Methods of Preproduction for Material Layout in a 3D Printing Pipeline.

Eric Greenberg

CSU Stanislaus

CS 4960, Spring 2015

Dr. Melanie J. Martin

4-3-2015

Methods of Preproduction for Material Layout in a 3D Printing Pipeline

Abstract

As more uses are discovered for 3D printers, it is becoming important to design software to control the way materials are handled. This paper will look at two ways to control material layout in the preproduction phase of a 3D printing pipeline. First, it will explore a method designed by Disney Research Labs for hollowing the internal space of a model in a non-programmable 3D pipeline. Second, it will explore the fablet process of a programmable 3D pipeline known as OpenFab.

Introduction

3D printers are a technology with vast potential in almost any field that involves prototyping, sculpting, or any form of production. At the industry level, medical researchers, car manufacturers, prop designers and many other professionals have found 3D printers useful in everything from car manufacturing to creating cell structures. As the prices of these printers have dropped to affordable levels, hobbyists have begun purchasing, building, and experimenting with personal 3D printers. These hobbyists have started to make a variety of objects, such as action figures, models, movie prop replicas, replacement parts for appliances, functional drones, functional weapons and artificial

limbs. These systems have allowed a vast array of digital models to become physical models in the homes of many enthusiasts.

Unfortunately, simply taking a model designed in software and sending it to a 3D printer can result in unforeseen difficulties. Some of the difficulties encountered in the 3D printing process include time constraints, costs, and stability.

Printing a 3D structure can be time intensive. A recent study showed a sample model taking anywhere from 3 hours to 11.5 hours on commercial printers. (Stultz, M. 2015) Cost of materials can be expensive. In 2013, the material costs per cubic centimeter could range from $1.40 USD for flexible plastic to $20 USD for silver. (Wang, W, 2013)

The structures that are modeled in a virtual environment may not be physically stable in the real world, resulting in structures that are too brittle or weak to withstand gravity, pressure or use. (Stava, O., et al. 2012)

Fortunately, there are methods to program solutions to address these issues. One method is to use Goal-Based Material Design solutions, which generally involve software that takes models and manipulates them to meet certain specifications or constraints. (Vidimče, K. et al. 2013) These specifications could be as simple as assessing the printability of a 3D model before sending it to be printed (Telea, A., & Dalba, A. 2001) or as complex as building custom internal frame structures to add durability while reducing wasted materials. (Wang, W., et al. 2013) These solutions can be self-contained and automatic. (Vidimče, K. et al. 2013) However, problems that require a professional's oversight, an artist's aesthetics, or fine tuning from the printer's operator can be difficult to solve when these solutions are automated. Worse still, not all of these Goal-Based Material Design solutions are transferable between models, meaning that any custom

adjustments made by the operator may have to be repeated on a per-model basis. To address this, research is being done on developing programmable 3D printer pipelines. A programmable 3D printer pipeline is a solution that could allow for adjustments to be saved and transferred between models, allowing for future automation. Potentially, a 3D printer pipeline could also lower computational strain as well as reduce print time, depending on the design implementation. (Vidimče, K. et al. 2013)

This paper will explore an example of how each of these methods of preproduction material layout systems can control the use of material. This paper will look at a Goal-Based Material Design solution for hollowing a structure, as well as how a programmable 3D printer pipeline called as OpenFab can handle the layout of material using a method known as fablets.

Definitions

Some definitions are needed before it is possible to discuss the 3D printer pipeline. Because designing custom objects to be produced by 3D printers begins at a digital modeling phase, it's possible to implement some material handling during the digital design phase (Stava, O. et al. 2012) — which means it's necessary to have an understanding of some basic computer graphics terminology before discussing the manipulation of models.

Pixels

In 2D graphics, the smallest unit of measurement is a picture element, abbreviated as “pixel”. (Shirley, P. et al. 2005) On most modern monitors, pixels are represented by the three lights that produce the wavelengths needed by the human eye to see color: red, green, and blue (RGB). These groupings of three lights are the physical manifestation of a pixel. When different intensities of these lights are used, different colors are perceived by humans. Traditionally, there are 256 steps between the least intense output of light to the most intense output of light.

The resolution of monitors in America is measured by pixels per inch (ppi). Pixels are almost always placed evenly on a rectangular grid. However, in a computer system a pixel is a bit different.

While the physical pixel only has three lights at various levels of intensity and a physical location, a digital representation of an image is considerably more complex. The simplest form is a bitmap image, but even then a pixel can have more information stored in it. While this information is dependent on the file structure and image mode, many pixels will have four channels, three for dictating the intensity of red, green and blue light, and one “Alpha” channel. This Alpha channel is used to describe transparency – the higher the value, the more “transparent” a pixel is. This is useful when multiple pixels are rendered over one another; the transparency caused by the Alpha channel allows pixels “below” the transparent pixels to show through.

Quadtrees

For this paper, you will need to understand how octrees are used in conjunction with 3D rendering, and to understand octrees you must first understand quadtrees and how they are used in 2D rendering.

As stated above, 2D graphics can utilize pixels, and each pixel's position needs to be stored. While there are simple methods for doing this, such as two-dimensional arrays, there are more flexible solutions that use tree structures. One common tree for encoding images is the quadtree (Karrels, E. 1999), which is a tree structure in which every parent has four children. A quadtree is mapped to an image in a manner similar to the geometric solution for Zeno's Paradox. The entire image is set to equal 1, and is linked to the root. When you give the root its four children, the image is subdivided into four quadrants, where each child represents one of the quadrants. Each time a node is given four children, the corresponding quadrant is further subdivided. (Demmel, J. 1996)

Changes to individual pixels, or connected areas of pixels, can be represented by adding the appropriate data to the corresponding nodes in the quadtree. This can allow areas of duplicate data to automatically be discarded. When a quadrant or sub-quadrant is composed of identical data, only the highest node (largest quadrant) needs to be stored. (Spatial Partitioning and Indexing: Regular Decomposition: Quadtrees. 2010)

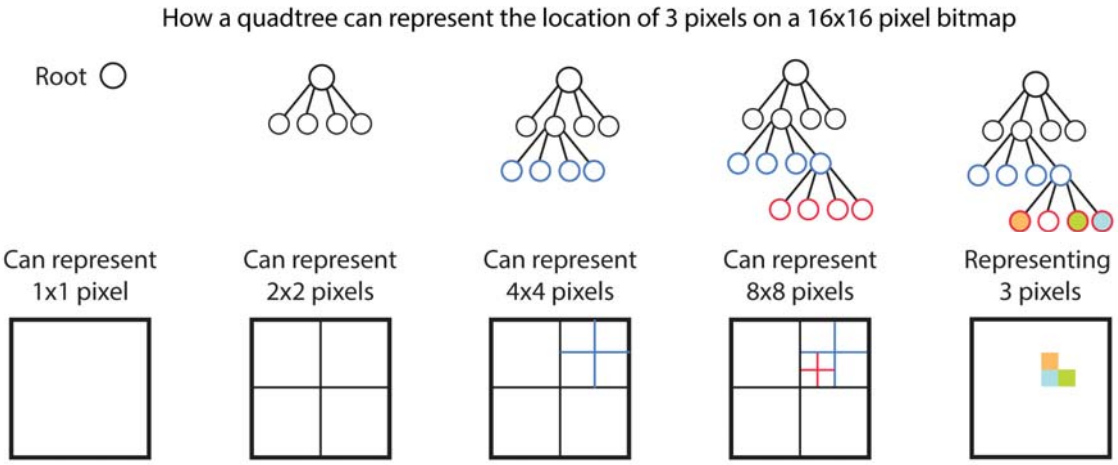


Diagram by Eric Greenberg; based on work by Demmel, J. 1996, Johnsonz, N. 2009, Karrels, E. 1999, and Spatial Partitioning and Indexing: Regular Decomposition: Quadtrees. 2010.

Vectors

In computer graphics, a vector is a description of direction and length. (Shirley, P. 2005) In 3D modeling, vectors are used to describe points, positions, offsets and other spatial relations. They are a standard way to model objects in most 3D systems, and can be used in conjunction with transformation matrices to describe and perform transformation processes on 3D structures.

Voxels

While a two-dimensional image only needs pixels to provide information about light intensity (and maybe transparency), three-dimensional images need considerably more information. Because 3D models are expected to be viewable from different angles, under different light sources and appear to be made out of different materials that interact in a variety of ways, considerably more data (such as diffusion, reflection, refraction, etc.) needs to be stored.

A volume element (abbreviated voxel) is a way of storing and describing the "material" a 3D model is "made" out of. (Angel, E. 2008) Essentially, it's the description of an object's volume behavior and texture design. This allows for complex interactive structures, like glass or stone, to be programmed independently of a model's shape. Additionally, this separation of shape (described with vectors) and volume (described with voxels) allows for textures and volumes to be reused in different models, saving programming time.

Because 3D printers produce a volumetric material, it's not uncommon to measure their output resolution in voxels per cubic inch or centimeter. (Fang, S. (n.d.); Crassin, C., & Green, S. 2012) Once a resolution is set, an entire physical structure can be described by voxels alone. Because of this, it is possible to digitally model structures using only a discrete set of voxels. In some instances, it's possible to dynamically adjust the resolution (size) of voxels over a discrete space by using octrees. (Bächer, M., et al. 2014)

Shaders

"*A shader procedurally defines the appearance of an object to be rendered in computer graphics*." (Vidimče, K. et al. 2013) While a voxel describes how a material behaves, and in doing so can indirectly describe the surface appearance, a shader deals with the texture (or image) applied to the material, thereby directly affecting the final render.

Octrees

A quadtree is capable of creating a reduced position map in 2D space. However, an octree is capable of creating a reduced position map in 3D space. (Demmel, J. 1996). Rather than taking a square, mapping it to the root of a tree and then subdividing it into four squares for each level that is descended, a cube is taken, mapped to the root of a tree and then subdivided into four cubes for each level that is descended.

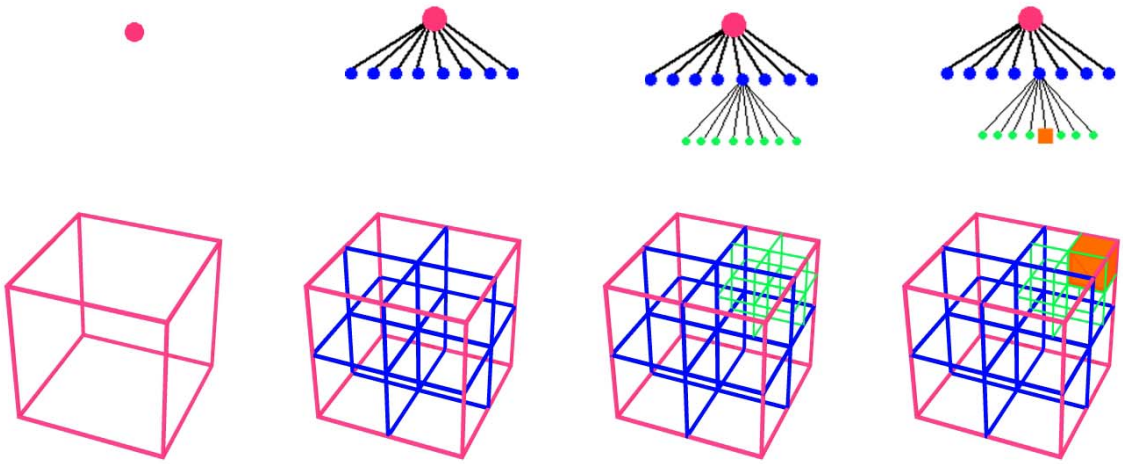


Diagram by Eric Greenberg; based on work by Demmel, J. 1996. As the octree gains depth, more voxels are created through division, resulting in an increased resolution throughout the finite volume of the cube.

3D Pipelines

"3D pipeline" is the general term for whatever system of software and hardware is in place to convert digital models into projectable images.

A fixed-function 3D pipeline is a 3D pipeline that has an unchangeable set of provided ways of handling data. (Zhao, Y. 2008a; Zhao, Y. 2008b)

A programmable 3D pipeline is one that allows for control over some or all of the steps taken in the pipeline. (Wei, L. 2005) This means that it is possible to refine and customize the pipeline for specific tasks, allowing for greater speed in the rendering process and more complex results.

Goal-Based Material Design solution

First we will look at a Goal-Based Material Design solution.

In 2014 Disney Research Labs designed a series of algorithms that could take a 3D model and, with minimal visible distortion, print a top that would balance on a fixed point when spun. The process that they created could be considered analogous to a fixed-function 3D pipeline, as it had a set of unchangeable pre-programmed ways of manipulating the data that was sent to it in order to produce a specific result.

The system has the single goal of creating a top from a model with minimal distortion, and it achieved this goal through a series of Goal-Based Material Design solutions. One of these solutions is a hollowing process that reduces the amount of material used during the printing process. The researchers designed this in order to control the weight of the tops being produced, but similar algorithms are designed and used to reduce material waste, thereby reducing cost. What makes the Disney Research Labs' hollowing solution interesting is that it is capable of adjustable print resolution and it is able to preserve sections of material within the hollowed area.

While considerable preprocessing is done earlier in this "pipeline" before the data is passed to the hollowing process, much of it consists of solving physics equations related to spinning tops and is beyond the scope of this paper. For brevity, any necessary information will be included in brief, as needed.

The hollowing process is as follows:

First, a model is passed from a previous algorithm, where the desired void areas have been calculated. At this point, the model has been treated as a region Ω, and has broken down into two parts: the skin of a specified thickness called the boundary shell or Ω_b, and the interior area of the model called Ω_i.

Additionally, the set of all material to be kept has been set during a previous process where the internal volume is initially calculated. This is done by marking off the areas to be voided, which is a subset of the interior Ω_i referred to as Ω' where $\Omega' \subseteq \Omega_i$

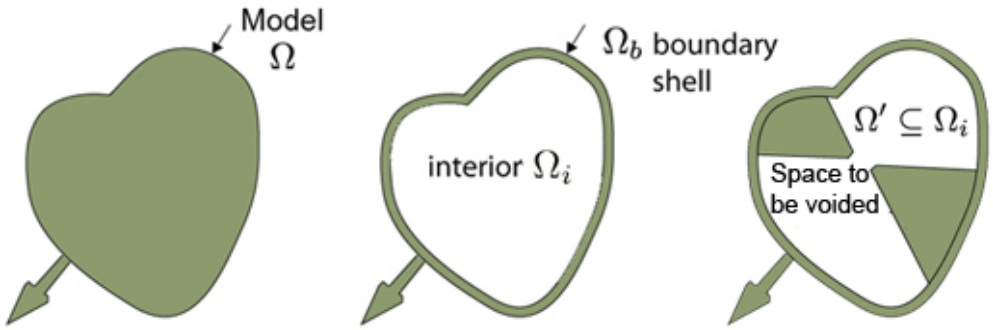


Diagram modified from diagrams in *Spin-It: Optimizing Moment of Inertia for Spinnable Objects* (Bächer, M. et al. 2014)

With this data preprocessing done, the hollowing process can start. To accomplish this, the adjusted volume, referred to as $S_{\Omega\text{-}\Omega'}$, is reduced into a binary fill variable β_k where $\beta_k \in \{0,1\}$. In other words, at any given location in the model the corresponding β_k will either be 0 (an area filled with a printed material) or 1 (a void, or an area not filled with a printed material). During this reduction process, it becomes clear that the size of the voxels can be varied. Where there is a large empty space, there doesn't need to be a high resolution detail of the area inside of the space. In fact, the only time that needs to be a high resolution of voxels is when an edge is encountered. (Bächer, M. et al. 2014) This

means that the interior can be represented by octree cells as shown below.

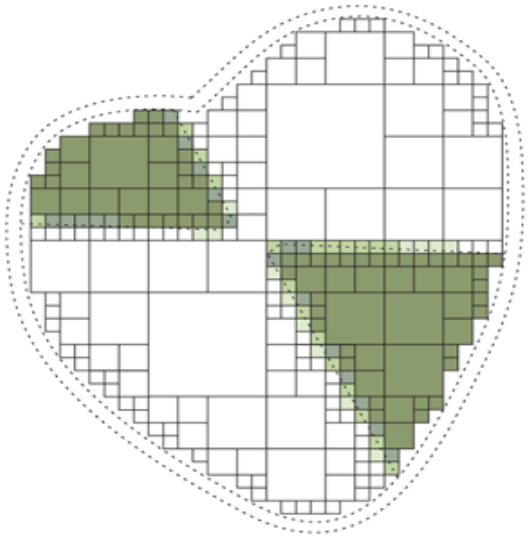


Diagram from *Spin-It: Optimizing Moment of Inertia for Spinnable Objects* (Bächer, M. et al. 2014)

To create this reduction, Disney Research Labs designed the following equation:

$$\text{Volume} = S_{\Omega} - S_{\Omega'} = S_{\Omega} - \sum_{k} \beta_k S_{\Omega_k}$$

where $\Omega_i = \cup_k S_{\Omega}$ is a partitioning of the interior into octree cells Ω_k.

The void space Ω' would consist of all cells Ω_k where $\beta_k = 1$ (Bächer, M. et al. 2014)

In this initial run, the resolution is set at a midpoint between the lowest and the highest possible resolutions the 3D printer is capable of. From this point it is possible to raise or lower the resolution as necessary.

Note that the only time a non-one or none-zero could be detected is on a boundary. This is searched for by simply checking each cell for a non-binary result. If one is found, it is resolved by increasing the resolution of the voxels in that octree node until either the fractional area is gone, or until the highest resolution supported by the 3D

printer is reached. If the latter occurs, then the binary numbers can be rounded to either a one or a zero. (Bächer, M. et al. 2014)

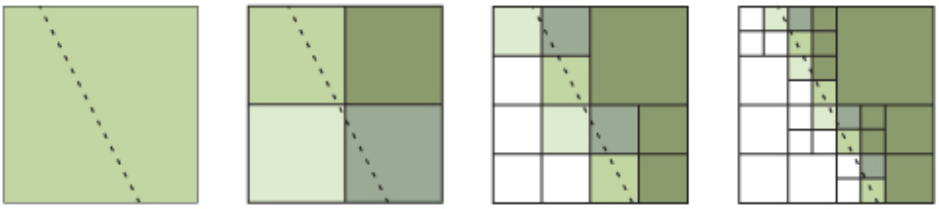


Diagram from *Spin-It: Optimizing Moment of Inertia for Spinnable Objects* showing the process of increasing the voxel resolution to refine an edge. (Bächer, M. et al. 2014)

Once the resolution has been raised as high as necessary in areas that require detail, the areas that do not require detail can be lowered. This is done easily by selecting a cell then searching its neighboring cells to see if the binary values are identical. If they are, the cells are merged. Once the cells are at the lowest resolution the 3D printer supports, the process is terminated. (Bächer, M. et al. 2014)

The last consideration of the voiding process is how to handle the cells Ω_k that overlap with the boundary shell Ω_b. The solution implemented was to simply treat the cells touching and overlapping with the boundary as a sub-tree where the boundary must be filled ($\beta_k = 0$) at the highest resolution the 3D printer can print. (Bächer, M. et al. 2014)

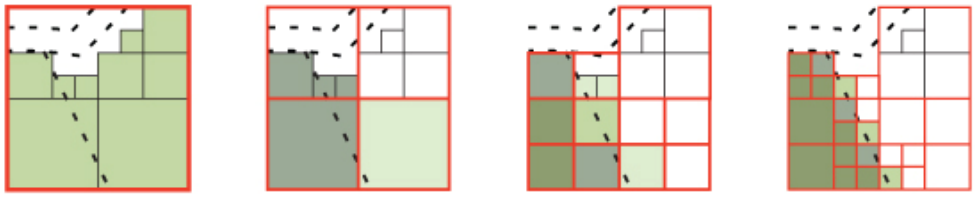


Diagram from *Spin-It: Optimizing Moment of Inertia for Spinnable Objects.* This diagram shows the process of increasing the voxel resolution to refine an edge. “Cells corresponding to fill variables are marked in red, their subtrees in black ” (Bächer, M. et al. 2014)

At this point, the Goal-Based Material Design solution for hollowing is complete, and the data is sent further down the pipeline for voxelization using an octree-cage extraction technique that is beyond the scope of this paper.

Programmable 3D Pipeline Solutions

Now that a Goal-Based Material Design solution for handling the materials in a non-programmable 3D pipeline has been covered in detail, it is possible to discuss an example of pre-production material handling in a programmable 3D Printing Pipeline. This paper will use the OpenFab Programmable 3D Pipeline's "fablets" as an example.

The OpenFab pipeline is a potential solution to a number of problems in the realm of 3D printing. It can be used with multiple types and brands of 3D printers, allowing for reusable code. (Vidimče, K. et al. 2013) The pipeline results in streamable data, which allows for 3D printers to handle smaller, less computationally intensive print jobs. This structure also allows OpenFab to perform under memory and computational constraints. OpenFab also allows for flexible, programmable, multi-material handling in the preprocessing phase using a technique involving "fablets." Fablets are responsible for the surface and volume controls of the material being printed. Essentially, they are in control of material layout. Fablets and their implementation will be the focus of this section, since much of the programmable OpenFab Pipline is beyond the scope of this paper.

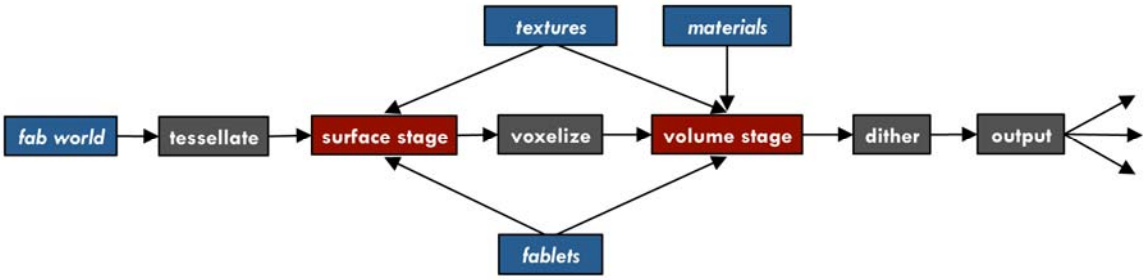


Diagram from: *Openfab: A programmable pipeline for multi-material fabrication* showing the fablets place in the OpenFab Pipeline, positioned where it can control the surface and volume of a printed model. (Vidimče, K. et al. 2013)

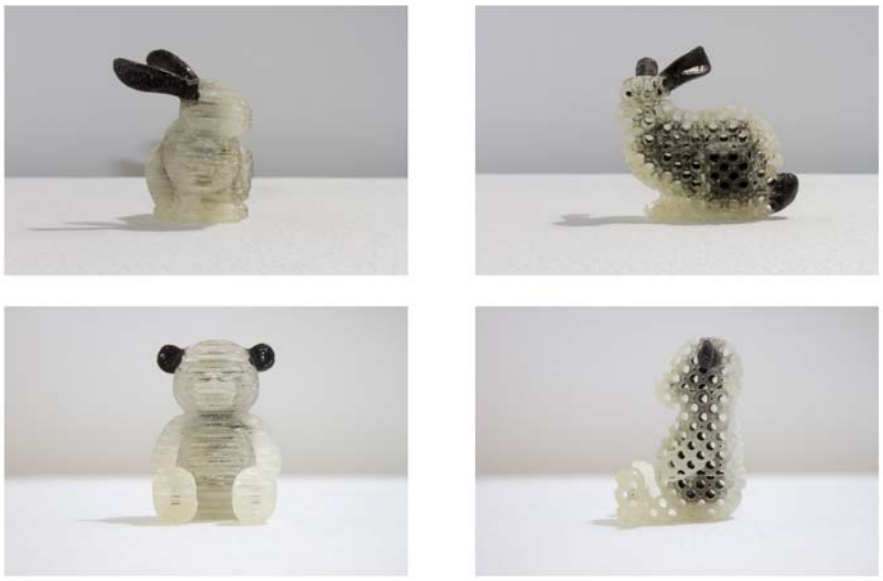
A fablet is "*a shader like program which procedurally defines surface detail and material composition throughout the object volume*." (Vidimče, K. et al. 2013) They are designed to work with additive manufacturing processes, specifically, 3D printers. The purpose of a fablet is to describe the material contents of an object without having to describe the object itself. This is done by using a domain-specific, C-like language called "OpenFL" to describe both how the volume of the material is to be built using additive materials as well as how those materials will interact with a geometry once they are applied to a model. This is accomplished by using various keywords in OpenFL to describe how to change a material when it is near the surface of a model ("near" being defined by the user). This means that unlike a Goal-Based Material Design solution, once a fablet is programmed it can detached from specific geometry and reused for other models. For example, while the hollowing process described previously was designed to avoid hollowing specific sections of a model, these sections were defined automatically by the pipeline without user intervention. This means that if the user wanted to have an identical hollowing pattern appear in different models, the entire algorithm would need to be re-designed. If a hollowing technique were to be implemented with a fablet in OpenFab, the structure could be repeated identically across different models.

Because they are programmable, fablets can also be designed to mimic Goal-Based Material Design solutions. An example of this is provided below.

```
fablet MagicPostcard {
        @uniform {
                float2 border;
                float textureDepth, maxThickness;
                ImageTexture2D fg, bg;
                Material white, black;
        }
const int CARD_FRONT = 0, CARD_BACK = 1;

@Surface(@varying {
        SurfaceAttributes attr,
        float2 uv, int face,
        out float2 uvOut, out int faceOut
        })
{
        // pass through attributes
        uvOut = uv;
        faceOut = face;
        if (face == CARD_BACK) {
         // back face
                float L = bg.Sample1(uv(0), uv(1), 0);
                float thickness;
                if (uv(0) < border(0) || uv(0) > 1 - border(0)
        ||
                        uv(1) < border(1) || uv(1) > 1 -
                border(1)) f
                        thickness = maxThickness;
                } else {
                // material approximation: transmission
                // has quadratic falloff with thickness
                thickness = sqrt(1 - L) * maxThickness;
                }
                return attr.n * thickness;
        } else {
        // no displacement on the front and sides
        return 0;
        }
}
@Volume(@varying {
                VolumeAttributes attr,
                @nearest float2 uv,
                @nearest int face
        })
{
        MaterialComposition mc;
                if (face == CARD_FRONT && // front face
                        abs(distance(attr.voxelCenter))
                <= textureDepth) {
                        // surface texture
                        float L = fg.Sample1(uv(0),
                uv(1), 0);
                        mc.Set(white, L);
                        mc.Set(black, 1 - L);
                } else {
                // background/border
                mc.Set(white, 1);
                }
        return mc;
    }
}
```

Code and images from: *Openfab: A programmable pipeline for multi-material fabrication* (Vidimče, K. et al. 2013) showing a fablet coded in OpenFL that takes two inputs – a foreground image (in this example, the postcard above) and a background image (not pictured). The fablet then creates a plate using displacement on a single axis (see the image below). This sort of process can be repeated using different images for input. This creates a structure that has the results one would expect from a Goal-Based Material Design solution, but can be fine-tuned by the operator.



Images from: *Openfab: A programmable pipeline for multi-material fabrication* (Vidimče, K. et al. 2013) demonstrating a repeating hollowing pattern copied from one model and reused on another model.

Images from: *Openfab: A programmable pipeline for multi-material fabrication*(Vidimče, K. et al. 2013) demonstrating how different fablets can be loaded onto the same model to create different, repeatable results.

As seen from the examples above, programmability in a 3D pipeline allows for a wide range of material layout solutions. Potentially, a well designed programmable 3D pipeline would be able to emulate many Goal-Based Material Design solutions. Unfortunately, this would mean the programmers of the Goal-Based Material Design solution would have to re-implement the algorithms in whatever domain-specific language is being used by the 3D pipeline the solution is being ported into. This may not always be palatable, as authoring tools for various stages in 3D pipelines, like OpenFab,

were still in development as of 2013. (Vidimče, K. et al. 2013) This could mean that the operator would have to learn a new programming language native to each pipeline. Additionally, Goal-Based Material Design solutions can be small and precise while a 3D printing pipeline can be quite large and only as precise as its adjustments allow it to be, meaning that there is a time and a place for each type of solution.

Summary

As stated before, it is becoming important to design software to control the way 3D printer materials are handled in order to address limitations such as physics, cost and other constraints that may not be immediately apparent to the 3D printer's operator. Preproduction material layout controls are a way to address these issues. Two kinds of preproduction material layout control methods in the 3D printing pipeline are Goal-Based Material Design solutions and Programmable 3D Pipelines. Goal-Based Material Design solutions generally involve software that takes models and manipulates them to meet certain specifications or constraints (Vidimče, K. et al. 2013), whereas Programmable 3D Pipelines can use fabrication languages to precisely control the layout of a material throughout a printed volume. (Chen, S. et al. 2013) Both methods have advantages that can make them appealing to programmers and 3D printer operators alike.

In the grand scheme of technology, 3D printers have been around for a relatively short amount of time, but have still managed to make made fantastic gains in technological advancements. The future of this technology is still under development, and promises to continue to advance in new and splendid ways.

Sources

Angel, E. (2008). OpenGL: A primer (3rd ed.). Boston: Pearson Addison-Wesley.

Bächer, M., Whiting, E., Bickel, B., & Sorkine-Hornung, O. (2014). Spin-It: Optimizing Moment of Inertia for Spinnable Objects. *ACM Transactions on Graphics, 33*(4).

Chen, S., Levin, D., Sitthi-Amorn, P., Didyk, P., & Matusik, W. (2013). Spec2Fab: A reducer-tuner model for translating specifications to 3D prints. *ACM Transactions on Graphics (TOG) - SIGGRAPH 2013 Conference Proceedings, 32*(4).

Crassin, C., & Green, S. (2012). Octree-Based Sparse Voxelization Using the GPU Hardware Rasterizer. In P. Cozzi & C. Riccio (Eds.), *OpenGL Insights* (pp. 303-319). New York: CRC Press.

Demmel, J. (1996, April 11). CS267: Lecture 24, Apr 11 1996. Retrieved March 21, 2015, from http://www.cs.berkeley.edu/~demmel/cs267/lecture26/lecture26.html

Fang, S. (n.d.). Hardware Accelerated Voxelization. Retrieved March 22, 2015, from http://cs.iupui.edu/~sfang/vg99.pdf

Johnsonz, N. (2009, November 9). Damn Cool Algorithms: Spatial indexing with Quadtrees and Hilbert Curves. Retrieved March 21, 2015, from

http://blog.notdot.net/2009/11/Damn-Cool-Algorithms-Spatial-indexing-with-Quadtrees-and-Hilbert-Curves

Karrels, E. (1999, September 20). AMC 1994 regional, problem F Quadtrees. Retrieved March 21, 2015, from http://www.karrels.org/Ed/ACM/weur94/prob_f.html

Shirley, P., Ashikhmin, M., Gleicher, M., Marschner, S., Reinhard, E., Sung, K., ... Willemen, P. (2005). *Fundamentals of Computer Graphics* (2nd ed.). Wellesley, MA: A K Peters.

Spatial Partitioning and Indexing: Regular Decomposition: Quadtrees. (2010, August 11). Retrieved March 22, 2015, from http://www.gitta.info/SpatPartitio/en/html/RegDecomp_learningObject3.html

Stava, O., Vanek, J., Benes, B., Carr, N., & Měch, R. (2012). Stress relief: Improving structural strength of 3D printable objects. *ACM Transactions on Graphics, 31*(4).

Stava, O., Vanek, J., Benes, B., Carr, N., & Měch, R. (Writer) (2012b). Stress relief: Improving structural strength of 3D printable objects. *ACM Transactions on Graphics, 31*(4). (Web). Retrieved from http://dl.acm.org/citation.cfm?doid=2508363.2508382&preflayout=flat#formats

Stultz, M. (2015, March 17). Will This New Technology Make 3D Printers 25 Times Faster? - Make:. Retrieved March 18, 2015, from http://makezine.com/2015/03/17/will-new-technology-make-3d-printers-25-times-faster/

Telea, A., & Dalba, A. (2001). Voxel-Based Assessment of Printability of 3D Shapes. Proc. of Mathematical Morphology and Its Applications to Image and Signal Processing, 393–404-393–404. Retrieved March 22, 2015, from http://www.cs.rug.nl/alext/PAPERS/ISMM11/paper.pdf

Vidimče, K. , Wang, S. , Ragan-Kelley, J. , & Matusik, W. (2013). Openfab: A programmable pipeline for multi-material fabrication. ACM Transactions on Graphics (TOG), 32(4), 1-12. doi:10.1145/2461912.2461993

Wang, W., Wang, T. Y., Yang, Z., Liu, L., Tong, X., Tong, W., . . . Liu, X. (2013). Cost-effective Printing of 3D Objects with Skin-Frame Structures. ACM Trans. Graph, 32(6), 177th ser. doi:10.1145/2508363.2508382 http://doi.acm.org/10.1145/2508363.2508382

Wang, W., Wang, T. Y., Yang, Z., Liu, L., Tong, X., Tong, W., . . . Liu, X. (2013, November). Cost-effective printing of 3D objects with skin-frame structures Supplimentals. *ACM SIGGRAPH ASIA 2013*. Lecture conducted from Hong

Kong,
http://dl.acm.org/citation.cfm?doid=2508363.2508382&preflayout=flat#formats

Wei, L. (2005, October 25). A Crash Course on Programmable Graphics Hardware. Retrieved March 24, 2015, from http://graphics.stanford.edu/~liyiwei/courses/GPU/paper/paper.pdf

Zhao, Y. (2008a). Programmable Graphics Pipeline. Retrieved March 24, 2015, from http://www.cs.kent.edu/~zhao/gpu/lectures/ProgrammableGraphicsPipeline.pdf

Zhao, Y. (2008b). Graphics Pipeline. Retrieved March 24, 2015, from http://www.cs.kent.edu/~zhao/gpu/lectures/graphicspipeline.pdf