EXTENSIBLE DIGITAL FABRICATION LANGUAGE FOR DIGITAL FABRICATION PROCESSES

Jeffrey I Lipton[1], Karl Gluck[2], Hod Lipson[1,2]
Cornell Computational Synthesis Lab
[1]Department of Mechanical Engineering
[2]Department of Computer Science
Cornell University, Ithaca NY 14850

Reviewed, accepted September 23, 2010

Abstract

While additive manufacturing objects are described by the STL and AMF standards, the protocol controlling the fabricator is typically machine-specific. In this paper, we explore a system architecture that converts geometric data into control processes for equipment. We propose a new Extensible Digital Fabrication Language (XDFL) and an interpreted ToolScript language that describes how a geometry is translated into machine commands. An initial implementation of this system architecture was created and deployed as part of the Fab@Home project. The introduction of a standard process control language will decouple process planning from the equipment manufacturer, thereby catalyzing the introduction of new equipment and development of better process planners.

Introduction

Additive manufacturing has the potential to transform into a horizontal industry. Prior to the personal computer revolution, many companies were vertically integrated, designing all aspects of their technology, from processor to programs. The current state of the computer industry is horizontal, with different companies specializing in parts of the system. Currently in Solid Freeform Fabrication, most devices are vertically integrated, with a single company designing the materials through planning software. With defined standards for geometric and material interchange formats, SFF could become a horizontally integrated industry.

Currently the field of SFF lacks standards for machine commands, and for material and geometric processing. Any standard must address the needs of various communities. SFF is a rapidly growing and changing field with a variety of different techniques and technologies. In order for standards to be successful for a wide variety of current and future SFF techniques and technologies, it must address the following concerns:

(1) ***Technology Independence:*** Given the huge depth and breadth of additive manufacturing techniques, any machine command, and geometric processing standard must easily be able to be adapted to a wide variety of SFF technologies.
(2) ***Simplicity:*** Any machine command standard must be easy to implement and understand. Commands should be able to be read and debugged in a simple text editor. While this would limit compatibility with low power microprocessors, many SFF systems are computer controlled.
(3) ***Future compatibility***: SFF is a rapidly evolving industry, and any command medium and processing standard must be easily extensible. New features must be easily added as

warranted by advances in technology, while maintaining compatibility with previous versions.

Definitions:

In this paper, we will use the following definitions of terms:

Process – a technique used for SFF such as FDM, Stereo-lithography, Electron Beam Freeform Fabrication (EBF3), etc

Machine – a digital fabricator using a specific tool, eg Fab@Home with syringe tool, Fab@Home with valve based tool, Fab@Home with FDM tool, EBF3 machine, Makerbot, RepRap, etc

Material distribution – a geometry associated with a particular material

Background

For the past three decades manufacturing industries have used G-code for computer numerical controlled processes. G-code was designed for repeated subtractive manufacturing tasks. The language defines tool paths and common machine interactions. Many SFF systems have used this language to contain information about their vectorized paths. Industrial and commercial machines such as EBF3 machines and LENS machines use G-code only for pathing information. (1) (2) Low cost kits such as Makerbot and RepRap use G-code to contain information about material deposition, environmental parameters and machine parameters in addition to path information. (3)

G-Code has severe limitations on its ability to be a useful command medium in the future. G-code itself has no widely adopted standards or governing body. This has lead to many different companies developing unique standards for their particular machines. In the field of SFF some machines, though using the same technology, have used different G-code language dialects. A Makerbot machine uses M-codes to start and stop an extruder while with RepRap machines the extrusion is treated like an axis of motion in the movement commands. It is difficult to extend G-code. Since each command is distinct and numbered, modifying the function of a command would require generating a new numbered command, or breaking backwards compatibility. G-code is designed explicitly for vectorized interaction and could not easily be used for other types of existing technologies. Additionally it is inherently mono-material, limiting its future usefulness of additive manufacturing.

Fab@Home robo-casting systems use a customized XML language called “fab” files in order to contain vectorized path information. (4) It is an multi-mateiral language, capable of describing a build process of n-number of materials. The language itself is flexible but is designed explicitly for vectorized printing using a syringe system. While it is possible to adapt the format for various other deposition heads, it is not a natural process. (5)

The SFF industry has used STL as a standard geometry format, and is adopting the new AMF format for geometric and material distribution information. (6) However, for the majority of SFF systems, the processing of material and geometric information is hard coded into print planning programs. This makes specifying unique printing requirements for a given material or geometry difficult. The ability to script the print planning process could provide an easy way to extend the functionality of SFF systems.

In finding a solution to the standards model, the web-browser was examined. Two standards are the core of a modern web browser are HTML and JavaScript. HTML is human and machine readable, and platform independent. It uses tags to clearly separate data and meta-data. JavaScript is the standard scripting language of web interactions. A variety of engines can securely execute embedded JavaScript code. (7)

Specifications: Overview

The architecture of the proposed system depends on two critical components, the Extensible Digital Fabrication Language (XDFL), and the ToolScript standards. XDFL is an XML-compliant command medium. ToolScript is an extension of the EMCAScript (JavaScript) language. It allows users to script the processing of geometric and material information. ToolScript processes a material distribution, such as an AMF file, along with materials settings, into XDFL. ToolScript is process-specific, and XDFL is process-specific and materials-specific.

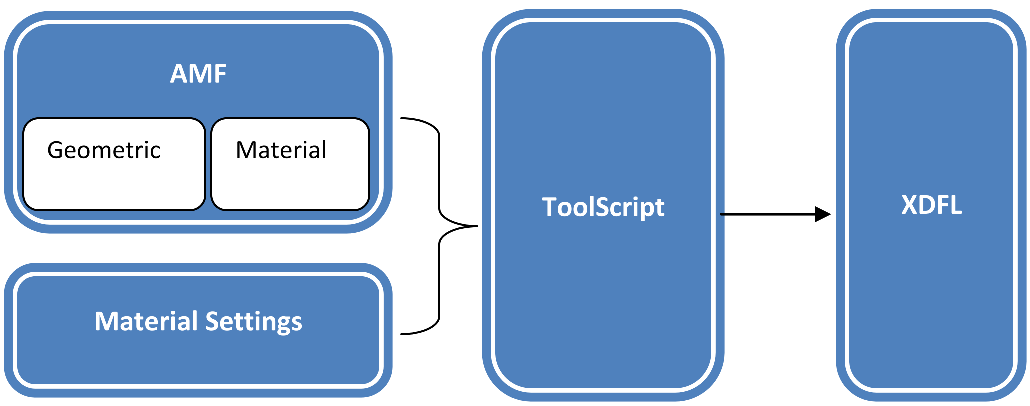


Figure 1: ToolScript processes the materials distribution information from an AMF into XDFL commands

Specification: ToolScript

The ToolScript for a given process has three primary functions. Firstly, it must slice and object into slices of given thicknesses. The height of these slices may be constant or vary based on a given geometry. Secondly, it must process the slices into deposition commands. A vectorized process requires the processing of slices into paths. A serial voxelized a process requires the processing of slices into voxels sorted in the order of deposition. A parallel voxelized process requires the processing of slices into bitmaps of regions inside of each slice. Finally, the ToolScript must generate the XDFL and write it to a file. In order to accomplish this, a variety of objects are required by a ToolScript.

Any given ToolScript file may only need a subset of the objects described here. A ToolScript needs to have an AMF file representation object, which contains AMF region objects. The AMF regions contain the geometric information for a given material. A slicer object must be able to take AMF regions and slice heights, and convert them into a slice object. A slice object must contain a single outer boundary and zero or more inner boundaries. Each slice has a single material and z height associated with them. Vectorized processes require a pather object, which can convert a slice and information about the process into paths. Each path is a list of special coordinates with an associated material. Voxelized processes require a voxelizer object similar to a pather, which outputs voxel information. Finally, an XDFL writer object is required which can convert representations of voxels, paths, and slices into XDFL code.

Using ToolScript it is possible to write custom implementations of the various objects. For example, if one needs a unique path planner for a specific application, it could be written as part of a ToolScript, provided it interacts with the rest of the tool chain as described. It is also possible to use multiple types of pathers for the same object. This would allow one to make an object with a solid top and bottom, and a hollow core, making it air-tight. (8)

Object	Inputs	Outputs	Properties	Purpose
AMF File			AMF regions	Contains AMF regions
AMF region			Material, (slices), (paths), (voxels), (bitmaps)	Contains material specific geometry
Slicer	AMF region, slice heights	Slices		Converts an AMF region into slices
Slice			Material, Z value, (Paths), (voxels), (bitmaps)	Contains outer and inner boundaries
Pather	Slice, path information	paths		Converts slices into paths for vector processes
Path			Material, points, (Speed), (Cross section)	A representation of a path to be taken
Voxelizer	Slice, voxelization information	Voxels bitmaps		Converts slices into voxels or bitmaps
Voxel			Material, Point, (Shape)	A representation of a voxel to be deposited
Bitmap			Material key, Location	A representation of a voxel region to be deposited in parallel
Material Calibration			Material properties	An array of material properties and corresponding values
XDFL writer	Material calibrations, SFF Process information (paths), (voxels), (slices),(bitmaps), (AMF regions)	XDFL file		An object which converts the information generated from the pathers, slicers, and voxelizers into XDFL commands

Figure 2: Objects required by ToolScript. A single ToolScript file may only need a subset. Optional items are in parenthesis.

Specification: XDFL

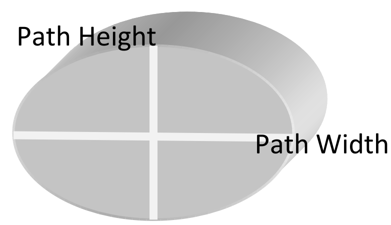
XDFL has two top-level tags, which denote the different types of information it contains. The palette tag contains information about the materials used in the printing process. The information contained within the palette tag must be globally accessible when using the XDFL file to execute a print. It contains the abstract information about the different materials to be used in the print process. The commands tag contains a sequential list of commands for the digital fabricator. The commands listed under the command tag may refer to the global information in the palette tag and may locally overwrite them. A value specified in the commands section is valid for any of its parent tag's children. Unlike the command medium discussed above, XDFL has a built-in knowledge and description of the print's volume. It is designed to be useful for vectorized, voxelized, and stratified processes. A list of all of the XDFL tags and their relationships is in appendix A.

XDFL: Vectorized

A material in a XDFL file for a vector process has five required tags: path height, width, area constant, speed, and compression volume. These allow the XDFL to be created without having explicit knowledge of how the machine works; rather, it only requires knowledge of how it will deposit material. The flow rate of a path is defined below. For any vectorized process, either the flow rate or the path speed can be fixed for a given resolution (width/height values). The compression volume defines how much material is deposited at the beginning and end of a path. A positive value denotes a process over-depositing initially. A negative value denotes a process under-depositing initially.

$$\frac{Vol}{Sec} = PathWidth * PathHieght * AreaConstant * PathSpeed$$

$$Cross\ Section = AreaConstant * PathWidth * Path\ Height$$



(a) (b)

Figure 3: The equations governing XDFL path flow rates (a), and a diagram of a given path (b)

Paths contain a list of points defining line segments. Each point can contain between two and six coordinates. If paths are contained in a layer tag, then they could have a default value for "x", "y", or "z" provided by the layer. Coordinates "u", "v", and "w" define rotations around the "x", "y", and "z" axis respectively. If a machine has less than the provided number of axes, information contained in the tags is ignored. Appendix B contains an example G-Code file and its XDFL equivalent.

XDFL: Stratified

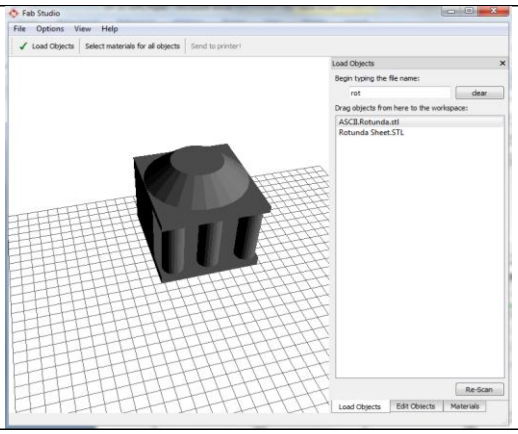
XDFL can be used to define purely stratified SFF processes such as laminated object manufacturing. In order to describe a stratified print, the materials would contain process specific properties and values, and the commands section would contain exclusively layer tags. These layer tags link to image files of the current layer. The properties of the materials could be used to map between the materials and the image files. The XDFL files and image files could be placed in a zip archive with a unique extension for laminated processes. This would ensure that the file completely described the process. An example XDFL file for stratified processes is in Appendix C.

XDFL: Voxelized

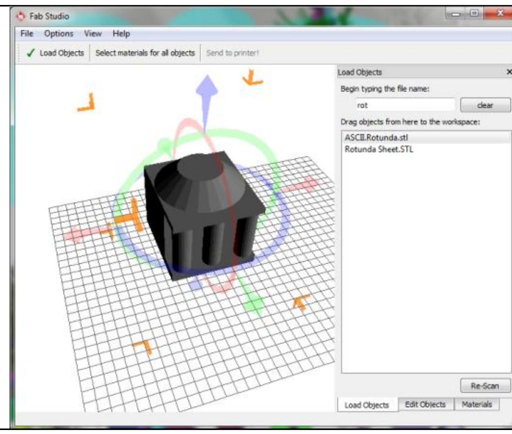
XDFL files work slightly differently for serial and parallel voxel machines. A serial voxelized XDFL file would be similar to the stratified and vectorized files, but would contain a sequential series of voxel tags. Each voxel tag would define its location in space. It optionally would contain a geometry attribute, which references, and STL or AMF file for visualizations purposes. A vectorized XDFL file may use the voxel tag to deposit a given volume at a given location. A parallel voxel machine would use path tags to move to a location, and then use the bitmap tag to define where space of voxels will be deposited. An example file is in Appendix D.

Implementation and Performance

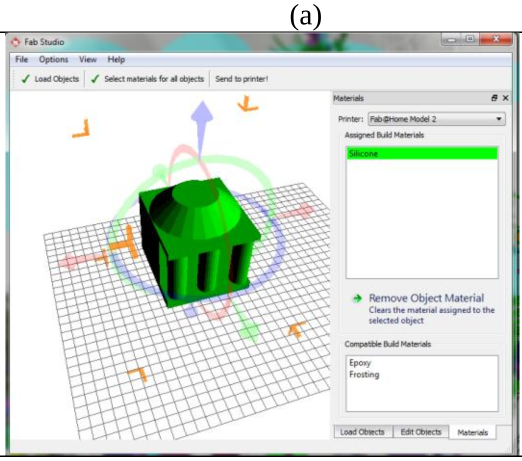
In order to test the performance of ToolScript a processing library and environment were necessary. A digital fabrication application library was created, called libFabApp. LibFabApp contains all of the code needed to run ToolScript and process AMF and STL files. The library processes tool files that have the process specific material settings and ToolScript embedded within. Using libFabApp, FabStudio version one was created to provide a GUI interface for interacting with the library. The first iteration of the library and studio is designed to work with vectorized print processes. Later versions will allow for voxelized and stratified prints.



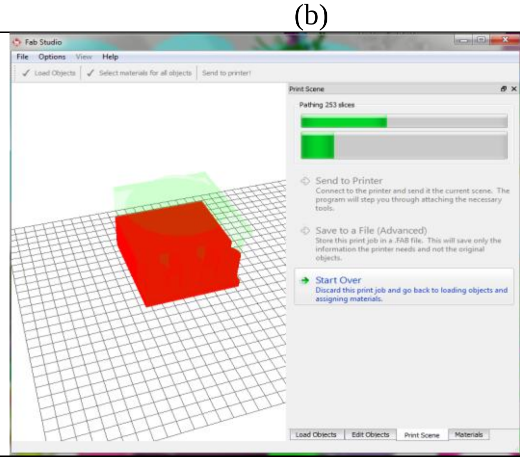
(a)



(b)



(c)



(d)

Figure 4: In order to use the new framework in FabStudio a user must load an object (a), position and scale it (b) assign a material to STL files or unmatched AMF files (c), and send to printer (d)

We used FabStudio to generate XDFL files for robo-casting processes. By combining the XDFL with a digital fabricator's configuration information, it is possible to generate the G-Code needed to operate the machine. A custom python script converted the XDFL file into G-code for a Makerbot and RepRap. The G-code from the script was compared to the XDFLs size in a zipped and unzipped form. Figure six shows that the XDFL is larger than G-code files when uncompressed, but can be reliably compressed compared to the un-standardized G-Code.

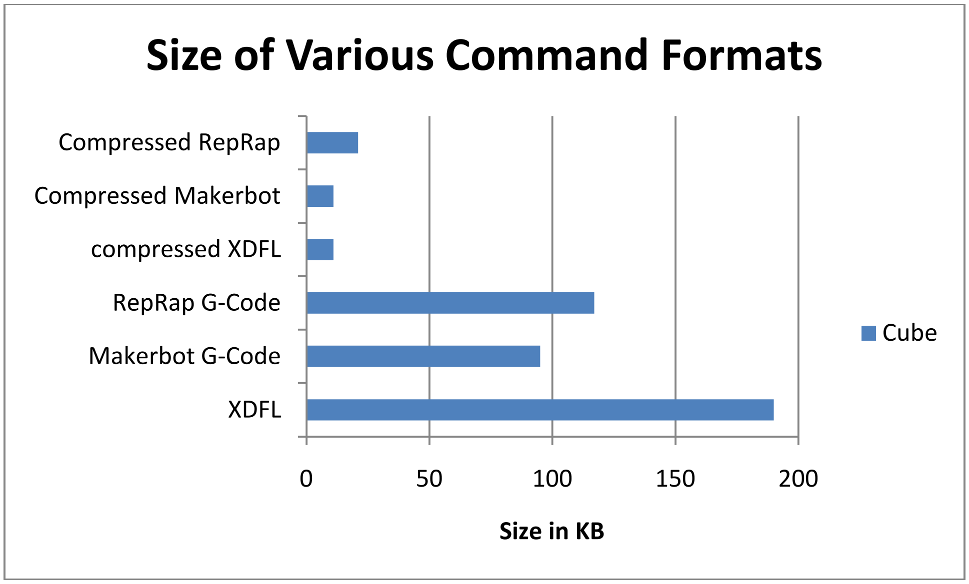


Figure 6: A comparison of XDFL verses two standard G-Code encodings for FDM

Future Work:

Non-vectorized ToolScript objects need to be added to libFabApp. Implementations of the non-vectorized XDFL should be tested on various platforms. There are several useful features which should be built on top of the XDFL-ToolScript framework. The simplest one to implement is the embedding of JavaScript into XDFL files. This would allow material data to be calculated dynamically. Complex curves could be approximated at machine resolution at runtime. Based on the history of websites embedding JavaScript in HTML this could have a variety of benefits. Following embedded JavaScript in XDFL, a hybrid DOM/SAX model with printing related events would allow a system to perform closed loop SFF.

Conclusions:

XDFL is a unique command medium, since it is applicable to a wide variety of SFF technologies. ToolScript provides a uniform means of programming geometric processing for SFF technologies. ToolScript and XDFL represent a powerful platform. Its full potential can only be realized if it is refined and adopted across systems and user groups. By standardizing the processing of geometries and the command mediums of SFF systems, SFF technology could rapidly develop by allowing effort to specialize and the industry to become more horizontal.

Acknowledgements:

The authors of this paper would like to thank the members of the Cornel University Fab@Home Student project team.

Bibliography

1. *A Design of Experiments Approach Defining the Relationships Betwen Processing and Microstructure for Ti-6Al-4V.* **Wallave, T A, et al.** Austin Tx : 15th Solid Freeform FAbrication Symposium, 2004.

2. *Free Form Fabrication ofMetallic Components using Laser Engineered Net Shaping (LENS).* **Griffith, M L, et al.** Austin TX : Proceedings of the Solid Freeform Fabrication Symposium, 1996.

3. **RepRap Project.** Mendel User Manual: RepRap GCodes. *reprap.org.* [Online] reprap research foundation, June 28, 2010. [Cited: June 29th, 2010.] reprap.org/wiki/Mendel_User_Manual:_RepRapGCodes.

4. *Fab@Home Model 2: Towards Ubiquitous Personal Fabrication Devices.* **Lipton, Jeffrey I, et al.** Austin TX : s.n., 2009. Solid Freeform Fabrciation Symposium.

5. **Fab@Home Project.** Fab@Home:Deposition Tools. *Fab@Home Wiki.* [Online] Fab@Home Project, March 17, 2010. [Cited: June 20, 2010.] fabathome.org/wiki/index.php/Fab%40Home:Deposition_Tools.

6. *STL 2.0: A Proposal for a Universal Multi-Material Additive Manufactuing File Format.* **Hiller, Jonathan D and Lipson, Hod.** Austin Tx : 20th Annual Solid Freeform Fabrication Symposium, 2009.

7. **ECMA International.** *ECMAScript for XML(E4X) specification.* Geneva : Ecma International, 2005. ECMA-357.

8. *Additive Manufacturing of Pneumatic Actuators and Mechanisms.* **Lipton, Jeffrey I, Hiller, Johnathan and Lipson, Hod.** Austin Tx : 21st Annual Solid Freeform Fabrication Symposium, 2010.

Appendix A: List of XDFL tags

Tag	Descriptor	attributes	parents	Comment
Xdfl	Top level tag for the file	process, version(#)		
palette	Header for holding material information		xdfl	
material	Opens material		palette	
name	Name		Material, property	Not unique locally
id	Locally unique ID of a material		Material	Integer value.
pathWidth	Width of vector path	units	Material	Required for vectorized
pathHeight	Height of vector path	units	Material	Required for vectorized
pathSpeed	Speed of vector path	units	Material	
areaConstant	Area constant of vector path	units	Material	Required for vectorized
compression	Compression volume of material	units	Material	Required for vectorized
property	Dynamically defined property of material	id	Material	Optional
value	Value of a property	units	Property	
commands	Body of file which holds commands		Xdfl	
layer	Defines a single layer	image	Commands, div	Optional, but recommended. Image = URL of an image of the layer slice
Div	Lable for sections	Id, title	Commands, Layers,	Used organize into outlines infill or any other organizational structure
Path	Opens a vector path	crossSectioncoordinates (abs/rel), units, objectId	Commands, Layer div	For machine movements or vectorized
materialID	Locally unique id of material to be deposited over path		Path, voxel	Presence denotes if a path is deposition or movement
speed	Overwriting speed for a path	Units	path	Optional
point	Opens a point		path	
x	x coordinate		Layer, Point, voxel	
y	Y coordinate		Layer, Point, voxel	
z	Z coordinate		Layer, Point. voxel	Optional if in layer
u	Rotation about x	Units(r/d)	Point,voxel	Optional
v	Rotation about y	Units(r/d	Point,voxel	Optional
w	Rotation about z	Units(r/d)	Point.voxel	Optional
voxel	Defines a volume to be deposited	Geometry coordinates objectId	Commands, layer, div	For voxel deposition or extruding a volume at a fixed location
volume	Defines volume of a voxel	units	Voxel	
script	Opens a script to be run	type	All	Can be embedded anywhere to provide scripting for a given tag
noscript	Provides default values if a script cannot be run		All	
bitmap	Contains link to bitmap for parallel voxel deposition	Src objectId	Layer, commands	Links to a bitmap representing a parallel voxel field
dwell		units	commands	Pauses for a given amount of time
pause	Pauses till a user responds		commands	

Appendix B: Vectorized XDFL file Example

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<xdfl>
<palette>
        <material >
                <name>silicone</name>
                <id>0<id>
                <pathWidth>2</pathWidth>
                <pathHeight>4</pathHeight>
                <pathSpeed units="mm/s">3</pathSpeed>
                <areaConstant units="">1<areaConstant>
                <Compression units="mm^3">10<Compression>
        </material>
</palette>
<commands>
        <path>
                <materialID>0</materialID>
                <point >
                        <x>1</x>
                        <y>2</y>
                        <z>3</z>
                </point>
                <point >
                        <x>2</x>
                        <y>2</y>
                        <z>3</z>
                </point>
                .......
        </path>
        <path>
                ......
        </path>
        ...
</commands>
</xdfl>
```

Appendix C: Stratified XDFL File


```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<xdfl>
<palette>
        <material>
                <id>0</id>
                <name> A1 Paper</name>
                <property>
                        <name> cure time<name>
                        <value units="s">.1</value>
                <property>
                        <name> Layer thickness <name>
                        <value units="mm"> .010</value>
        </material>
</palette>
<commands>
        <layer id="0"  image="layer0.svg">
                <materialID>0</materialID>
                <z>0</z>
        </layer>
        <layer id="1"  image = "layer1.svg">
                <materialID>0</materialID>
                <z>0.01</z>
        </layer>
        ...
</commands>
</xdfl>
```

Appendix D: Voxelized XDFL file

```
<xdfl>
<palette>
	<material>
		<id>0</id>
		<name>steel</name>
		<property>
			<name>Cure time</name>
			<value units = "seconds">0.01</value>
		</property>
	</material>
</palette>
<commands>
	<voxel >
		<materialID>steel</materialID>
		<volume units = "mm^3">2</volume>
		<x>1</x>
		<y>2</y>
		<z>2</z>
	</voxel>
	<voxel >
		<materialID>steel</materialID>
		<volume units = "mm^3">2</volume>
		<x>1</x>
		<y>2</y>
		<z>4</z>
	</voxel>
	...
</commands>
</xdfl>
```