International Journal of Computer Trends and Technology
ISSN: 2231 – 2803 / https://doi.org/10.14445/22312803/IJCTT-V68I11P108

Volume 68 Issue 11, 59-63, November 2020
© 2020 Seventh Sense Research Group®

*Original Article*

# Compressed Representation of Color Information for Converting 2D Images Into 3D Models

Poorna Banerjee Dasgupta

*M.Tech Computer Science and Engineering, Nirma Institute of Technology, Ahmedabad, Gujarat, India.*

Received Date: 14 October 2020
Revised Date: 22 November 2020
Accepted Date: 24 November 2020

***Abstract*** *- Three-dimensional (3D) modeling is applied in numerous fields such as medical imaging, archaeology, industrial fabrication of machine parts and tools, gaming, animation, architecture, and interior design. In several such applications, the 3D model is developed based on an initial two-dimensional (2D) image of the model. An important aspect of converting a 2D image into a 3D model is the preservation and representation of the associated color information. With the advent of 3D printing technology, colored 3D models can be developed from their corresponding 2D images and can then be printed with the aid of 3D slicing software. Although several file formats are available for supporting generic 3D printing, only a few of them support colored 3D printing. Additionally, if the 3D model is complex, the associated color data will be huge, which will lead to longer processing times and greater storage requirements for both the 3D modeling software and the 3D slicing software. Hence, in this study, a lossy compression algorithm is proposed for representing the color information associated with OBJ and MTL file formats. The proposed compression algorithm has been designed to reduce the memory storage required for representing a 3D model's color information without significantly affecting the model's visual color appearance.*

***Keywords*** *- 3D modeling, 3D printing, lossy compression, MTL file format, OBJ file format.*

## I. INTRODUCTION

In generic terms, 3D modeling refers to the process of developing a mathematical representation of any surface of an object in three dimensions by employing specialized software tools. 3D modeling has applications in numerous fields such as medical imaging, archaeology, industrial fabrication of machine parts and tools, gaming, animation, architecture, and interior design. In several such applications, the 3D model is created based on an initial 2D image of the model. One of the most common techniques employed for converting a 2D image into its corresponding 3D model involves adding depth to the existing 2D image, also termed as image extrusion. For example, Figure 1(a) depicts a simple rectangle, and Figure 1(b) depicts the corresponding cuboid obtained after adding depth (padding) to the said rectangle by using the FreeCAD software. The process of image extrusion can be employed for adding depth to both elementary and complex shapes. After the 3D modeling process is complete, the model can subsequently be used in graphics applications or can be used for 3D printing. In this study, the emphasis has been laid on 3D printing applications.

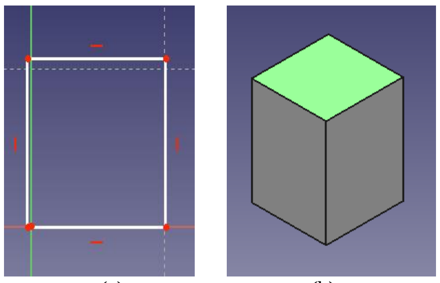


**Fig. 1 Example of image extrusion**
**(a) 2D rectangle (b) Corresponding padded cuboid**

An important aspect of converting a 2D image into its 3D model is the preservation and representation of the associated color information. With the advent of 3D printing technology, colored 3D models can be developed from their corresponding 2D images and can then be printed with the aid of 3D slicing software. Slicing software (slicer) is often used for converting a 3D model into specific instructions for the 3D printer. Although various file formats are available for developing and storing a 3D model, only a few of them support color representation and color 3D printing. Some of the file formats that support 3D printing are STL, OBJ, VRML, AMF, and 3MF. Out of these, the STL and OBJ file formats are predominantly used for 3D printing [1][2].

The STL file format was developed by 3D Systems and is widely used for rapid prototyping, 3D printing, and computer-aided manufacturing [3]. The STL file format is compatible with several software packages associated with 3D modeling and printing. There are two primary versions of the STL format, namely ASCII STL and binary STL. Although the STL format is a popular choice for 3D printing, it suffers from certain drawbacks. Since the STL format incorporates triangular tessellation for modeling an object's surface, the corresponding file size can become extremely large while modeling complex curved surfaces with high accuracy [4]. Additionally, the

STL format provides very limited support for color representation. In some binary STL formats, such as those used by VisCAM, SolidView, and Materialise Magics software packages, a 16-bit parameter known as the "attribute byte count" contains color information and is accordingly specified for each tessellated triangle[3][5]. However, adding color information for each tessellated triangle can further increase the file size and processing time. Furthermore, such binary STL format variations are non-standard and are often not supported by other software packages. Consequently, the STL format is primarily chosen for monochrome 3D printing.

The OBJ file format was developed by Wavefront Technologies and is a popular choice for both 3D printing and 3D graphics designing applications [1][6]. Unlike the STL format, the OBJ format allows modeling of surfaces through polygonal tessellation, freeform curves (such as cardinal splines and Bezier curves), and freeform surfaces (non-uniform rational B splines or NURBS). Furthermore, the OBJ format provides support for representing color and texture information through a supplementary file format, namely the material template library (MTL) file format [7][8]. MTL files are represented in the ASCII format, and they contain surface shading properties of objects that can be referenced by OBJ files for rendering 3D objects. In an MTL file, the light reflecting properties of a surface are specified according to the Phong reflection model [9].

Although the OBJ format supports the colored rendering of 3D objects, the MTL file size can become quite large when numerous materials with several color and texture properties are defined in the file. Furthermore, if an OBJ file containing objects with complex surface geometries repeatedly references a large MTL file, the corresponding processing time will be high as well. Based on this premise, in this study, an algorithmic approach based on lossy compression is proposed for representing and compacting the color information of a 3D model included in an MTL file, without significantly affecting the model's visual color appearance. In Section II, the OBJ and MTL formats are described in detail, and the proposed algorithm is elaborated through two case-studies. Section III comprises an in-depth analysis of the obtained results. Finally, in Section IV, the conclusions derived from the proposed study are elucidated, and the future scope of work is discussed.

## II. LOSSY COMPRESSION OF COLOR INFORMATION

### A. *MTL and OBJ File Formats*

As explained earlier, the OBJ format supports colored rendering of 3D models with the aid of a companion file format, namely the MTL format. Typically, to reference an MTL file, the OBJ file will include "mtllib" and "usemtl" commands, as shown [8][10]:

*mtllib* [external .mtl file name]

*usemtl* [material name]

The "mtllib" command is used for declaring the name of the referenced MTL file that contains one or more named material definitions. The "usemtl" command is employed for referencing a particular named material from the declared MTL file. To represent color information associated with defined materials, an MTL file includes several light-reflecting parameters in accordance with the Phong reflection model [8–10]. In Table, I, various MTL light-reflecting properties with their corresponding data types and value ranges are summarized.

**Table 1.**
**Light-Reflecting Properties Specified In An Mtl File**

| Name | Significance | Data Type | Value Range |
|---|---|---|---|
| *Ka* | Represents the ambient material color in RGB format. | Floating point | The value of each color channel lies in the range of 0.0–1.0. |
| *Kd* | Represents the material's diffuse reflectivity in RGB format. | Floating point | The value of each color channel lies in the range of 0.0–1.0. |
| *Ks* | Represents the material's specular reflectivity in RGB format. | Floating point | The value of each color channel lies in the range of 0.0–1.0. |
| *Ns* | Represents the material's specular exponent. | Floating point | The value lies in the range 0.0–1000.0. |
| *Ni* | Represents the material's optical density (index of refraction). | Floating point | The value lies in the range of 0.0–10.0. |
| *Tr/d* | Represents the material's transparency/non-transparency. | Floating point | The value lies in the range of 0.0–1.0. |
| *Tf* | Represents the material's transmission filter color in RGB format. | Floating point | The value of each color channel lies in the range of 0.0–1.0. |
| *illum* | Represents the material's illumination model. | Integer | The value lies in the range of 0–10. |

It should be noted that the parameters *Tr* (transparency) and *d* (dissolve or non-transparency) are exact opposites of each other, and only either one of them needs to be specified in the MTL file. Furthermore, the value of the parameter *illum* determines the material's lighting and shading effects such as spectral highlights and shadow casting. It can be observed from Table I that majority of the parameter values have the floating-point data type, which is stored in a 32-bit format based on the IEEE-754 standard [13]. Hence, if an MTL file contains several material definitions, each material definition having numerous defined properties, the file size can become quite large, which in turn increases the processing time. A compression algorithm is thus required for compaction of the color information included in the MTL file.

## *B. Compressing Color Information*

Contrary to lossless compression algorithms, a lossy compression algorithm incorporates inexact approximations and partial discarding for representing data [11][12]. Lossy compression techniques are employed for reducing data size to efficiently store, handle, and transmit large data, and they are often applied to multimedia data such as audio, image, and video data. In the proposed algorithmic approach, a lossy compression scheme was designed for reducing the data size of color information stored in an MTL file. The proposed algorithmic approach comprises three components, namely the sending and receiving MTL file parsers and the intermediate structure (IS). The sending parser reads through an MTL file, collects and compresses the necessary color information, and generates the corresponding IS that stores the compacted color information. The generated IS can subsequently be exported and transmitted to other 3D modeling and slicing software. The receiving parser imports the transmitted IS and generates a new MTL file based on the compressed data values, which can be incorporated by other 3D modeling and slicing software. Parsing is performed based on the following rules:

- The light-reflecting parameters are stored in the order: *Ka, Kd, Ks, Ns, Ni, Tr/d, Tf*, and *illum*.
- All values in the IS are stored as 8-bit ASCII characters.
- All floating values between zero and one are rounded and terminated at the first decimal position. For example, 0.5963 will be stored as 0.6.
- All values for color channels are stored in the RGB order.
- Parameter *Ns* is represented by five characters, and its value can range from 0.000 to 999.9.
- Parameter *Ni* is represented by three characters, and its value can range from 0.0 to 9.9.
- Parameter *Tr/d* is represented by four characters, and its value can range from 0.0 to 1.0. The fourth character is either 0 or 1 depending on whether *Tr* or *d* is being represented; 0 indicates the parameter is *Tr*, and 1 indicates the parameter is *d*.
- Parameter *illum* is represented by two characters and its value can range from 0 to 10.
- Any missing parameter is represented by the null character '\0'.
- The end of a material definition is indicated by the character 'E' unless it's the last material definition.

In this study, the proposed parsers and IS were designed with the help of the C programming language. Figure 2 illustrates the proposed IS template.

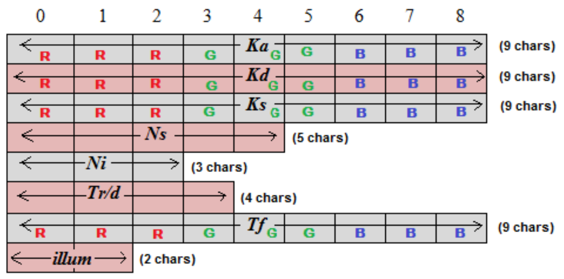

0 1 2 3 4 5 6 7 8
R R R G Ka G G B B B (9 chars)
R R R G Kd G G B B B (9 chars)
R R R G Ks G G B B B (9 chars)
Ns (5 chars)
Ni (3 chars)
Tr/d (4 chars)
R R R G Tf G G B B B (9 chars)
illum (2 chars)

**Fig. 2 IS template**

Based on the IS template demonstrated in Figure 2, the proposed sending parser algorithm was applied for two 3D modeling case-studies, namely a triangular prism and a cuboid. Figure 3(a) depicts the initial 2D image of a triangle, and Figure 3(b) illustrates the corresponding 3D prism obtained after image extrusion. Figure 3(c) depicts the initial 2D image of a square, and Figure 3(d) illustrates the corresponding 3D cuboid obtained after image extrusion. In this study, the 3D modeling and rendering of OBJ and MTL files were done with the help of the Blender software package.

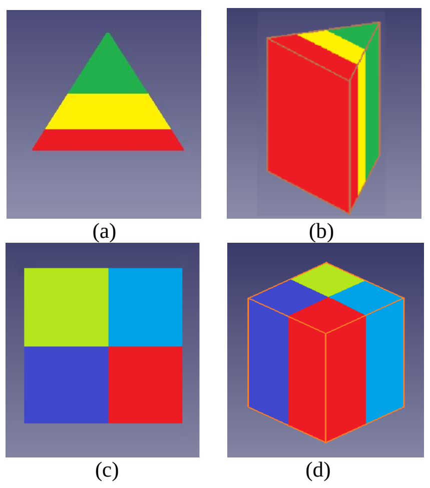
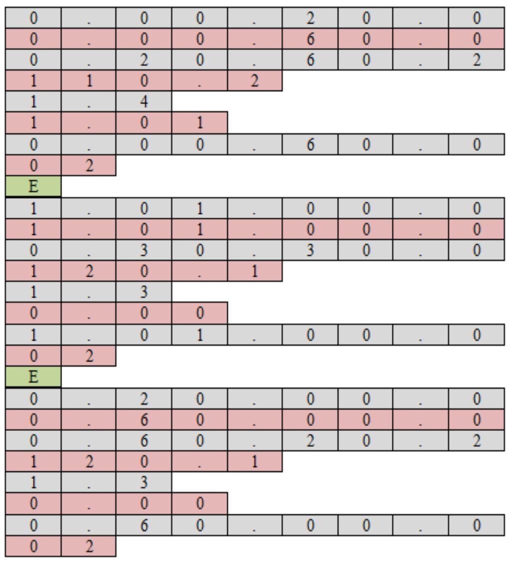


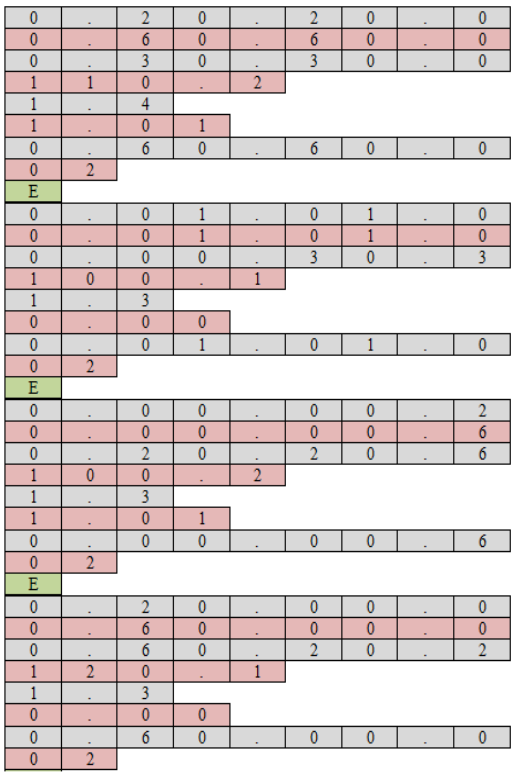
**Fig. 3 Conversion of the 2D image to 3D model**
**(a) Initial 2D triangle (b) Corresponding padded prism**
**(c) Initial 2D square (d) Corresponding padded cuboid**

As observed from Figure 3(a) and Figure 3(b), three colors (i.e., three material definitions—G1, Y1, and R1) were used in the original MTL file for the 3D prism. Similarly, as observed from Figure 3(c) and Figure 3(d), four colors (i.e., four material definitions—G2, B1, B2, and R1) were used in the original MTL file for the 3D cuboid. These MTL file specifications used for the prism and cuboid are listed in Table II and Table III, respectively. The corresponding ISs generated by the proposed sending parser algorithm for the prism and cuboid are depicted in Figure 4 and Figure 5, respectively.



| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | . | 0 | 0 | . | 2 | 0 | . | 0 |
| 0 | . | 0 | 0 | . | 6 | 0 | . | 0 |
| 0 | . | 2 | 0 | . | 6 | 0 | . | 2 |
| 1 | 1 | 0 | . | 2 | | | | |
| 1 | . | 4 | | | | | | |
| 1 | . | 0 | 1 | | | | | |
| 0 | . | 0 | 0 | . | 6 | 0 | . | 0 |
| 0 | 2 | | | | | | | |
| E | | | | | | | | |
| 1 | . | 0 | 1 | . | 0 | 0 | . | 0 |
| 1 | . | 0 | 1 | . | 0 | 0 | . | 0 |
| 0 | . | 3 | 0 | . | 3 | 0 | . | 0 |
| 1 | 2 | 0 | . | 1 | | | | |
| 1 | . | 3 | | | | | | |
| 0 | . | 0 | 0 | | | | | |
| 1 | . | 0 | 1 | . | 0 | 0 | . | 0 |
| 0 | 2 | | | | | | | |
| E | | | | | | | | |
| 0 | . | 2 | 0 | . | 0 | 0 | . | 0 |
| 0 | . | 6 | 0 | . | 0 | 0 | . | 0 |
| 0 | . | 6 | 0 | . | 2 | 0 | . | 2 |
| 1 | 2 | 0 | . | 1 | | | | |
| 1 | . | 3 | | | | | | |
| 0 | . | 0 | 0 | | | | | |
| 0 | . | 6 | 0 | . | 0 | 0 | . | 0 |
| 0 | 2 | | | | | | | |

**Fig. 4 Generated IS for prism**



| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | . | 2 | 0 | . | 2 | 0 | . | 0 |
| 0 | . | 6 | 0 | . | 6 | 0 | . | 0 |
| 0 | . | 3 | 0 | . | 3 | 0 | . | 0 |
| 1 | 1 | 0 | . | 2 | | | | |
| 1 | . | 4 | | | | | | |
| 1 | . | 0 | 1 | | | | | |
| 0 | . | 6 | 0 | . | 6 | 0 | . | 0 |
| 0 | 2 | | | | | | | |
| E | | | | | | | | |
| 0 | . | 0 | 1 | . | 0 | 1 | . | 0 |
| 0 | . | 0 | 1 | . | 0 | 1 | . | 0 |
| 0 | . | 0 | 0 | . | 3 | 0 | . | 3 |
| 1 | 0 | 0 | . | 1 | | | | |
| 1 | . | 3 | | | | | | |
| 0 | . | 0 | 0 | | | | | |
| 0 | . | 0 | 1 | . | 0 | 1 | . | 0 |
| 0 | 2 | | | | | | | |
| E | | | | | | | | |
| 0 | . | 0 | 0 | . | 0 | 0 | . | 2 |
| 0 | . | 0 | 0 | . | 0 | 0 | . | 6 |
| 0 | . | 2 | 0 | . | 2 | 0 | . | 6 |
| 1 | 0 | 0 | . | 2 | | | | |
| 1 | . | 3 | | | | | | |
| 1 | . | 0 | 1 | | | | | |
| 0 | . | 0 | 0 | . | 0 | 0 | . | 6 |
| 0 | 2 | | | | | | | |
| E | | | | | | | | |
| 0 | . | 2 | 0 | . | 0 | 0 | . | 0 |
| 0 | . | 6 | 0 | . | 0 | 0 | . | 0 |
| 0 | . | 6 | 0 | . | 2 | 0 | . | 2 |
| 1 | 2 | 0 | . | 1 | | | | |
| 1 | . | 3 | | | | | | |
| 0 | . | 0 | 0 | | | | | |
| 0 | . | 6 | 0 | . | 0 | 0 | . | 0 |
| 0 | 2 | | | | | | | |

**Fig. 5 Generated IS for cuboid**

**Table 2.**
**Mtl light-reflecting property specification for prism**

| | **Prism** | | |
|---|---|---|---|
| **Material Name** | G1 | Y1 | R1 |
| ***Ka*** *(RGB)* | 0.0000<br>0.1993<br>0.0000 | 1.0000<br>1.0000<br>0.0000 | 0.2314<br>0.0000<br>0.0000 |
| ***Kd*** *(RGB)* | 0.0166<br>0.6102<br>0.0000 | 1.0000<br>1.0000<br>0.0000 | 0.6022<br>0.0166<br>0.0000 |
| ***Ks*** *(RGB)* | 0.2084<br>0.5991<br>0.2084 | 0.2991<br>0.2991<br>0.0000 | 0.5963<br>0.2014<br>0.1974 |
| ***Ns*** | 110.2133 | 120.1123 | 120.1217 |
| ***Ni*** | 1.4 | 1.33 | 1.26 |
| ***Tr/d*** | d=1.0 | Tr=0.0 | Tr=0.0 |
| ***Tf*** *(RGB)* | 0.0166<br>0.6102<br>0.0000 | 1.0000<br>1.0000<br>0.0000 | 0.6022<br>0.0166<br>0.0000 |
| ***illum*** | 2 | 2 | 2 |

**Table 3.**
**Mtl light-reflecting property specification for cuboid**

| | **Cuboid** | | | |
|---|---|---|---|---|
| **Material Name** | G2 | B1 | B2 | R1 |
| ***Ka*** *(RGB)* | 0.1993<br>0.1993<br>0.0000 | 0.0000<br>1.0000<br>1.0000 | 0.0000<br>0.0000<br>0.1986 | 0.2314<br>0.0000<br>0.0000 |
| ***Kd*** *(RGB)* | 0.6145<br>0.6145<br>0.0000 | 0.0000<br>1.0000<br>1.0000 | 0.0166<br>0.0000<br>0.5922 | 0.6022<br>0.0166<br>0.0000 |
| ***Ks*** *(RGB)* | 0.2991<br>0.2991<br>0.0000 | 0.0000<br>0.2973<br>0.2973 | 0.2084<br>0.2084<br>0.5974 | 0.5963<br>0.2014<br>0.1974 |
| ***Ns*** | 110.2133 | 100.1425 | 100.2237 | 120.1217 |
| ***Ni*** | 1.4 | 1.28 | 1.31 | 1.26 |
| ***Tr/d*** | d=1.0 | Tr=0.0 | d=1.0 | Tr=0.0 |
| ***Tf*** *(RGB)* | 0.6145<br>0.6145<br>0.0000 | 0.0000<br>1.0000<br>1.0000 | 0.0166<br>0.0000<br>0.5922 | 0.6022<br>0.0166<br>0.0000 |
| ***illum*** | 2 | 2 | 2 | 2 |

## III. RESULT ANALYSIS

The ISs generated by the sending parser algorithm were decoded by the receiving parser algorithm, and the MTL light-reflecting properties were extracted for creating new MTL files based on the extracted values. Figure 6(a) illustrates the regenerated prism, and Figure 6(b) depicts the regenerated cuboid. Upon visual comparison with their respective original counterparts, i.e., Figure 3(b) and

Figure 3(d), it can be observed that there is no discernible change in the visual color appearance of the 3D models.

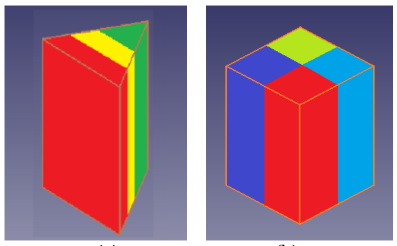


**Fig. 6 Regenerated 3D models (a) Prism (b) Cuboid**

Statistics of the achieved data compression are summarized in Table IV. It can be observed from Table IV that by generating ISs, the proposed algorithmic approach can achieve significant compression of color information (up to 71%) without any apparent change in the visual color appearance. It should be noted that the file sizes mentioned in Table IV refer to the actual data size, not the disk storage size (which varies according to the file allocation block-size used by the host operating system). Furthermore, since the compressed representation of color information through the IS template can help realize a considerable reduction in data size, the associated transmission and processing times get reduced as well.

**Table 4. Statistics of achieved data compression**

| | **Prism** | **Cuboid** |
|---|---|---|
| **Original MTL File Size** | 522 Bytes | 701 Bytes |
| **Regenerated MTL File Size** | 414 Bytes | 553 Bytes |
| **IS Size** | 152 Bytes | 203 Bytes |
| **Compression Rate** | compression of MTL file size: (1 - 414/522) x 100 = 21% compression of file size with respect to IS: (1 - 152/522 ) x 100 = **70.9%** | compression of MTL file size: (1 - 553/701) x 100 = 21.1 % compression of file size with respect to IS: (1 - 203/701 ) x 100 = **71%** |

In this study, an algorithmic approach was developed for the compressed representation of color information included in an MTL file, which serves as the companion file format for OBJ files, while rendering colored 3D models from 2D images. The proposed algorithmic approach comprises three components, namely the sending and receiving parsers and the IS. By incorporating the compacted representation of color information in the generated IS, a considerable reduction in the data size is achieved without significantly affecting the model's visual color appearance, which in turn decreases the data transmission and processing times.

## IV. CONCLUSION

In this study, the OBJ and MTL file formats were chosen because the emphasis was laid on 3D printing applications. Furthermore, while compressing the color information, the emphasis was laid on light-reflecting properties. As future work, the proposed algorithmic approach can be extended to 3D graphics applications, and other parameters associated with color information (such as texture maps) can be included for data compression. Furthermore, the proposed IS can further adapted for including format variations of color parameters, and duplicate data values occurring in the IS can be further compressed.

## REFERENCES

[1] Dibya Chakravorty. (2018) OBJ File Format. [Online] Available: https://all3dp.com/1/obj-file-format-3d-printing-cad

[2] Cătălin Iancu, Daniela Iancu, and Alin Stăncioiu, From CAD Model To 3D Print Via "STL" File Format, Fiability & Durability, 1(2010) 73-80.

[3] StereoLithography Interface Specification, 3D Systems Inc.,(1989).

[4] Dibya Chakravorty. (2019) STL File Format. [Online] Available: https://all3dp.com/what-is-stl-file-format-extension-3d-printing/

[5] Jonathan D. Hiller and Hod Lipson, STL 2.0: A Proposal for a Universal Multi-Material Additive Manufacturing File Format, Cornell University. (2009).

[6] WANG Jin-feng and YAO Guo-qing, OBJ Three-dimensional Model File Format in OpenGL Input and Processing, Computer Knowledge and Technology, 2011.

[7] Cătălin Iancu, 3D PRINTING FILE FORMATS, Constantin Brâncuşi University, 2018.

[8] MTL Files: Material Definitions for OBJ Files. (2018) [Online] Available: https://people.sc.fsu.edu/~jburkardt/data/mtl/mtl.html

[9] Bui Tuong Phong, Illumination for computer-generated pictures, Communications of ACM 18, No. 6, 311–317, 1975.

[10] Diane Ramey, Linda Rose, and Lisa Tyerman, MTL Material Format (Lightwave, OBJ), Alias and Wavefront Inc., 1995.

[11] Pavithra Mohan, "Enhanced Image Compression System" SSRG International Journal of Mobile Computing and Application 6.3 (2019): 1-7.

[12] Ian H. Witten et al., Semantic and Generative Models for Lossy Text Compression, The Computer Journal, Oxford University Press, 1994.

[13] IEEE 754-2019 - IEEE Standard for Floating-Point Arithmetic. (2019) [Online]

[14] Available: https://standards.ieee.org/standard/7542019.html