Title:
Investigation on Reducing Geometry Files Size Through Floating Points Indexing

Authors:
Benjamin Vaissier, benjamin.vaissier@ensam.eu, Arts et Métiers, LISPEN EA 7515, HeSam, Aix-en-Provence, France
Jean-Philippe Pernot, jean-philippe.pernot@ensam.eu, Arts et Métiers, LISPEN EA 7515, HeSam, Aix-en-Provence, France
Laurent Chougrani, l.chougrani@poly-shape.com, Poly-Shape, 235 rue des Canesteu, Salon-de-Provence, France
Philippe Véron, philippe.veron@ensam.eu, Arts et Métiers, LISPEN EA 7515, HeSam, Aix-en-Provence, France

Keywords:
File size reduction, lossless compression, floating-point geometry, additive manufacturing

DOI: 10.14733/cadconfP.2019.147-152

Introduction:

With the emergence and the increasing adoption of Industry 4.0, industrial processes are generating more and more data [2]. For its treatment and analysis, many steps are often required along with the use of various specialized software. The manipulation and exchange of heavy data encoding files are thus frequent. For example, in the context of the manufacturing industry, 3D geometry files are widely used to produce parts with intricate geometries. Moreover, though traditional manufacturing processes are also manipulating geometry files, with the emergence of 3D Printing or Additive Manufacturing (AM) technologies, the design, manipulation and fabrication of complex shapes are not restricted to professionals anymore. The development of Additive Manufacturing technologies is also accompanied by the apparition of online geometry exchange platforms (such as Thingiverse, GrabCAD, PinShape...) [3].

In this context full of cooperative data exchanges, frequent transfers between specialized software and remote design and manufacturing, fluidity is the key. It is thus important to reduce the size of data encoding files in order to ease their manipulation. In particular, in the case of 3D-geometry-based processes, triangulated meshes are often used. In such files, the 3D points are localized in space thanks to three coordinates. In order to reduce the size of geometry files, this paper investigates the use of an indexing mechanism to encode these floating-point coordinates.

Overall Framework:

The indexing mechanism is a simple two-step process: first, all the objects considered for indexation are gathered without redundancy into a unique collection associating an index to each object, through an *association* step. This collection of objects is then encoded generally at the beginning of the resulting file, through an *aggregation* step. Then, in both cases, all the places where an indexed object would have been described with a traditional file encoding, the index corresponding to the object in question is

specified instead. In this paper, the objects considered for indexation are the floating-point coordinates used to describe the 3D points of a geometry file.

Within the aggregation step, the identification of the index associated with a specific object can be explicit (the index associated with each object is specified next to the description of the object) or implicit through the order of apparition of the objects within the collection (in the case of an m long collection, the index 0 is associated with the first object described and the index $m-1$ is associated with the last object, without any index being mentioned). The implicit indexing is more efficient in term of resulting file size because the indices are not mentioned within the objects collection description. However, in an explicitly indexing, it is easier when reading an index (with a human eye) to identify the object to which it corresponds. In the rest of this paper, only the implicit indexing mechanism is considered, privileging file size reduction over human eye readability.

The indexing mechanism is already used for the size reduction of some geometry files format such as the .obj, the .ply or the .vrml files [4]. However, to the authors knowledge, in all the file formats adopting an indexing mechanism, it is used to describe the geometry at the level of the triangle vertices: all the 3D points are first listed through the description of their three Cartesian coordinates, and then each triangle is encoded as a set of three point indices. In this paper, the indexing mechanism is employed but at a smaller level to avoid the redundancies in the floating-point coordinates. Figure 1 illustrates the two association and aggregation steps of the indexing framework, in the case of floating-points indexing. The HOCA and BTA approaches used in this illustrative example, and detailed further in the paper, are the ones implemented in the proposed framework.

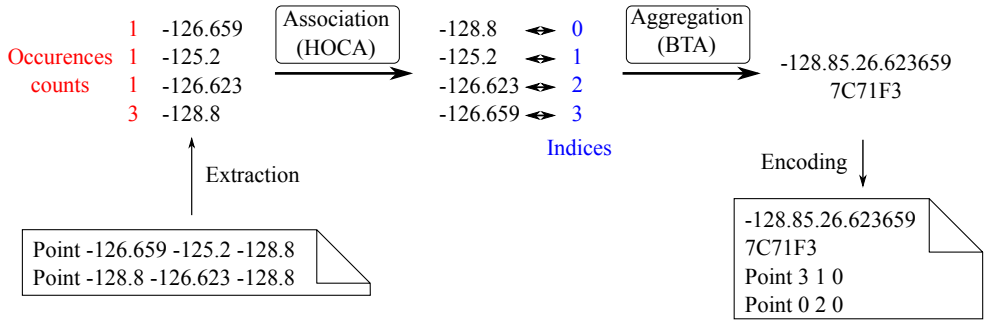

Fig. 1: The proposed two step indexing framework for floating-points encoding

Association:

The association of an index with a specific floating point can be realized in several ways. The classical approach is to index the floating points according to their order of apparition: the first floating-point to be processed will be associated with the index 0, the next one with the index 1, etc... This approach will be designated as the Apparition-Order-Association (AOA) approach. Another approach to consider is to index floating-points according to their values: the smallest (respectively the highest) floating-point is associated with the smallest index, whereas the highest (respectively the smallest) floating-point is associated with the highest index. These approaches are entitled Ascending-Value-Association or AVA approach (respectively Descending-Values-Association or DVA approach). Though these methods are easy to understand and implement, they are not the most efficient. Indeed, if a floating point must be encoded many times within the resulting file, it is beneficial to assign to it a small index (with a small number of digits).

Another approach is thus to index a floating-point according to the number of times it must appear

within the resulting file: the floating-points appearing most often will be associated with the smallest index, namely 0, whereas the floating-points appearing the least often will be associated with the biggest index (containing the highest number of digits). This approach has been denominated as the Occurrences-Count-Association (OCA) approach.

Finally, the indexing mechanism chosen in the rest of this article is a hybrid one, coupling the AVA and the OCA approaches. Indeed, the floating-points are first sorted according to their number of occurrences in the encoding file, like with the OCA, and a temporary index is thus associated with each floating-point, according to its position. Then, the floating-points whose indices have the same number of digits are gathered together: the ones with temporary indices between 0 and 9 forms the first group, the ones with temporary indices between 10 and 99 forms the second group, etc... Within each group, the floating-points are then sorted according to their value, before the groups are concatenated together again, attributing to each floating-point its final index. This hybrid mechanism, denominated Hybrid-Occurrences-Count-Association (or HOCA), is identical to the OCA in term of size reduction performances, but has the advantage to attribute close indices to similar floating-points. This property will be exploited by the aggregation mechanism implemented in the proposed framework.

Aggregation:

The objective of the aggregation step is to describe the list of all the indexed floating points. Two approaches can be considered. The simplest one consists of concatenating all the floating points one after the other. This aggregation mechanism is called End-to-End-Aggregation (EEA). Although simple to create and read with a human eye, this aggregation mechanism is far from ideal. Indeed, it doesn't benefit from the repeated digits of the various floating points. For example, if one tries to aggregate the following sorted list of distinct floating-points $L_w = \{$ 126.659, 125.2, 126.623, 128.8 $\}$ with the EEA, it comes the following describing sequence: "126.659 125.2 126.623 128.8". Though the four floating-points are starting by the same two digits, these digits are repeated for each floating-point description

In order to benefit from this digits redundancy, another aggregation approach is proposed, using a binary tree. The concept is to create a rooted tree in which with each node is associated a digit, and where each path from the root to a leaf encode a single floating-point (by concatenating all the encountered digits). Then, a reversible encoding of the rooted tree enables to describe all the encapsulated floating-points. This so-called Binary-Tree-Aggregation (or BTA) mechanism is illustrated on figure 2. It is the aggregation mechanism implemented in the proposed indexing framework. The reversible encoding used to describe the binary tree is called Zaks'sequence [1]. In the proposed framework, this binary sequence is converted into hexadecimal format in order to further reduce the resulting number of encoding characters, as illustrated in figure 2.

Experimentations and Results:

As a first step, the proposed encoding framework is applied to several geometry files, and the resulting sizes are compared to the initial ones. The encoding of 8 geometries through 4 types of file formats are considered: STL, OBJ, VRML and PLY files. Table 1 details the sizes of all the initial files, and compares them to the ones of the resulting files, after applying the proposed encoding framework. It also specifies the size reduction ratio between the two files. For information, the number of vertices and triangles of each encoded 3D geometry is mentioned. Analyzing these results, file reductions ranging from -42.4% to +18.1% can be observed. For one model (namely the Stanford Bunny), and two file formats (VRML and PLY files), it appears that the size of the generated files has increased. This can be explained by the counter-productivity of the proposed framework in some extreme cases. Indeed, if each floating-point considered for indexation is present only once in the initial file, the indexation mechanism is generating additional characters (through the indexes digits) without exploiting any redundancy. Furthermore, if the floating points are not presenting enough similarities between each other, the description of the binary

tree can require more characters than a simple EEA description.

To go further, the impact of the geometry orientation on the resulting file size is studied: indeed, because the orientation modifies the geometry points coordinates, it can alter or enhance the encoding efficiency. For example, if a 3D rotation can transform a set of points aligned on a randomly positioned right line into a set of vertically aligned points, the resulting file size will be considerably reduced (the X and Y coordinates being shared by all the encoded points). To investigate this influence, several geometries are oriented according to various sets of Euler angles, and are then encoded through the proposed framework. The considered orientations are determined through a dichotomic algorithm so that they are uniformly distributed over the 3D orientations space. For each 3D geometry, the resulting file size reduction distributions are displayed in figure 3 in the form of histograms: the value associated to a specific interval is an estimation of the proportion of orientations generating such a file size reduction. Here, only STL files are considered. The proportion of orientations is computed by applying the proposed framework on each orientation and by counting the ones for which the file size reduction lies within the considered range. It can also be viewed as the probability of observing a file size reduction within the considered interval if taking a random orientation for the geometry to encode. From these histograms, it can be observed that selecting a judicious orientation can considerably enhance the file size reduction: from -25.0% to -50.0% in the case of the Heat Sink geometry, for example.

Conclusion and Perspectives:

In this paper, the interest of indexing the floating-point coordinates of geometry encoding files has been investigated. A framework coupling an association and an aggregation mechanism to encode floating-points within a file has first been proposed. Its execution on various geometry files has demonstrated size reduction reaching up to -42.4%. The improvement of the file size reduction through the choice of a judicious orientation for the encoded geometry has revealed to generate up to -50.0% lighter files in

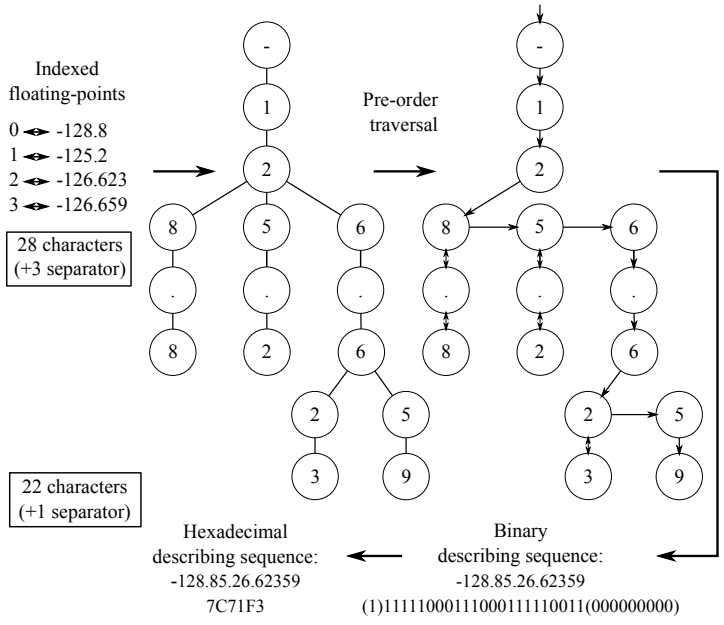

Fig. 2: Binary Tree Aggregation (BTA) mechanism demonstrating a reduction from 28 to 22 characters.

	Number of vertices	Number of triangles	Initial file size	Resulting file size	Resulting size reduction	Initial file size	Resulting file size	Resulting size reduction
			STL files			OBJ files		
Heat sink	88	172	32.8 ko	18.8 ko	-42.4%	10.5 ko	7.2 ko	-31.9%
Arches[1]	528	1 040	205 ko	129 ko	-37.1%	69 ko	51 ko	-26.1%
Bird[1]	3 241	6 486	1 274 ko	1 106 ko	-13.2%	461 ko	429 ko	-6.9%
Stanford Bunny[2]	35 947	69 451	13.8 Mo	13.1 Mo	-4.9%	5.4 Mo	5.3 Mo	-0.2%
Armadillo[2]	172 974	345 944	78 016 ko	59 265 ko	-24.0%	29 847 ko	26 527 ko	-11.1%
Dragon[2]	437 645	871 414	146 728 ko	122 164 ko	-16.7%	61 615 ko	48 479 ko	-21.3%
Happy bouddha[2]	543 652	1 087 716	179 921 ko	149 056 ko	-17.2%	75 764 ko	59 574 ko	-21.4%
Turbine blade[3]	882 954	1 765 388	374.2 Mo	329.2 Mo	-12.0%	154.6 Mo	131.7 Mo	-14.8%
			VRML files			PLY files		
Heat sink	88	172	5.9 ko	4.4 ko	-26.2%	4.5 ko	3.1 ko	-29.9%
Arches[1]	528	1 040	38 ko	31 ko	-18.4%	30 ko	23 ko	-23.3%
Bird[1]	3 241	6 486	250 ko	246 ko	-1.6%	195 ko	191 ko	-2.1%
Stanford Bunny[2]	35 947	69 451	2.9 Mo	3.4 Mo	+16.0%	2.3 Mo	2.7 Mo	+18.1%
Armadillo[2]	172 974	345 944	21.7 Mo	19.9 Mo	-8.2%	13.8 Mo	12.2 Mo	-11.2%
Dragon[2]	437 645	871 414	33.7 Mo	29.9 Mo	-11.3%	33.0 Mo	30.0 Mo	-9.1%
Happy bouddha[2]	543 652	1 087 716	41.5 Mo	36.2 Mo	-12.7%	41.6 Mo	37.4 Mo	-10.2%
Turbine blade[3]	882 954	1 765 388	84.8 Mo	70.6 Mo	-16.8%	69.0 Mo	56.9 Mo	-17.7%

[1]: Models from Vanek et al. [5];
[2]: Models from "The Stanford 3D Scanning Repository";
[3]: Models from the "Large Geometric Models Archive" of the Georgia Institute of Technology

Table 1: File size reductions observed by using the proposed encoding framework (implementing the HOCA and the BTA mechanisms).

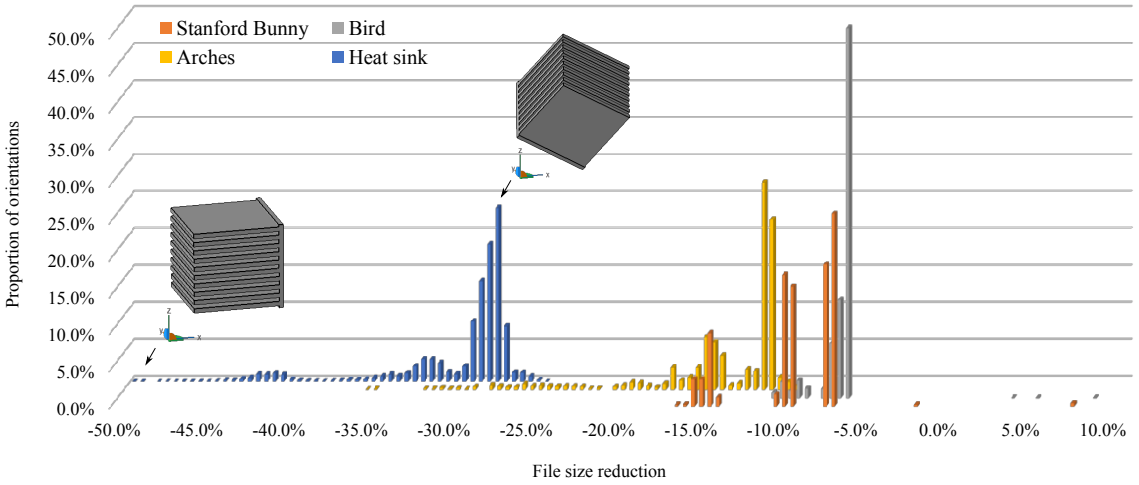

Fig. 3: Distribution of orientations according to the file size reduction (with respect to the initial orientation) for several test cases: the Stanford Bunny (in red), the Bird (in grey), the Arches (in orange) and the Heat Sink (in blue).

comparison to using their original orientation.

Though the impact of the geometry orientation on the file size reduction efficiency has been highlighted, the identification of the best orientation for this purpose remains to examine. Because of the

infinity of the research space, a metaheuristic algorithm could be employed to quickly determine a good orientation for a particular geometry.

References:

[1] Davoodi, P.; Raman, R.; Satti, S. R.: On succinct representations of binary trees, Mathematics in Computer Science, 11, 2017, 177-189. https://doi.org/10.1007/s11786-017-0294-4

[2] Dilberoglu, U. M.; Gharehpapagh, B.; Yaman, U.; Dolen, M.: The Role of additive manufacturing in the era of industry 4.0, Procedia Manufacturing, 11, 2017, 545-554. https://doi.org/10.1016/j.promfg.2017.07.148

[3] Rayna, T.; Striukova, L.; Darlington, J.: Co-creation and user innovation: The role of online 3D printing platforms, Journal of Engineering and Technology Management, 37, 2015, 90-102. https://doi.org/10.1016/j.jengtecman.2015.07.002

[4] Taubin, G.; Horn, W. P.; Lazarus, F.; Rossignac, J.: Geometry coding and VRML, Proceedings of the IEEE, 86(6), 1998, 1228-1243. https://doi.org/10.1109/5.687837

[5] Vanek, J.; Galicia, J. A. G.; Benes, B.: Clever support: efficient support structure generation for digital fabrication, Computer Graphics Forum, 33(5), 2014, 117-125. https://doi.org/10.1111/cgf.12437