STL 2.0: A PROPOSAL FOR A UNIVERSAL MULTI-MATERIAL ADDITIVE MANUFACTURING FILE FORMAT

Jonathan D. Hiller, Hod Lipson
Mechanical and Aerospace Engineering
Cornell University
Ithaca NY 14853, USA

The de-facto standard STL file format has served the rapid prototyping community for over two decades, but falls short with the advent of new technological developments such as the ability to handle multiple and graded materials, specify volumetric digital inkjet patterns and surface colors. We study a variety of requirements for additive fabrication technologies and propose a new compact XML-based file format. The new Additive Manufacturing File (AMF) format allows the resolution-independent specification of geometry and material properties. Regions may be defined geometrically either using a triangle mesh, using functional representations, or through a voxel bitmap. Each region is associated with a material, which may be defined as a base (single) material or hierarchically by a combination of other materials, either functionally (enabling smooth gradients) or voxel-wise (for arbitrary microstructure). Files can be self-contained or refer to external or online material libraries. With a simple conversion, the AMF file format is both forward and backwards compatible with the current standard STL format, and the flexibility of the XML structure enables additional features to be adopted as needed by CAD programs and future additive manufacturing processes. Code and examples are publicly available.

Introduction

Additive manufacturing technology is quickly evolving from producing primarily single-material, homogenous shapes to producing multi-material geometries in full color with functionally graded materials and microstructures. Already, several vendor-specific file formats have been introduced to overcome the lack of a standard interchange file format that contains these features. This results in little or no compatibility between files for machines from different vendors, such as the ObjDF format for Objet's Connex^{TM} printers (Objet, 2009) and ZCorp's color ZPR format (ZCorporation, 2009). Here, we propose a framework for a simple, intuitive file format to address the severe limitations of the industry standard single-material STL file format with regards to the additive manufacturing technology of the future.

There is a tradeoff between the generality of a file format, and its usefulness for a specific purpose (McMains et al., 2002). Thus, features designed to meet the needs of one community may hinder the usefulness of a file format for other uses. In order to be successful across the field of additive manufacturing, a file format should address the following concerns:

(1) Technology independent: A file format should describe an object in a general way such that any machine can build it to the best of its ability (Jacob et al., 1999). A suitable file format should be resolution and layer-thickness independent, and must not contain

information specific to any one manufacturing process or technique. This does not negate the inclusion of properties that only certain advanced machines support (color, multiple materials, etc.), but they must be defined in such away to avoid exclusivity.

(2) Simplicity: A file format should be easy to implement and understand. A file should be able to be read and debugged in a simple ASCII text viewer to encourage understanding and adoption. However, this is at odds with the size requirements of the file format. Also, there is no place for redundancy in a file format, and no identical information should be stored in multiple places.

(3) Scalability: As the parts to print increase in size and complexity, the file format should scale well. This includes being able to handle large arrays of identical objects, complex repeated internal features (e.g. meshes), and multiple components arranged in the optimal packing for printing.

(4) Future compatibility: In order to remain useful in a rapidly changing industry, a file format must be easily extensible while remaining compatible with earlier versions and technologies. This allows new features to be added as advances in technology warrant, while still working flawlessly for simple homogenous geometries on the oldest hardware.

Additionally, there are several specific aspects of an additive manufacturing interchange format that are addressed in the proposed format to meet the demands of the current generation of research and hardware. These pressing aspects include:

- Multiple/graded materials
- Colors and surface textures
- Hierarchical microstructure and mesostructure

Background

For the last two decades, the STL file format has been the industry standard for transferring information between design programs and the software specific to a given additive manufacturing hardware (Kumar and Dutta, 1997, Jurrens, 1999, Hague and Reeves, 2000). An STL file contains information only about a surface mesh, and has no provisions for representing color, texture, material, etc, although several extensions have been proposed (but not widely accepted) (Chiu and Tan, 2000, Stroud and Xirouchakis, 2000). The surface is represented by an unordered list of triangles, and each triangle is defined by 12 floating point numbers. A 3D surface normal is defined, followed by three coordinates that define the vertices of the triangle in three dimensions. Already, this contains redundant information, since the surface normal can be calculated from the order and location of the three vertices. By default, the right-hand rule is used to define the direction of the normal based on the order the points are encoded. Since each triangle is represented separately, each vertex must be written repeatedly for every triangle that shares that vertex (three or more times). This introduces leaks, where small rounding errors result in vertices that do not precisely line up, which make subsequent slicing algorithms ineffective without an intermediate "welding" step to (hopefully) repair these defects.

Also, the STL file has no provisions for defining the physical units intended. Even though pre-processing software can make an educated guess between inches or mm depending on the build size of the machine, there is still unnecessary ambiguity. An additional point of confusion regarding the STL file is that in fact there are two separate file formats that may be used: binary and ASCII. The ASCII version exists to make the format human readable, but the binary version is often used by mature programs to minimize storage space. A summary of the advantages and disadvantages of the current STL file format is shown in Table 1.

Table 1: Advantages and disadvantages of the current STL format.

Advantages	**Disadvantages**
Simple	Geometry leaks
Sequential memory access*	No specified units
Portable	Unnecessary redundancy
	Incompatible with color, multiple materials, etc
	Poor scalability
*Does not require large amounts of RAM, critical in '80s	Lacks auxiliary information

Many 3D file formats have been used over the years for a wide variety of purposes. Several of these have been proposed for use within the Additive Manufacturing (AM) community to replace the STL. However, none has gained traction (Pratt et al., 2002, Patil et al., 2002, Rock and Wozny, 1991) for two main reasons. First, up until recently the AM end-user community has not needed functionality past what the STL offered. Secondly, the more general file formats in existence include many features that are irrelevant for the AM field, and do not include many features that would be useful, which causes unnecessary coding and complexity.

Several file formats that have been proposed for the additive manufacturing community are summarized below:

- **X3D (VRML)**: (Virtual Reality Modeling Language) This mesh-based file format was intended to allow 3D content to be viewed over the web. As such, it includes information about a 3D surface and its color, but also information that is not relevant for AM, such as transparency, animations, lights, sounds, and embedded navigation URLs. Other disadvantages include no provisions for defining multiple materials within a given mesh or arbitrary microstructure.
- **STEP:** This format is a general-use solid model representation, using extruded and swept solids, wireframe, boolean primitives modeling, and many other modeling paradigms to represent a 3D object. (Gilman and Rock, 1995) As such it is unnecessarily complex for the needs of the AM community.
- **PLY**: This format was intended to store and view data from 3D scanners. It uses polygon meshes and can include information about texture and color. However, like other purely mesh-based formats it does not define materials or microstructure volumetrically.

- **SAT:** The ACIS SAT format is widely used for boundary-representation (B-Rep) objects in CAD packages. However, the entire format revolves around its internal topological data structure, which makes it difficult to understand and unsuitable for an exchange format.
- **OBJ**: This meshed file format is simple, compact, widely accepted in the 3D modeling community, and can map textures easily. However, it lacks the ability to define materials or microstructure volumetrically.
- **DXF**: Although the DXF format allows the definition of 3D triangle meshes and solids, it was originally intended for 2D drawings and remains best suited for such.
- **3DS**: Another triangular mesh-based format with color and texture information. It is limited to 65536 vertices and polygons, and contains much information not necessary for the AM industry, such as lighting and animation info.
- **SLC:** The SLC format represents individual 2D slices of a 3D object as contours representing internal and external boundaries. However, since a specific Z-slice distance is assumed, this format is not suitable for a cross-platform interchange format.

Proposed AMF Format

In order to address the lack of a suitable file format for the additive manufacturing industry, we propose the following framework for a simple, flexible, extensible file format.

First, the information should be stored in standard XML format. Using this widely accepted data format opens the door to a rich host of tools for creating, viewing, manipulating, and storing AMF files. However, the beauty of XML is that it accomplishes this without alienating programmers who wish to code low-level native parsing/storing routines. XML is human readable, which makes debugging errors in the file possible. Unlike many of the current file formats which specify completely separate ASCII and binary file formats, the AMF format will be store entirely in ASCII XML, then compressed if desired in a post-processing step using highly optimized standardized compression routines. This allows significantly smaller file sizes without maintaining multiple parallel file specifications.

Another significant advantage of XML is its inherent flexibility. Missing or extra parameters do not present a problem for a parser as long as the document conforms to the XML standard. Practically, this allows new features to be added without needing to update old versions of the parser, such as in legacy software.

Top Level Tags

There are four top level tags in the AMF file, of which only a single object tag is required for a fully functional AMF file that encompasses the usefulness of the STL format.

`<Object>` The object tag defines a region or regions of material, each of which are associated with a material ID for printing.

<Constellation> The constellation tag hierarchically combines objects and other constellations into a relative pattern for printing. If no constellation tags are specified, each object tag will be imported with no relative position data.

<Palette> The palette tag defines one or more named materials for printing with an associated material ID. If no palette tag is included, a single default material is assumed.

<Print> The print tag specifies which constellations and/or objects to print, and is necessary only if multiple constellations or objects introduce ambiguity as to how many of each to print.

Examples

Basic STL Equivalent

The first example presents a simple case of an AMF file that contains all the functionality of an STL file (Figure 1). Note that in the sample code, the ellipses (...) denotes a continuing, similar list of tags.

```
<?xml version="1.0"?>
<AMF>
  <Object PrintID = "0" units = "mm">
    <Mesh>
      <Vertices>
        <Vertex VertexID="0">
          <VertexLocation x="0" y="1.332" z="3.715"/>
        </Vertex>
        <Vertex VertexID="1">
          <VertexLocation x="0" y="1.269" z="3.715"/>
        </Vertex>
        ...
      </Vertices>

      <Region FillMaterialID = "0">
        <Triangle V1 = "0" V2 = "1" V3 = "3"/>
        <Triangle V1 = "0" V2 = "1" V3 = "4"/>
        ...
      </Region>
    </Mesh>
  </Object>
</AMF>
```

(a)

(b)

Figure 1: The AMF file can use a simple list of vertices and triangles to define a mesh and replicate the functionality of an STL file, but without leaks. The XML-compliant format makes storing, parsing, and reading the file easy.

The opening `<AMF>` tag is necessary to denote the file type, as well as fulfill the requirement that all XML files have a single root element. The top level `<Object>` tag contains two sub-tags: `<Vertices>` and `<Region>`. The required `<Vertices>` tag lists all vertices that are used in this object, and assigns a unique vertex ID to each. The sub-tag `<VertexLocation>` gives the position of the point in 3D space. After the vertex information, at least one `<Region>` tag must be included, using the `<Triangle>` tag to define triangles by the right-hand rule (vertices listed in counter-clockwise order as viewed from the outside) from the indices of the defined vertices. The problem of STL

leaks is solved by the fact that common vertices of triangles reference the same <Vertex> tag.

Multiple-material STL equivalent

One of the most critical limitations of the current STL format is the lack of support for multiple materials. With the AMF format, this minor extension introduces the <Palette> tag. Here, any number of materials may be defined by name and associated with a material ID. Other relevant attributes may also be added to each material. Then, within the <mesh> tags, additional <Region> tags can be added that reference different material indices. Since the vertex list is shared, no leaks are introduced at the boundaries between materials (Figure 2).

```
<?xml version="1.0"?>
<AMF>
  <Palette>
    <Material MaterialID = "0">
      <Name>StiffMaterial</Name>
    </Material>
    <Material MaterialID = "1">
      <Name>FlexibleMaterial</Name>
    </Material>
  </Palette>

  <Object PrintID = "0" units = "mm">
    <Mesh>
      <Vertices>
        ...
      </Vertices>

      <Region FillMaterialID = "0">
        ...
      </Region>
      <Region FillMaterialID = "1">
        <Triangle V1 = "5" V2 = "6" V3 = "7"/>
        <Triangle V1 = "5" V2 = "7" V3 = "9"/>
        ...
      </Region>
    </Mesh>
  </Object>
</AMF>
```

(a) (b)

Figure 2: With the addition of the palette tag, multiple materials may be easily defined and assigned to different regions. A common vertex list ensures no leaks between materials.

Gradated material example

The power of the material palette comes from the fact that "meta" materials may be defined using any previously defined (i.e. lower material index) materials. These "meta" materials may be defined functionally, enabling arbitrarily simple or complex gradients of two or more materials to be defined as a single material (Figure 3). When defining functions, the only variables that should be used are "x", "y" and "z", representing the respective spatial coordinates. A list of acceptable operations is given in Table 2.

Table 2: Operators used in the functional representation of material gradients and geometry in the AMF format.

Precedence	Operator	Description
1	**()**	Parentheses block
2	**^**	Power
3	*****	Multiply
3	**/**	Divide
3	**%**	Modulus
4	**+**	Add
4	**-**	Subtract
5	**=**	Equal
5	**<, <=**	Less than (or equal to)
5	**>, >=**	Greater than (or equal to)
6	**&**	Intersection (Logical AND)
6	**|**	Union (Logical OR)
6	****	Difference (Logical XOR)
6	**~**	Negation (Logical NOT)

The computational process to determine the relative concentrations of materials at any given sample point is quite easy. The equation tag for each material present is evaluated to a single number by plugging in the desired 3D spatial coordinates. All values are assumed to be zero where not defined, and are also floored at zero, so that it is possible to have regions that are fully a single material. Then the values for each material are added and the actual concentration of each is calculated as its proportion of the sum. Any equation using an undefined or self-referencing material ID is ignored.

```
<?xml version="1.0"?>
<AMF>
  <Palette>
    <Material MaterialID = "0">
      <Name>StiffMaterial</Name>
    </Material>
    <Material MaterialID = "1">
      <Name>FlexibleMaterial</Name>
    </Material>
    <Material MaterialID = "2">
      <Name>GradientMaterial</Name>
      <Equation UseMaterialID = "0">0.30*X</Equation>
      <Equation UseMaterialID = "1">0.30*(1-X)</Equation>
    </Material>
  </Palette>

  <Object PrintID = "0" units = "mm">
    ...
  </Object>
</AMF>
```

(a)

(b)

Figure 3: A "meta" material is defined in the material palette as a functional combination of two previously defined materials. This enables smooth gradients and arbitrary 3D material distributions within an object.

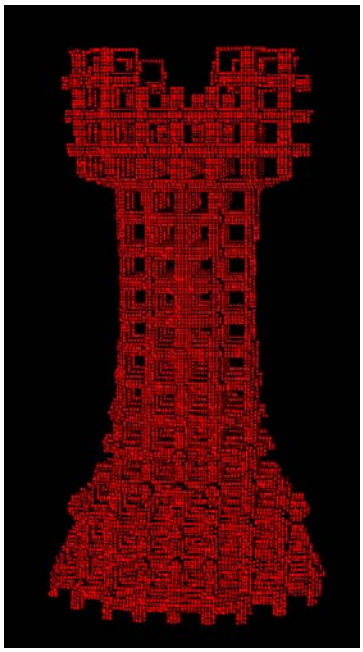
Mesostructure example

Meta-materials may also be defined as a tiled combination of materials and empty space to create micro and meso structures that are repeated throughout the region. These may

be defined functionally, by a mesh or a voxel bitmap. This allows complex internal structure to be defined once, then tiled for efficient use of storage space. The region is tiled at its envelope dimension, or in the case of the functional representation the modulus operator may be used to create periodic structures.

```
<?xml version="1.0"?>
<AMF>
  <Palette>
    <Material MaterialID = "0">
      <Name>StiffMaterial</Name>
    </Material>
    <Material MaterialID = "1">
      <Name>FlexibleMaterial</Name>
    </Material>
    <Material MaterialID = "2">
      <Name>MesoStructureMaterial</Name>
      <Lattice>
        ...
      </Lattice>
      <Structure>
        <X_Voxels>10</X_Voxels>
        <Y_Voxels>10</Y_Voxels>
        <Z_Voxels>10</Z_Voxels>
        <Data Layer = "1">0000110000...</Data>
        <Data Layer = "2">0001111000...</Data>
        ...
      </Structure>
    </Material>
  </Palette>

  <Object PrintID = "0" units = "mm">
    ...
  </Object>
</AMF>
```



(a) (b)

Figure 4: Mesostructure is defined by a voxel bitmap here using a simple repeated mesh pattern. Any tiling size and material combination may be used.

Colored surface example

It is desirable to be able to define surface properties of the geometry, such as color, that do not necessarily need to penetrate into the 3D body. Thus, a `<Color>` tag can be introduced at either the `<Object>` level, the `<Region>` level, or the `<Vertex>` level, in increasing order of precedence. Thus, the entire object or region can be made one color with a single tag, but certain vertices can be recolored individually. All RGB values must be real values ranging from 0 to 1. Also, it is possible to map images onto the surface of the part. This is accomplished by adding `<ColorMap>` or `<ColorFile>` tags with unique map ID numbers. Then vertices in the vertex list are associated with the desired map ID and pixel location within the image. This way, any triangle in which all three vertices have a `<VertexMap>` tag with the same ID will have that associated triangle of the image mapped onto it. Vertices may have more than one `<VertexMap>` tag, which can also refer to material distribution on the surface (with an associated depth) or a physical texture (bump map) that is to be mapped onto the surface. In the case of ambiguity (I.E. multiple color maps) the lowest map ID takes precedence.

```
<?xml version="1.0"?>
<AMF>
  <Object PrintID = "0" units = "mm">
    <Mesh>
      <ColorFile MapID="0">
        <File>Logo.bmp</File>
      </ColorFile>
      <Vertices>
        <Vertex VertexID="0">
          <VertexLocation x="0" y="1.332" z="3.715"/>
          <VertexMap UseMapID="0" MapXPixel="65"   MapYPixel="87"/>
        </Vertex>
        <Vertex VertexID="1">
          <VertexLocation x="0" y="1.269" z="3.715"/>
          <VertexMap UseMapID="0" MapXPixel="64" MapYPixel="87"/>
        </Vertex>
        <Vertex VertexID="2">
          <VertexLocation x="0" y="1.310" z="3.587"/>
          <VertexMap UseMapID="0" MapXPixel="32" MapYPixel="10"/>
        </Vertex>
        ...
      </Vertices>

      <Region FillMaterialID = "0">
        <Color R = "0" G = "0" B = "0.5"/>
        <Triangle V1 = "0" V2 = "1" V3 = "2"/>
        <Triangle V1 = "0" V2 = "1" V3 = "4"/>
        ...
      </Region>
    </Mesh>
  </Object>
</AMF>
```

(a) (b)

Figure 5: Surface properties such as color and texture may be added with several additional tags. Individual regions or vertices may be colored, or a bitmap can be mapped onto the 3D surface.

CSG (Computational Solid Geometry) Example

In addition to defining regions by a mesh, there are cases in which it is desirable to define regions via computational solid geometry, or a functional representation. From a complexity standpoint, it is not desirable to replicate all the features of the advanced CAD engines such as ACIS in the file format. However, regions may be defined using the `<FRep>` tag as an equation involving the three variable "X", "Y", and "Z", along with any number of unions, intersections, differences, and negations. By using binary operators "less than" and "greater than", complex regions may defined. By convention, wherever the equation evaluates to "true", geometry is present and wherever it is false, no geometry is present. If color is to be added, separate continuous `<RedEquation>`, `<GreenEquation>`, and `<BlueEquation>` tags should be added. The additional CDATA text in the file is a requirement of XML when storing arbitrary text, such as an equation.

```
<?xml version="1.0"?>
<AMF>
  <Object PrintID = "0" units = "mm">
    <FRep MaterialID = "0">
      <GeometryEquation>
        <![CDATA[X^2+Y^2+Z^2-4 <= 0 & Z >= 0]]>
      </GeometryEquation>
    </FRep>
  </Object>
</AMF>
```

Additional Capabilities and Extensions

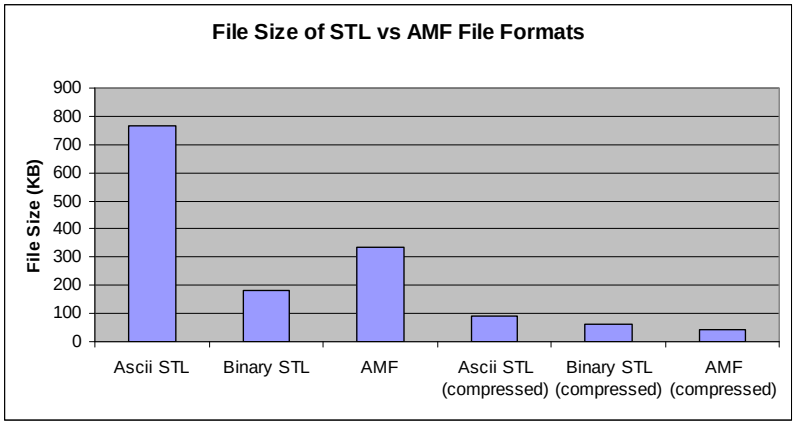
In addition to the <Mesh> and the <FRep> tags, a region can also be defined by the <Voxels> tag. Within the <Voxels> tag, a <Lattice> tag sets up the dimensions and packing type of the voxels, and a <Structure> tag defines a three dimensional matrix of material IDs to define the geometry and materials. The <Voxels> tag can also be used for a material in the palette to define complex, repeating microstructures of multiple materials.

Materials within the palette may refer to external material library files using the <File> tag. This allows vendors to maintain a database of existing materials, including useful material data, and for an AMF file to specify a specific material to be built with. Also, objects can include a <tolerance> to define the build tolerances that are required.

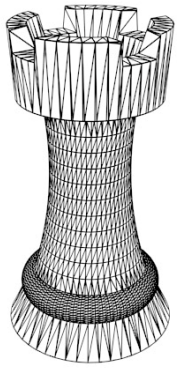
This paper is not intended to be an exhaustive tutorial on the AMF file format. A glossary of available tags is included in Appendix 1, along with a short description of each. Sample code and future updates are available at http://ccsl.mae.cornell.edu/AMF.

Performance

One disadvantage of the XML format is that the human readability comes at the cost of file size. In order to compare the AMF file sizes to the standard STL files, a sample mesh geometry of a rook was created with 3680 triangles. This geometry was saved as both ASCII and binary STL files. As expected, the binary STL exhibited a much smaller file size, or about 24% of the ASCII STL version (Figure 6). The XML text version of the AMF file was already 44% smaller than the ASCII STL file, and after applying standard compression routines, the AMF file was approximately 25% smaller than the binary STL. Compressing the binary STL yielded a file that was still 48% larger than the compressed AMF. Different types of compression routines may also be used for the XML (Ng et al., 2006), including those in which elements can be efficiently extracted without decompressing the entire file.



(a)



(b)

Figure 6: The size of files generated for both STL and AMF formats are shown in (a) for a rook geometry with 3680 triangles (b). The ASCII readable AMF file is 44% smaller than the equivalent ASCII STL, and after compression the AMF is 24% smaller than the binary STL and 67% smaller than the similarly compressed binary STL.

Conclusion

Here we proposed a replacement for the STL file format for use in additive manufacturing applications. We addressed the shortcomings of the STL file—namely leaks, lack of multi-material support, and no provisions for surface data—with a flexible XML-based format. This allows the file format to be extensible while maintaining compatibility with legacy applications. The AMF format is easily forwards and backwards compatible with STL files with a simple conversion, which will allow adoption driven by need and not by mandate as additive manufacturing hardware becomes more versatile. Additionally, the AMF format is easy to understand and human readable (before standard compression routines) which will allow for easy debugging and adoption by software developers. Ultimately, the success of a file format depends on its adoption by both manufacturers and users, and its acceptance as a formal standard.

Open source code for implementation of a parser and viewer for AMF files is available at http://ccsl.mae.cornell.edu/AMF

References

CHIU, W. K. & TAN, S. T. (2000) Multiple material objects: from CAD representation to data format for rapid prototyping. *Computer-Aided Design,* 32**,** 707-717.

GILMAN, C. R. & ROCK, S. J. (1995) The use of STEP to integrate design and solid freeform fabrication. *Solid Freeform Fabrication Symposium.* Austin, TX.

HAGUE, R. J. M. & REEVES, P. E. (2000) *Rapid Prototyping, Tooling and Manufacturing,* Rapra Technology Ltd.

JACOB, G. G. K., KAI, C. C. & MEI, T. (1999) Development of a new rapid prototyping interface. *Computers in Industry,* 39**,** 61-70.

JURRENS, K. K. (1999) Standards for the rapid prototyping industry. *Rapid Prototyping Journal,* 5**,** 169-178.

KUMAR, V. & DUTTA, D. (1997) An assessment of data formats for layered manufacturing. *Advances in Engineering Software,* 28**,** 151-164.

MCMAINS, S., SMITH, J. & SEQUIN, C. (2002) The evolution of a layered manufacturing interchange format. *ASME Design Engineering Technical Conferences.* Montreal, Quebec.

NG, W., LAM, W.-Y. & CHENG, J. (2006) Comparative Analysis of XML Compression Technologies. *World Wide Web,* 9**,** 5-33.

OBJET (2009) CADMatrix for Solidworks Brochure.

PATIL, L., DUTTA, D., BHATT, A. D., JURRENS, K., LYONS, K., PRATT, M. J. & SRIRAM, R. D. (2002) A proposed standards-based approach for representing heterogeneous objects for layered manufacturing. *Rapid Prototyping Journal,* 8**,** 134-146.

PRATT, M. J., BHATT, A. D., DUTTA, D., LYONS, K. W., PATIL, L. & SRIRAM, R. D. (2002) Progress towards an international standard for data transfer in rapid prototyping and layered manufacturing. *Computer-Aided Design,* 34**,** 1111-1121.

ROCK, S. J. & WOZNY, M. J. (1991) A Flexible File Format for Solid Freeform Fabrication. IN MARCUS, H. L. (Ed. *Solid Freeform Fabrication Symposium.* Austin, TX.

STROUD, I. & XIROUCHAKIS, P. C. (2000) STL and extensions. *Advances in Engineering Software,* 31**,** 83-95.

ZCORPORATION (2009) ZEdit Software.

Appendix 1

Summary of tags for the AMF file format:

Tag	Parent tag(s)	Attributes	Multi tags?	Description
<AMF>			No	Root XML tag
<Print>	<AMF>		No	Contains information about which objects/constellations to print
<Instance>	<Print>, <Constellation>		Yes	An instance of an object or constellation to print
		UsePrintID		The PrintID of the object or constellation to use
<Translate>	<Instance>		No	Translate the relative position of an object or constellation
		DX, DY, DZ		the distance of translation in x, y, and z directione
<Rotate>	<Instance>		Yes	Rotate the relative position of an object or constellation
		Ax, Ay, Az		The axis to rotate about
		Deg		How much to rotate about the defined axis, in degrees by the right-hand rule
<Constellation>	<AMF>		Yes	A collection of objects or constellations with specific relative locations
		PrintID		Assigns the PrintID for the constellation
<Object>	<AMF>		Yes	An object definition
		PrintID		Assigns the PrintID for the constellation
		Units		The units to be used. May be "IN", "MM", or "M" for inches, millimeters, and meters respectively.
<Tolerance>	<Object>, <Region>		No	Defines a desired tolerance
		XT, YT, ZT		The tolerance (in physical units) to be achieved if possible
<Color>	<Object>, <Region>, <Vertex>		No	The color to display the object in, and to print if supported
		R, G, B		Red, Green, and Blue values ranging from 0 to 1
<Mesh>	<Object>		Yes	A 3D mesh hull
<Vertices>	<Mesh>		No	The list of vertices to be used in defining triangles
<Vertex>	<Vertices>		Yes	A vertex to be referenced in triangles
		VertexID		Unique ID of this vertex.
<VertexLocation>	<Vertex>		No	The 3D location of this vertex
		X, Y, Z		3D coordinates in specified units of this vertex
<VertexMap>	<Vertex>		Yes	Maps this vertex onto the 2D coordinates of the specified bitmap
		UseMapID		Which map to use
		MapXPixel, MapYPixel		The coordinates within the map to map to this vertex
<Region>	<Mesh>		Yes	Defines a region from the established vertex list
		FillMaterialID		Which MaterialID from the Palette to use
<Triangle>	<Region>		Yes	Defines a 3D triangle from three vertices, according to the right-hand rule
		V1, V2, V3		VertexIDs of the desired vertices
<HeightFile>	<Mesh>		Yes	References an external heightmap to apply physical texture to a surface
		MapID		MapID for mapping this heightmap onto vertices
		Amplitude		Defines the height of the bump-map as it is applied to a surface.
<File>	<HeightFile>, <ColorFile>, <Material>		No	The location of the file to use

Tag	Parent tag(s)	Attributes	Multi tags?	Description
<HeightMap>	<Mesh>		Yes	An internally stored bump map for physical textures.
		MapID		MapID for mapping this heightmap onto vertices
		Amplitude		Defines the height of the bump-map as it is applied to a surface.
<XPixels>, <YPixels>	<HeightMap>, <MaterialMap>, <ColorMap>		No	Defines the size of the bitmap to be encoded
<PixelData>	<HeightMap>, <ColorMap>		No	The actual pixel data in order RGB from lowest to highest X, iterating from lowest to highest Y
<MaterialMap>	<Mesh>		Yes	An internally stored material map for the surface
		MapID		MapID for mapping this materialmap onto vertices
		Amplitude		Defines the depth of the material from the surface
<IndexData>	<MaterialMap>		No	Actual material index data in order from lowest to highest X, iterating from lowest to highest Y
<ColorFile>	<Mesh>		Yes	References an external file to be used as a color map.
		MapID		MapID for mapping this colormap onto vertices
<ColorMap>	<Mesh>		Yes	An internally stored color map for surface color
		MapID		MapID for mapping this colormap onto vertices
<FRep>	<Mesh>, <Material>		Yes	Defines a shape using computational solid geometry
		FillMaterialID		Which MaterialID from the Palette to use
<GeometryEquation>	<FRep>		No	Defines an equation using X, Y, and Z that evaluates to true for regions of material and false elsewhere
<RedEquation>, <GreenEquation>, <BlueEquation>	<FRep>		No	Defines a color to use from a real-valued equation using X, Y, and Z
<Voxels>	<Mesh>, <Material>		Yes	Defines a 3D bitmap of voxels, each associated with a material index
<Lattice>	<Voxels>		No	Defines the lattice to be used for the voxel bitmap.
<Lattice_Dim>	<Lattice>		No	The main dimension (distance) between voxels
<X_Dim_Adj>, <Y_Dim_Adj>, <Z_Dim_Adj>,	<Lattice>		No	Amount to adjust the LatticeDim in each dimension (0 to 1)
<X_Line_Offset>, <Y_Line_Offset>	<Lattice>		No	Amount to offset each line within each layer in X and Y dimensions
<X_Layer_Offset>, <Y_Layer_Offset>	<Lattice>		No	Amount to offset each subsequent layer
<Structure>	<Voxels>		No	Defines a 3D spatial arrangement of voxels
<X_Voxels>, <Y_Voxels>, <Z_Voxels>	<Structure>		No	The size (in voxels) of the specified region
<VoxelData>	<Structure>		No	Array of material indicies iterating through x, then y, then z
<Palette>	<AMF>		No	Contains the materials that make up the objects
<Material>	<Palette>		Yes	An available material
		MaterialID		A unique material ID
<Name>	<Material>		No	A descriptive name for the material