arithmetic_decoder.ExpGolombDecode(K) + M + 1);
                        }
                        else {
                            errTable[v*dim+i] = arithmetic_decoder.ExpGolombDecode(K) + M + 1;
                        }
                    }
                }
            }
        }
    }
}
```

1.7.2 Semantics

q; a 8-bit unsigned integer indicating the number of quantization bits.

decodingMode; a 8-bit unsigned integer indicating the applied encodingMode. Table XXX.1 summarizes all possible configurations.

Table XXX.1 — Decoding mode: all possible configurations.

decodingMode value	Decoding mode
0	No prediction with respect to the previous value.
1	One bit encoded to indicate if the residual error to be decoded is the same as the last decoded one.
2-255	Reserved for ISO purposes

predictionMode; a 8-bit unsigned integer indicating the applied predictionMode. Table XXX.2 summarizes all possible configurations.

Table XXX.2 —Prediction mode: all possible configurations.

predictionMode value	Prediction mode
0	Barycenteric prediction: the predicted value associated to the current vertex is equal to the barycenter of its neighbors.
1	Delta prediction: the predicted value associated to the current vertex is equal to the value of the last decoded one.
2	Parallelogram prediction: the predicted value associated to the current vertex is obtained by applying the parallelogram prediction rule while considering all the decoded neighbors.
3	Adaptive prediction: one bit per vertex is encoded to indicate if mode 0 or mode 2 should be applied.
4	Parallelogram prediction: the predicted value associated to the current vertex is obtained by applying the parallelogram prediction rule while considering only the neighbors in a triangle fan.
5-255	Reserved for ISO purposes

M: a 8-bit unsigned integer describing the interval of values described by the arithmetic encoded representation. More specifically, if a decoded value ranges in the interval]-M, M[it is decoded only with the arithmetic decoder. Otherwise, an additional part is decoded with an Exponential Golomb code representation.

K: a 8-bit unsigned integer describing the order of the Exponential Golomb code representation.

minValues: an array of 32-bits float of dimension *dim* indicating the minimal values reached. These values are needed for the unquantization process.

maxValues: an array of 32-bits float of dimension *dim* indicating the minimal values reached. These values are needed for the unquantization process.

nbrPredModes: a 32-bit unsigned integer describing the number of prediction modes to be decoded.

streamSize: a 32-bit unsigned integer describing the size in bytes of the arithmetic encoded stream for the order stream.

nbrPredModes: a 32-bit unsigned integer describing the number of prediction modes to be decoded.

predModes: a bits array of dimension nbrPredModes describing for each vertex the prediction mode (0 or 2) to be applied.

errTable: a integer array of dimension (nV*dim) describing for each vertex the its quantized prediction residual.

dV: am integer value ranging in the interval [-M, M].

1.8 TFANConnectivityDecoderNTFans class

1.8.1 Syntax

class TFANConnectivityDecoderNTFans(nV) {

```
    unsigned char(8) maxNTFans;
    unsigned int(32) streamSize;
    for (int v = 0; v < nV; v++)
    {
        nbrTFans[v] = aithmetic_decoder.decode();
    }
}
```

1.8.2 Semantics

nV: parameter indicating the number of vertices for the connectivity array to be decoded.

maxNTFans: a 8-bit unsigned integer indicating the maximal number of triangle fans per vertex. This value is exploited to initialize the statistics model of the arithmetic encoding.

streamSize: a 32-bit unsigned integer describing the size in bytes of the arithmetic encoded stream for the numbers of triangles fans.

nbrTFans: an array of integer of dimension nV indicating for each vertex the number of triangles fan incident to it.

1.9 TFANConnectivityDecoderDegrees class

1.9.1 Syntax

```
class  TFANConnectivityDecoderDegrees{
    unsigned int (32) nDegrees;
    unsigned int (32) maxDegree;
    unsigned int (32) streamSize;

    for (int v = 0; v < nDegrees; v++) {
        degrees[v]=arithmetic_encoder.decode();
    }
}
```

1.9.2 Semantics

nDegrees: a 32-bit unsigned integer describing the number of degrees to be decoded.

maxNTFans: a 8-bit unsigned integer indicating the maximal number of triangle fans per vertex. This value is exploited to initialize the statistics model of the arithmetic decoder.

streamSize: a 32-bit unsigned integer describing the size in bytes of the arithmetic encoded stream for the degrees.

degrees: an array of integer of dimension nDegrees indicating the dimension of each triangle fan.

1.10 TFANConnectivityDecoderCases class

1.10.1 Syntax

```
class  TFANConnectivityDecoderCases {
    unsigned int (32) streamSize;
    for (int v = 0; v < nDegrees; v++) {
        cases[v]=arithmetic_decoder.Decode();
    }
}
```

1.10.2 Semantics

nDegrees: a 32-bit unsigned integer describing the number of degrees to be decoded and obtained when decoding the last TFANConnectivityDecoderDegrees stream.

streamSize: a 32-bit unsigned integer describing the size in bytes of the arithmetic encoder stream for the TFAN topological configurations.

cases: an array of integer of dimension nDegrees indicating for each triangle fan its topological configuration.

1.11 TFANConnectivityDecoderVerticesIndices class

1.11.1 Syntax

```
class  TFANConnectivityDecoderVerticesIndices {
    unsigned int (32) nBitsVerticesIndices;
    unsigned int (32) streamSize;

    for (int v = 0; v < nBitsVerticesIndices; v++) {
        binaryStreamVerticesIndices[v] = arithmetic_decoder.Decode();
    }
}
```

1.11.2 Semantics

nBitsVerticesIndices: a 32-bit unsigned integer describing the number of bits of the binary version of the vertices indices stream.

streamSize: a 32-bit unsigned integer describing the size in bytes of the arithmetic encoded stream for the vertices indices.

binaryStreamVerticesIndices: an array of bits of dimension nBitsVerticesIndices. It should be interpreted as stream of type TFANBinaryStreamVerticesIndices.

1.12 TFANConnectivityDecoderOps class

1.12.1 Syntax

```
class  TFANConnectivityDecoderOps {
    unsigned int (32) nOps;
    if (nOps > 0) {
        unsigned int (32) streamSize;
        for (int v = 0; v < nOps; v++) {
            ops[v]=arithmetic_decoder.Decode();
        }
    }
}
```

1.12.2 Semantics

nOps: a 32-bit unsigned integer describing the number of operation values to be decoded.

streamSize: a 32-bit unsigned integer describing the size in bytes of the arithmetic encoded stream for the operation stream.

ops: an array of bits of dimension nOps indicating for each vertex of a triangle fan if it should be created by the decoder or not.

1.13 TFANConnectivityDecoderOrder class

1.13.1 Syntax

```
class  TFANConnectivityDecoderOrder(N) {
```

```
    unsigned char(8) K;
    unsigned int (32) streamSize;

    map[0] = aithmetic_decoder.ReadBinary(nBitsFLOrder);

    int d = 0;
    for (int v = 1; v < N; v++) {
        bit (1) rep = arithmetic_decoder.Decode();
        if (rep == 0) {
            unsigned int (32) d = arithmetic_decoder.ExpGolombDecode(K) + 1;
            bit (1) sign = arithmetic_decoder.Decode();
            if (sign == 1) d = -d;
            map[v] = vertexMapR[v-1] + d;
        }
        else {
            map[v]=  arithmetic_decoder.ReadBinary(nBitsFLOrder);
        }
    }
}
```

1.13.2 Semantics

N; a parameter indicating the number of the vertices or triangles to be reordered.

K: a 8-bit unsigned integer describing the order of the Exponential Golomb code representation.

nBitsFLOrder; value indicating the number of bits needed to encode the parameter N.

streamSize: a 32-bit unsigned integer describing the size in bytes of the arithmetic encoded stream for the order stream.

rep: a 1-bit value indicating if the FL binary representation should be used or the Exponential Golomb one.

sign: a 1-bit value indicating if the sign of the Exponential Golomb decoded value.

map: an array of integers of dimension N indicating for each vertex or triangle its original position.

1.14 TFANBinaryStreamVerticesIndices class

1.14.1 Syntax

```
class  TFANBinaryStreamVerticesIndices {
    unsigned char(8) nBitsFL;
    unsigned int minValue;
    for (int v = 0; v < nVerticesIndices; v++) {
        verticesIndices[v]=stream.ReadBinary(nbits) + minV;
    }
}
```

1.14.2 Semantics

nBitsFL: a 8-bit unsigned integer describing the number of bits used for the fixed length binarization of the vertices indices.

minValue: a 32-bit unsigned integer describing the minimal shift value that should be subtracted from all the vertices indices.

nVerticesIndices: a parameter describing the number of vertices indices. It is derived as follows. If nBitsFL= 0 then nVerticesIndices = 0. Otherwise, nVerticesIndices = (nBitsVerticesIndices -40)/ nBitsFL.

verticesIndices: an array of integer of dimension nVerticesIndices describing the vertices indices.

ANNEX A: TFAN connectivity encoding process

The TFAN technique encodes the mesh connectivity by exploiting a deterministic traversal of mesh vertices combined with a decomposition of the mesh triangles into triangle fans. The TFAN encoder describes the connectivity of any triangular mesh, by encoding for each triangle fan the following information:

a) the degree: number of triangles composing the triangle fan,

b) a binary sequence indicating for each vertex the set of its already visited neighboring vertices,

c) a sequence of relative indices with respect to a local list *L*, used to identify the indices of the already visited vertices.

In order to efficiently compress such a representation, the TFAN approach distinguishes 9 topological configurations which are encoded in a specific and more compact manner. The generic case (all configurations not described by the 9 special configurations), called configuration 10, is handled by directly encoding the information a), b) and c).

1) Definitions

A Triangle Fan (TF) of degree *d* is an ordered set of *d* triangles $(t_j)_{i\in\{0,...,d-1\}}$ defined by an ordered sequence of $d+2$ vertices $(v_0, v_1, ..., v_{d+1})$, such that :

$$\forall\ j \in \{0,1,...,d-1\},\quad t_j = \{v_0, v_{j+1}, v_{j+2}\}$$

Figure 1 illustrates a triangle fan of degree 4.

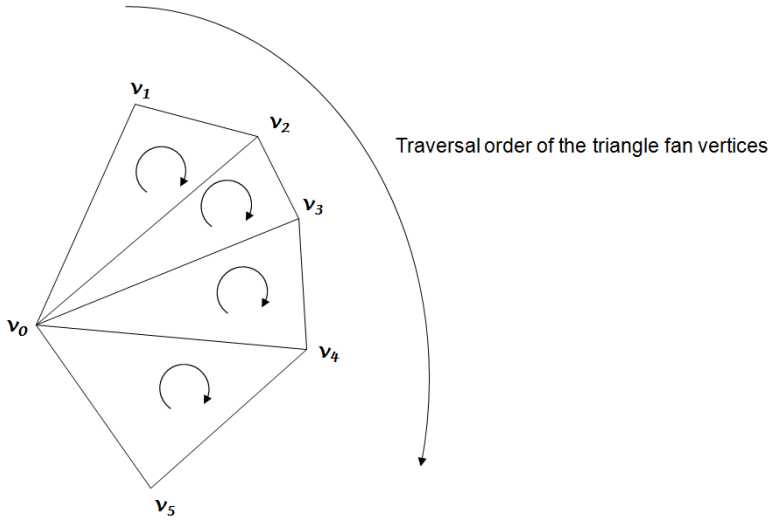

Figure 1: A triangle fan of degree 4 described by the vertices $(v_0, v_1, v_2, v_3, v_4, v_5)$.

By definition, a triangle fan satisfies the following properties:

- Two successive triangles of a triangle fan are adjacent (i.e., share a common edge).
- All the triangles of a triangle fan have the some orientation.
- All the triangles of a triangle fan share a common vertex v_0, called central vertex or centre of the triangle fan.

Let us note that the common orientation of the triangles implies a unique order of traversal of the triangle fan vertices. The triangle fan TF is then completely determined by giving the ordered sequence of vertices $(v_0, v_1, ..., v_{d+1})$ enumerated starting by the central vertex.

2) Construction of the TFAN representation

The global schema of the TFAN approach can be described as follows. At the initialization step, all the mesh vertices and triangles are considered as non-visited (or non-traversed). The TFAN encoder exploits a FIFO structure F in order to store the mesh vertices. The first non-visited vertex is pushed in F.

At each iteration, a vertex is extracted from F. Let v_j be the extracted vertex at iteration j. The vertex v_j is then marked as visited, its new traversal order is stored in $O(v_j)$ and all the non-visited triangles incident to v_j are decomposed into a set of triangles fans. Let us note that different decomposition strategies are possible. In the current version of the TFAN encoder we have adopted an iterative approach described here-below.

At each iteration, a triangle fan $TF_n(j)$ is created by starting from the triangle t_0 with the minimal number of non-visited neighbors (i.e., triangles sharing an edge with t_0) having the same orientation as t_0. The triangle t_0 is added to the triangle fan $TF_n(j)$ and marked as visited. If t_0 has non-visited neighbors with the same orientation, then a neighbor t_1 is randomly chosen, added to $TF_n(j)$ and marked as visited. The same operation is then applied to t_1. This process is iterated until there are no more non-visited neighbors. If there still non-visited triangles incident to v_j, then a new triangle fan is created by starting again from the triangle with the minimal number of non-visited neighbors having the same orientation.

Let:
- $(TF_n(j))_{n\in\{1,\ldots,N(j)\}}$ be the set of the $N(j)$ triangle fans associated to the vertex v_j,
- $\left\{v_j, w_j^n(1), w_j^n(2), \ldots w_j^n(d(j,n)+1)\right\}$ the ordered vertices of the triangle fan $TF_n(j)$ and $d(j,n)$ its degree.

We define $L(j)$ as the list of vertices sharing with v_j at least one visited triangles and having a traversal order higher than v_j. The list $L(j)$ is sorted (in the increasing order) by considering the following order relationship:

$$\forall \left\{w_1, w_2\right\} \in L(j), \quad w_1 < w_2 \Leftrightarrow O(w_1) < O(w_2).$$

At the exception of vertex v_j, all the vertices of the triangle fan $TF_n(j)$ are treated while considering the order defined by the triangle fan $TF_n(j)$. Here, a binary value $s_j^n(k)$ is associated to each vertex $w_j^n(k)$, in order to indicate if it has been visited or not. All the tags $s_j^n(k)$ associated to the same triangle fan are stored in a FIFO $S(j,n)$.

If the vertex $w_j^n(k)$ was not visited (*i.e.*, $s_j^n(k)=0$), the it is marked as visited, added to the FIFO F and inserted at the end of the list $L(j)$. Its traversal order is stored in $O(w_j^n(k))$.

If the vertex $w_j^n(k)$ was visited (*i.e.*, $s_j^n(k)=1$), we distinguish two cases. If $w_j^n(k)$ belongs to $L(j)$ then its relative index $\mu_j^n(k)$ in the list $L(j)$ is stored in the FIFO $I(j,n)$. Otherwise, the negative value $\mu_j^n(k) = O(v_j) - O(w_j^n(k))$ is stored in $I(j,n)$. Let us recall that by definition of $L(j)$, all its vertices have a traversal order higher to v_j. Therefore, the quantity $\mu_j^n(k)$ is always negative. This property will be exploited at the decoder side in order to determine if the index of the vertex $w_j^n(k)$ is located in the list $L(j)$ or not.

When all the vertices of the triangles fan $TF_n(j)$ are treated, its triangles are marked as visited and the same process is iterated until all the mesh vertices will be visited. If the FIFO F becomes empty and there still non-visited vertices, then the first non-visited vertex is added to F and the process restarted.

In conclusion, the information necessary to completely represent the mesh connectivity is composed of the following elements:

- The number V of mesh vertices,
- The sequence $(N(j))_{j\in\{1,...,V\}}$ indicating for each vertex v_j the number of its triangles fans,
- The sequence $(d(j,n))_{j\in\{1,...,V\}}^{n\in\{1,...,N(j)\}}$ describing the degrees of all the triangles fans incident to the vertex v_j,
- The FIFO $(S(j,n))_{j\in\{1,...,V\}}^{n\in\{1,...,N(j)\}}$ describing for each triangle fan the set of its visited vertices,
- The sequence of the indices FIFOs $(I(j,n))_{j\in\{1,...,V\}}^{n\in\{1,...,N(j)\}}$ which describes the indices of the visited vertices.

Figure 2 presents the pseudo-code of the TFAN encoding algorithm. Figure 3, illustrates an encoding example.

Algorithm : TFAN encoder

Objective : Encode the connectivity of a triangular mesh

Input : List of triangles

Output : V , $(N(j))_{j\in\{1,...,V\}}$, $(d(j,n))_{j\in\{1,...,V\}}^{n\in\{1,...,N(j)\}}$, $(S(j,n))_{j\in\{1,...,V\}}^{n\in\{1,...,N(j)\}}$, $(I(j,n))_{j\in\{1,...,V\}}^{n\in\{1,...,N(j)\}}$

{

$F \leftarrow \{\}$

$\text{ordreVisite} \leftarrow 1$

$j \leftarrow 0$

For $h \in \{1,...,V\}$

If (v_h non visited) {

F .PushBack(v_h)

MarkAsVisited (v_h)

$O(v_h) \leftarrow \text{ordreVisite++}$

while(F non empty){

$j++$

$v_j \leftarrow F$ .PopFirst()

$L(j) \leftarrow F$ .ComputeOrderedListOfNonVisitedNeighbors()

$(TF_n(j))_{n\in\{1,...,N(j)\}} \leftarrow$ TriangleFanDecomposition(j)

For $n \in \{1,...,N(j)\}$ {

For $k \in \{1,...,d(j,n)\}$ {

If (the vertex $w_j^n(k)$ is non visited)} {

$s_j^n(k) \leftarrow 0$

Mark as visited($w_j^n(k)$)

$L(j)$ .PushBack($w_j^n(k)$)

$O(w_j^n(k)) \leftarrow ordreVisite++$

F .PushBack($w_j^n(k)$)

}

else {

$s_j^n(k) \leftarrow 1$

If ($w_j^n(k) \in L(j)$){

$\mu_j^n(k) \leftarrow L(j)$ .FindIndex($w_j^n(k)$)

$I(j,n)$ .PushBack($\mu_j^n(k)$)

}

else {

$I(j,n)$ .PushBack($\mu_j^n(k)$)

$I(j,n)$ .PushBack($O(v_j) - O(w_j^n(k))$)

}

}

}

MarkAllTheTrianglesAsVisited ($TF_n(j)$)

}

}

}

}

}

Figure 2 : Pseudo-code of the TFAN encoding algorithm.

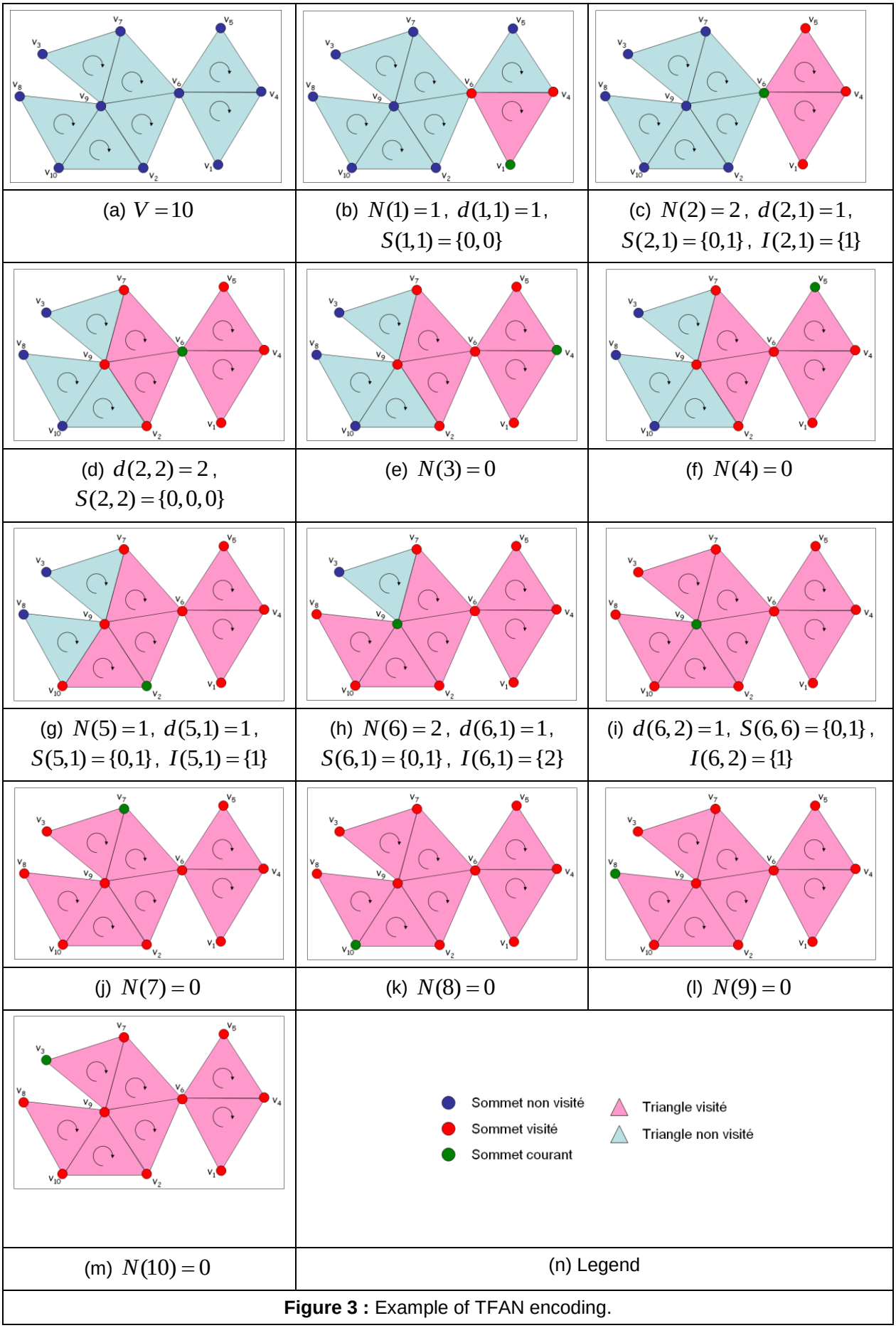

Figure 3 : Example of TFAN encoding.

3) Compression of the TFAN representation

In order to compress the TFAN representation, it is necessary to compactly encode for each triangle fan $TF_n(j)$ its $d(j,n)$ and the two lists $S(j,n)$ and $I(j,n)$. Let us note that we can directly encode these

information by applying an arithmetic encoder. However, in order to obtain more efficient compression performances, we propose to distinguish 9 frequent topological configurations, which we encode more compactly. In order to handle all the other cases, a tenth configuration is introduced. It consists in directly encoding the information $d(j,n)$, $S(j,n)$ and $I(j,n)$.

Let $C(j,n)$ be the configuration associated to the triangle fan $TF_n(j)$. Table 1 describes the ten configurations retained by the TFAN approach.

The determined configurations are finally encoded by applying an arithmetic encoder.

Configuration	Initial information	Encoded information
$C(j,n)=1$	$d(j,n)$, $S(j,n)=\underbrace{\{1,0,0,...,0,0,1\}}_{1+d(j,n)}$ and $I(j,n)=\{1,2\}$	$C(j,n)=1$ and $d(j,n)$
$C(j,n)=2$	$d(j,n)$, $S(j,n)=\underbrace{\{1,X,X,...,X,X,1\}}_{1+d(j,n)}$ and $I(j,n)=\{1,X,X,...,X,X,2\}$	$C(j,n)=2$, $d(j,n)$, $S'(j,n)=\underbrace{\{X,X,...,X,X\}}_{d(j,n)-1}$ and $I(j,n)=\{X,X,...,X,X\}$
$C(j,n)=3$	$d(j,n)$, $S(j,n)=\underbrace{\{0,0,...,0,0,1\}}_{1+d(j,n)}$ and $I(j,n)=\{1\}$	$C(j,n)=3$ and $d(j,n)$
$C(j,n)=4$	$d(j,n)$, $S(j,n)=\underbrace{\{0,0,...,0,0,1\}}_{1+d(j,n)}$ and $I(j,n)=\{2\}$	$C(j,n)=4$ and $d(j,n)$
$C(j,n)=5$	$d(j,n)$, $S(j,n)=\underbrace{\{1,0,...,0,0,0\}}_{1+d(j,n)}$ and $I(j,n)=\{1\}$	$C(j,n)=5$ and $d(j,n)$
$C(j,n)=6$	$d(j,n)$, $S(j,n)=\underbrace{\{1,0,...,0,0,0\}}_{1+d(j,n)}$ and $I(j,n)=\{2\}$	$C(j,n)=6$ and $d(j,n)$
$C(j,n)=7$	$d(j,n)$, $S(j,n)=\underbrace{\{0,0,...,0,0,0\}}_{1+d(j,n)}$ and $I(j,n)=\{\}$	$C(j,n)=7$ and $d(j,n)$
$C(j,n)=8$	$d(j,n)$, $S(j,n)=\underbrace{\{1,0,0,...,0,0,1\}}_{1+d(j,n)}$ and $I(j,n)=\{2,1\}$	$C(j,n)=8$ and $d(j,n)$
$C(j,n)=9$	$d(j,n)$, $S(j,n)=\underbrace{\{1,X,X,...,X,X,1\}}_{1+d(j,n)}$ and $I(j,n)=\{2,X,X,...,X,X,1\}$	$C(j,n)=9$, $d(j,n)$, $S'(j,n)=\underbrace{\{X,X,...,X,X\}}_{d(j,n)-1}$ and $I(j,n)=\{X,X,...,X,X\}$
$C(j,n)=10$	$d(j,n)$, $S(j,n)=\underbrace{\{X,X,...,X,X\}}_{1+d(j,n)}$ and $I(j,n)=\{X,X,...,X,X\}$	$C(j,n)=10$, $d(j,n)$, $S(j,n)=\underbrace{\{X,X,...,X,X\}}_{1+d(j,n)}$ and $I(j,n)=\{X,X,...,X,X\}$

Tableau 1: The 10 configurations considered by the TFAN approach (*X* represents an arbitrary value).

ANNEX B: TFAN connectivity decoding process

The TFAN decoded reconstructs the mesh connectivity by successively decoding the set of the triangle fans. Let us note that the mesh vertices are traversed in the order defined by the encoder. Figure 4 describes the pseudo-code of the TFAN decoding algorithm.

At the iteration j, the non-decoded triangle fans $(TF_n(j))_{n\in\{1,...,N(j)\}}$ incidents to the vertex j are reconstructed as follows. First, the ordered list $L(j)$ of the neighboring vertices of v_j with a higher index is computed and the number of triangles fans $N(j)$ is decoded. The triangle fans are then generated in the order of their encoding. In order to reconstruct the triangle fan $TF_n(j)$, the decoder reads from the binary stream the following elements :

- The degree $d(j,n)$ of the triangle fan,
- The FIFO $S(j,n)$ indicating the set of the visited vertices, and
- The FIFO of indices $I(j,n)$.

The triangle fan $TF_n(j)$ is initialized with an sequence containing only the vertex j. The other vertices are added successively in the order of their encoding. Let $w_j^n(k)$ be the vertex number k of the current triangle fan. In order to determine if it is a new vertex or an already decoded one, the TFAN decoder extracts from the FIFO $S(j,n)$ one bit $s_j^n(k)$ associated to the vertex $w_j^n(k)$.

If $s_j^n(k)=0$, then $w_j^n(k)$ is a new vertex. It is then created by assigning it an index equal to its traversal order. This index is then added to the triangle fan $TF_n(j)$ and at the end the list $L(j)$. The traversal order is finally incremented by 1.

If $s_j^n(k)=1$, the vertex $w_j^n(k)$ is identified as already decoded. In order to determine its index, the decoder extracts the first element $\mu_j^n(k)$ of the FIFO $I(j,n)$. If $\mu_j^n(k)>0$, the index of the vertex $w_j^n(k)$ is obtained by reading the element $\mu_j^n(k)$ of $L(j)$. Otherwise (*i.e.*, $\mu_j^n(k)<0$), the index of $w_j^n(k)$ is $j-\mu_j^n(k)$. In both cases the vertex index is added to $TF_n(j)$.

Algorithm : TFAN Decoder

Objective: Decode the connectivity of a triangular mesh

Input: V, $(N(j))_{j\in\{1,...,V\}}$, $(d(j,n))_{j\in\{1,...,V\}}^{n\in\{1,...,N(j)\}}$, $(S(j,n))_{j\in\{1,...,V\}}^{n\in\{1,...,N(j)\}}$, $(I(j,n))_{j\in\{1,...,V\}}^{n\in\{1,...,N(j)\}}$

Output: List of triangles

```
{
F ← {}
ordreVisite ← 1

For j ∈ {1,...,V}
    If ( j == ordreVisite ) {
        ordreVisite++
    }
    L(j) ← F . ComputeOrderedListOfNonVisitedNeighbors()
    Lire( N(j) )
    For n ∈ {1,..., N(j)}
        Lire( d(j,n) )
```

```
        Lire( S(j,n) )
        Lire( I(j,n) )
        TF_n(j) ← {j}
        For k ∈ {1,...,1+d(j,n)}
            s_j^n(k) ← S(j,n).PopFirst()
            If ( s_j^n(k) == 0 ) {
                TF_n( j ) .PushBack( ordreVisite )
                L( j,n) .PushBack( ordreVisite )
                ordreVisite++
            }
            Else
            {
                μ_j^n(k) ← I( j,n).PopFirst()
                If ( μ_j^n(k) > 0 )
                {
                    TF_n( j ) .PushBack( L(μ_j^n(k)) )
                }
                else
                {
                    TF_n( j ) .PushBack( j − μ_j^n(k) )
                }
            }
        }
}
```

Figure 4: Pseudo-code of the TFAN decoder.