Utrecht University

Master Thesis

Designing Playground Equipment with VR and 3D printing

Author:
Christian Knaapen

Supervisor:
prof. dr. Remco C. Veltkamp
External supervisors:
Joost Bosman
Martijn M.J.G. Koenis
Second Examiner:
dr. Wolfgang O. Hürst

ICA-5819342
Graduate School of Natural Sciences

November 2018

ING

KLEUR IN CULTUUR

Abstract

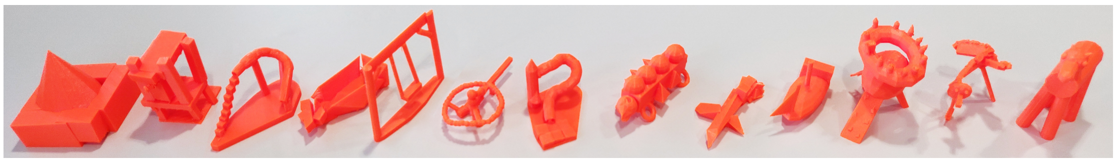


FIGURE 1: Prints from models created by children in VR.

Designing objects in 3D is usually performed in CAD programs, which show the 3D object on a 2D screen. This has a disconnect with the final real world object that can be walked around and interacted with. To bridge the gap between design and the real world result, virtual reality can be used in conjunction with 3D printing. Converting models made in virtual reality to 3D prints is not straightforward, as prints have to show a number of desired properties. Previous studies show that these properties are: (1) it should consist of one connected component that touches the printing platform, (2) it should be able to stand without falling over and (3) it should be structurally sound. A program has been implemented that analyzes these properties on models made in VR. This is the first 3D print analysis program that works in virtual reality. To test the effectiveness of the program we asked 35 elementary school children to design playground equipment in Google Blocks. Then these designs where 3D printed and tested for desired properties. Our program analyzed the connectedness 92.9% of the time, the balance 100% of the time and the strength of the print 50% of the time. This shows that computing vertex connectivity, component intersection, convex hull and the center of mass is sufficient to objects with the first and second property in most cases, but we can not conclude that it is sufficient in all cases. We used a limited sample size of 16 models, so edge cases could exist where this functionality is not sufficient. However, we can conclude that thin area detection is not enough to successfully analyze strength in all models. While further research is needed to draw more conclusions, this is a good first step in gaining understanding of the correlation between virtual reality and 3D printing.

Acknowledgements

First of all I would like to thank prof. dr. Remco C. Veltkamp, my supervisor, for his guidance and feedback throughout the project. I would also like to express my gratitude to Joost Bosman and Martijn J.G. Koenis from ING for their support and enthusiasm in this project. Furthermore I would like to thank Amanda de Gast from Stichting KleurInCultuur for organizing the experiment and Harriet Bosman and Mannie Krak for their assistance with the experiment. Without them this project would not have been possible. Additionally I would like to thank the teachers Cathelijne Doeze Jagers and Dong-Shik van Mierlo from Sterrenschool de Ruimte and the children of group 8a and 8b for their cooperation and time. Lastly I would like to express my gratitude to my colleagues at ING, in particular Wendy Bolier for her previous research in this area, Daan de Rond for his 3D printing expertise and Jenna Hardin for making a beautiful background for my program. And a special thanks to Takman, Reinier, Denzel, Thomas, Marjolein, Daan and Ishdeep for playtesting and providing me with test models.

Contents

Chapter 1

Introduction

1.1 Context

Designing objects in 3D is usually performed in CAD programs, which show the 3D object on a 2D screen. This has a disconnect with the final real world object that can be walked around and interacted with. To bridge the gap between design and the real world result, virtual reality can be used in conjunction with 3D printing. Virtual reality has the benefit of showing objects in 3D using different viewpoints for each eye and the ability to walk around the object. In addition, the virtual part ensures a rapid design process with no material cost and easy modification. 3D printing on a smaller scale can help in the design process by providing a tangible object that can more easily be shown to and passed around in a group of people. Furthermore, advancements in 3D printing technology allow printing to be done at a bigger scale to potentially create the final product.

However, making a 3D print takes time and effort. The model needs to be prepared to fit certain requirements, including being a watertight manifold mesh. Supports are also usually needed and after the print is finished those supports need to be manually removed. The print itself can easily take several hours, so the entire process is not quick by any means. Furthermore, there is no guarantee that the print will succeed and create a stable and strong object. And because the process is not quick, using trial and error to make stable 3D prints is not desirable. An automatic method of preparing a model for 3D printing and analyzing whether it will be successful is therefore needed.

This research project aims to answer the question of what that method entails; what is needed in a program to make sure 3d prints will result in stable and strong objects. This project is part of a bigger plan that includes two other projects to be performed at a later date. The design of the application will be analyzed and adjusted to focus on compliance to safety and user-friendliness by a Games and Interaction student from the University of the Arts Utrecht (HKU). A Science Education student from the Freudenthal Institute will help the children pick a best design and measures the effect of the design and implementation during play.

This project is a collaboration between ING Corebank University and Stichting Kleur-InCultuur (KIC). The ING Corebank University is a department within the Dutch bank ING, which experiments with the newest technologies including the HTC Vive, the Microsoft Hololens and the Pepper robot. They provided a working place and the hardware (HTC Vive, 3D printer) needed for this project. Foundation Color in Culture, located in Almere, introduces children to art and culture with several activities in- and outside of school hours. One of those activities is the Spacelab, where kids learn to draw in virtual reality using the HTC Vive headset and are encouraged to follow a specific art style. They organized and helped with the experiment sessions at the elementary school and the logistics around it.

1.2 Objectives

The main objective of this research project is to find out what functionalities are necessary to support the design of structurally integral 3D printable objects. To do this, we first need to know what the requirements are for a 3D print to be considered successful. In this regard the first research question is:

Question 1a (Q1a) What are desirable properties of 3D printed objects that commonly cause issues?

To answer this question a literature study is carried out, as can be seen in Chapter 2.

1.3 Outline

In Chapter 2 the background and related works are described. The chapter starts with an explanation of the research methodology and then moves on to give a background in 3D printing and the challenges that come with that. Various file formats are discussed and in the second half assorted papers are described that deal with printability assessment.

Chapter 3 contains the approach and the formulated research question. In addition, a description of the experiment is given as well as the time plan.

In Chapter 4 the implementation of the program is described. It explains how VR was implemented and it explains each of the five steps: loading a Google Blocks model, connectedness analysis, balance analysis, strength analysis and saving to a 3D printable format.

Chapter 5 contains the setup and results of the experiment and Chapter 6 contains the discussion and conclusion.

Chapter 2

Related work

2.1 Research Methodology

Research in the fields of modeling in VR, 3D printing, printability assessment and structural integrity were performed using Google Scholar. Relevant keywords for each field were chosen, optionally together with one of the following: "overview, "comparison" or "review". These were only chosen when there were enough papers in the field found previously. In addition to searching directly through Google Scholar, papers were also found by going through the references of previously found papers, and through papers that cite the previously found papers.

2.2 Modeling in VR

In recent years virtual reality has gained a substantial amount of popularity, with the release of the Oculus Rift, HTC Vive, PSVR and Gear VR. With that, there are many games and software products in development or already released. A small portion of those focus on the creation of 3D models, for a variety of purposes such as game development, architecture and art. There are several ways to approach modeling in VR. These techniques are separated into 4 categories: brush strokes, sculpting, surface modeling and primitives. An overview of existing VR modeling software is given in table 2.1.

One method of modeling in VR is by use of brush strokes, as introduced in Tilt Brush by Google [Google, 2016]. This method uses separate objects for each brush stroke. The advantage of this is that it supports flat brushes, particles and fancy shaders. The disadvantage is that this does not create models that can easily be transferred to other programs, including 3D printer software, because of the unconventional brush strokes.

Table 2.1: Overview of VR modeling software.

Software	**Technique**	**Representation**	**Export format**
Tilt Brush	Brush strokes	Curves	FBX
MasterpieceVR	Sculpting	Voxels	OBJ, FBX & STL
Gravity Sketch	Surface modeling	NURBS	OBJ
Fantastic Contraption	Primitives	Polygons	None
Google Blocks	Primitives	Polygons	OBJ

By the looks of it the brush strokes are represented using curves, but no information can be found on it. Tilt Brush and its brush stroke method is made to create and view those creation within Tilt Brush, not to be exported and used for other software.

Figure 2.1: Tilt Brush by Google.

Sculpting is another method that is still upcoming in the world of virtual reality. It mimics working with real clay by providing tools to add, remove and move the surface of an object. This allows for easier creation of more natural objects like people and animals and, depending on the resolution of the workspace, allows for a lot of detail. Sculpting in virtual reality is still in its early days, because of the increased performance requirements compared with other modeling techniques. One software product that already manages this is MasterpieceVR [MasterpieceVR, 2017].

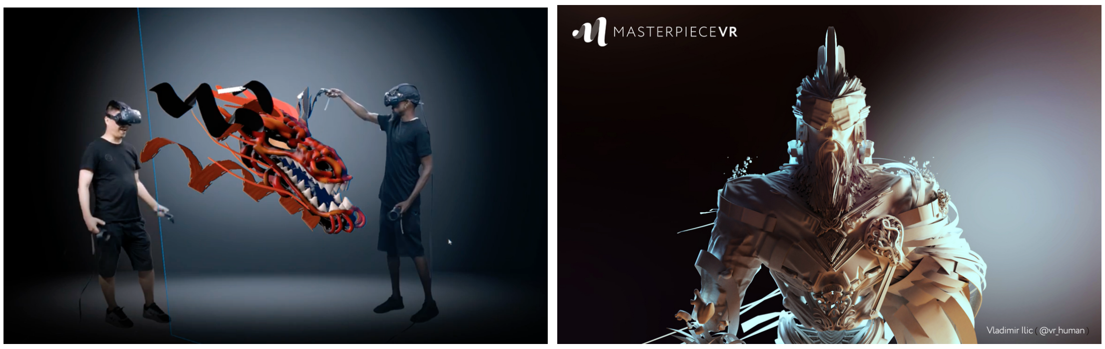


Figure 2.2: MasterpieceVR.

Galyean and F. Hughes [1991] introduced sculpting as a modeling technique. They used a voxel array much like a bitmap but in three dimensions. These voxels were manipulated using a 3D input device called the Polhemus Isotrack device. This device is similar to today's VR controllers in that it is not a representation of a 2D tool converted to 3D, but it tracks within a 3D space in the real world. Their input device has considerable noise though and is not intuitive to use, whereas the VR controllers of today are very accurate and are even easy to use for children. To visualize the voxel map they used the marching cube algorithm [Lorensen and E. Cline, 1987].

Surface modeling is a technique that uses nonuniform rational B-splines (NURBS) to form a model. This creates smooth surfaces that can be bent and curved. This has applications in the automotive industry, and is in development for VR in the form of Gravity Sketch [Limited, 2017]. An interactive technique of how to implement surface modeling for VR is described by McGraw et al. [2017].

Another approach is to work with primitive 3D objects like cubes and cylinders. It is possible to create complex models with this method by placing many of these primitives and by manipulating them (rotate, scale, skew). Popular software products that use this method are Google Blocks [Google, 2017] and Fantastic Contraption [RadialGames, 2016]. The latter uses spheres you can grab on certain spots of each primitive to manipulate it. Dragging one sphere might stretch the primitive, while another one might rotate it.

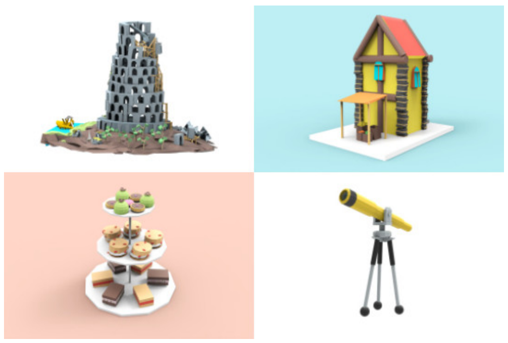


FIGURE 2.3: Some models created by others in Google Blocks.

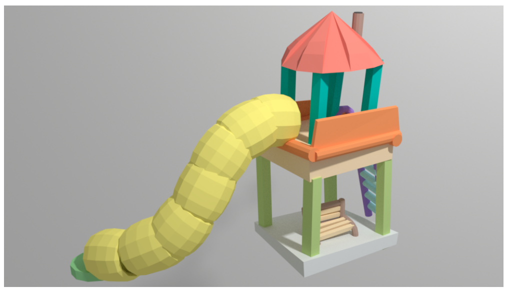


FIGURE 2.4: A model created by us in Google Blocks in around 15 minutes with no prior experience.

FIGURE 2.5: Fantastic Contraption by RadialGames [2016].

Google Blocks uses a method where primitives can be stretched as soon as you place them. This is achieved by clicking and holding the trigger to place the object in its initial position and then dragging to make the object stretch to the second point. This can also be done on only one or two axes. For example, you might create a pole by stretching a cube along one axis, and a platform by stretching along two axes. There are also more advanced tools like extruding a face or grabbing and moving faces and edges. Google Blocks always makes sure that faces do not overlap and there are no holes in the model. It also includes an advanced snapping option and easy copy and paste of parts of the model. Figure 2.3 shows a collection of examples of models created by others in Google Blocks. Figure 2.4 shows a model created by us in around 15 minutes with no prior experience in the program. The program is very intuitive and for this reason we let the children design the majority of their playground equipment in this program.

Bolier et al. [2017] investigated the effects of introducing drawing in VR to elementary school education. She performed an experiment with 18 children between the ages of 10 and 12 and split them into three groups. One group participated in training sessions in VR with several exercises, another group were allowed to freely paint in VR for the same amount of time, and the third group did not get to train in VR at all. Afterwards their spatial ability was tested and their proficiency in creating drawings in VR. The results show that their 3D drawing skills do improve, even after only a few sessions. And while there does seem to be a correlation between their drawing skills and their spatial ability, further research is needed to confirm this. No previous research has been done into the educational use of VR drawing, so this thesis laid an important foundation to gain a better understanding of the benefits of VR drawing when introduced at elementary schools.

2.3 3D Printing

3D printing is the process of joining and fusing material under computer control to create a 3 dimensional object. There are multiple techniques for this, one of which is stereolithography. Using photopolymerization it causes molecules to link together, forming polymers. The most common technique for consumer 3D printers, however, is fused deposition modeling (FDM). Instead of printing ink on paper it prints a material onto a platform during the first layer, and onto the material itself on the consequent layers. This material can be of several types, like thermoplastics or even metal using specialized 3D printers. Thermoplastics become malleable when exposed to heat. Two types of thermoplastics are used for 3D printing: Polylactic Acid (PLA) and Acrylonitrile Butadiene Styrene (ABS) [Hesse, 2015]. PLA is made from organic material such as sugarcane and cornstarch. It has a lower melting point and is weaker than ABS. ABS is an oil-based plastic and is sturdier but requires a heated printing bed to prevent warping during cooling.

Oropallo and Piegl [2016] summarize limitations of 3D printing:

- A proper CAD design is needed with the right topological requirements. The design needs to be a watertight manifold surface that cannot overlap itself.
- Consumer-grade printers currently only work with thermoplastics. Other materials such as metal are still in development.
- Printers only print in one color. If you want multiple colors, you need a printer that supports multiple filaments or you have to paint over the finished object manually.
- The finished objects have a rough surface finish. The printed layers are clearly visible, especially on near horizontal angles. This can be reduced by using smaller layer sizes but that increases the printing time and the layers will still be visible. Manual processing in the form of sanding is required to get a legitimate smooth surface.
- The objects are not as strong as conventionally manufactured parts.
- 3D printing is not cost-effective for mass production, because cost per item cannot be decreased. Each object you want to print costs the same amount of material and time, no matter how many objects you want, whereas conventional mass production is able to reduce those drastically.
- The accuracy of printers is limited. Other object manufacturing processes such as casting are able to achieve higher precision.
- The size is limited, usually between 10 and 20 centimeter in all 3 dimensions. If you want a bigger object you can either print them in parts and glue the parts together or call in a professional company, and then the costs are non-trivial.

There are several file formats that contain information about 3D models, in particular geometry, color, texture and material information. Some examples include OBJ, FBX, STL and COLLADA. These 3D file formats can technically all be used for 3D printers, but only a small subset of those are supported by 3D printer manufacturers. When searching for the most common 3D printer file formats, the sources all give different answers [Chakravorty, 2018] [NIH, 2018] [Clouds, 2018]. All of them mention STL and OBJ, indicating these are in fact the most common formats used today. Most of them mention VRML, AMF and FBX as well.

FIGURE 2.6: Da Vinci 1.0 Aio 3D printer by XYZprinting.

TABLE 2.2: File sizes of a model with approximately 63.000 triangles.

3D File Format	Encoding	Size in MB
PLY	Binary	1.1
X3DB	Binary	1.3
OBJ	ASCII	2
PLY	ASCII	2
X3D	ASCII	2.1
VRML	ASCII	2.7
STL	Binary	3
STL	ASCII	11

The STL format is a simple format that only describes the geometry of an object. It was created by Chuck Hull in 1987, at the same time as the invention of the 3D printer [3DSystems, 2005]. The first 3D printer used the stereolithography technique, which is what the STL format is named after.

The STL format can be encoded in ASCII and binary encodings. ASCII is humanly readable but has a larger file size, while binary is only readable by computers and is more compact. Regardless, even in binary encoding it creates one of the biggest file sizes, as can be seen in table 2.2.

The OBJ format is used for multicolored printing. Unlike STL, OBJ can store information about color, texture and material. It is also able to store geometry using free-form curves and surfaces instead of triangles. This reduces file size and increases precision. Materials are stored in a separate Material Template Library (MTL) file. It can define properties such as diffuse and specular color, texture coordinates and ambient occlusion. The downside of OBJ is that it is much more complex than STL and as a result repairing an OBJ file is no easy task. It is also not as widely adopted as STL.

To address many of the shortcomings of the STL format, AMF was introduced in 2011 by ASTM International [ASTM, 2018]. It has support for geometry, scale, RGBA colors, materials, lattices, duplicates, and orientation. It uses the XML format which makes it more readable and has support for curved triangles. Just like surfaces in OBJ this can increase precision and reduce file size. It can handle mixed materials and microstructures. And if printers do not support certain features like multicolored printers, it is easy to ignore that part of the AMF file. The main drawback of AMF is its limited adoption. At the time of its introduction the old STL format did not pose too many issues so there was no necessity to change entire pipelines to the better AMF format. Since then, a more widely adopted format has been introduced: 3MF.

The 3MF file format was initially developed by Microsoft internally. In 2015 they founded the 3MF Consortium to govern further development of the format, together with key stakeholders in the 3D printing space such as Autodesk, Shapeways, 3D Systems, HP, Siemens and many more [Vanderkay, 2015]. The design goals of the 3MF format are completeness: human readability, simplicity, extensibility, unambiguous language and free access. It uses an XML format like AMF for readability but stores triangles in a more compact way. It also ensures the objects are completely manifold without holes and overlapping triangles. This means no mesh repairs are necessary before printing and it reduces problems that can occur during printing. 3MF is still in the process of being adopted, but already well on its way to become the de-facto standard 3D printer file type.

2.4 Printability assessment

Several issues can arise in the process of 3D printing and with the objects after they are printed. During printing the surface might be too steep (and requires additional supports), the material might not stick to the platform if the calibrations are off, the material might not stick to the previous layer of the object is too thin or the object might warp a little due to cooling down too fast. Several test prints were performed using the Da Vinci 1.0 Aio 3D printer by XYZprinting and some of the defects that occurred are given in figures 2.7 to 2.10. This thesis project does not focus on issues during printing but rather on the potential issues the object can have after it is printed.

If the printing process did go well there could still be issues with the object after the print. This thesis project focuses on these issues most of all. It might be unbalanced and fall over once the supports for printing are removed. It might also break apart due to gravity or picking the object up. Especially thin areas are susceptible to breaking. It would be ideal if we could analyze models beforehand to detect possible problem areas before we even started printing. This way time and material can be saved by minimizing the chance a print would fail.

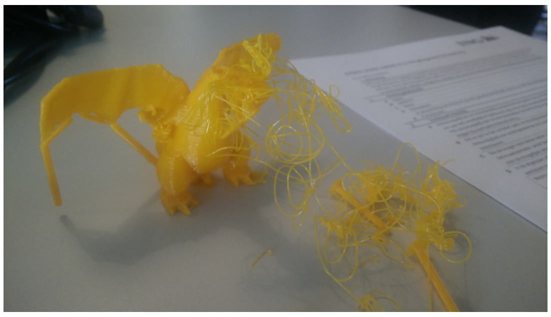


FIGURE 2.7: One of the supports fell over.

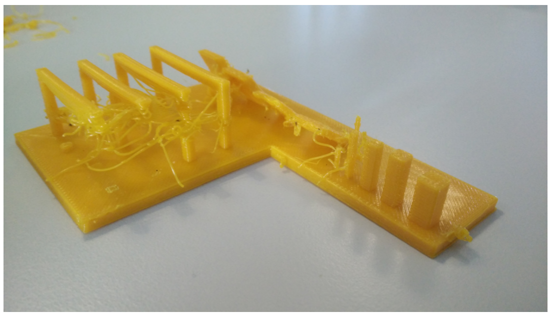


FIGURE 2.8: The material did not stick to itself.

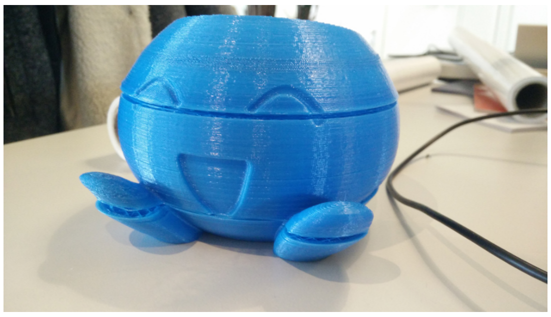


FIGURE 2.9: The material did not stick to itself.

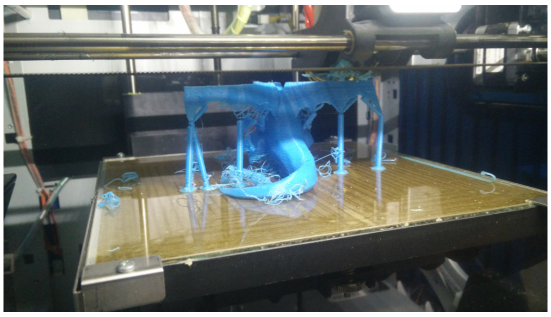


FIGURE 2.10: The feet did not stick to the platform.

A model can consist of multiple components. For printability it is important that these components are either connected to each other or that they touch the ground. If a component is not connected and floats it will waste material in the form of supports required to print that component. These supports will be removed at the end of the printing process, so printing them close to the platform is much more efficient. The detection of separate components is straightforward: start at a vertex v and perform a breadth-first or depth-first search. To detect all components, loop over all vertices and start a search for each vertex that has not been included in a previously found component [Hopcroft and Tarjan, 1973]. It is also important to detect which components intersect. Unity's built-in collision detection will take care of this. It uses the NVIDIA PhysX collision algorithm [Anthony, 2014].

2.4.1 Balance

One of the requirements for a successful 3D print that we proposed is that it can stand in its default upright position without falling over. To detect if that is the case, we can use a combination of a convex hull and the center of gravity. First we calculate the convex hull of the vertices that touch the ground (below a certain height) and the center of mass of the model. If the center of mass projected onto the ground plane lies within the convex hull, the model will not fall over.

There are several algorithms to calculate the convex hull of a set of points. Chand and Kapur [1970] and Jarvis [1973] both independently created the gift wrapping algorithm: one of the simplest convex hull algorithms and has a time complexity of $O(nh)$, where n is the number of points in the set and h the number of points in the hull. In 1972 Graham [1972] created the Graham Scan method with time complexity $O(n \log n)$. If the points are sorted by one of the coordinates or by the angle to a fixed vector, the algorithm completes in $O(n)$ time. The Quickhull algorithm was created a few years later, independently by both Eddy [1977] and Bykat [1978], which also takes $O(n \log n)$ time, but in the worst case takes $O(n^2)$ time. The first optimal output-sensitive algorithm was created by Kirkpatrick and Seidel [1986] and takes $O(n \log h)$ time, using the marriage-before-conquest technique. A simpler optimal output-sensitive algorithm was created a decade later by Chan [1996].

The center of mass of a model can be approximated in several ways. We will assume that the material density will be the same throughout the model, so the center of mass is equal to the geometric center (or centroid). One approximation is taking the average of all the vertices:

$$centroid = average(x), average(y), average(z)$$

The accuracy of this method depends on the distribution of the vertices. If the mesh has a highly detailed area, the center of mass approximation will be distorted towards that area. Another approximation is taking the mathematical middle in each dimension:

$$centroid = middle(x), middle(y), middle(z)$$

also known as the mean of the extrema. This will be the centroid of the bounding box of the mesh. If the mesh does not look like its bounding box, a more accurate approximation can be determined by creating a voxel representation of the model. Each voxel inside the mesh will be considered a point in the point set. The resulting point set will be equally distributed throughout the mesh, making it possible to get an accurate approximation using the averages of all points, where the accuracy depends on the resolution of the voxel grid.

A paper worth mentioning in the area of balancing 3D printed objects is 'Make it stand: balancing shapes for 3D fabrication' by Prévost et al. [2013]. They introduce an algorithm that assists users in producing balanced designs by interactively deforming an existing model. The user edits a shape and the optimizer continuously analyses the model to make sure it can stand on its intended basis with the chosen orientation. With their method it is possible to balance shapes in surprising poses, as seen in figure 2.11.

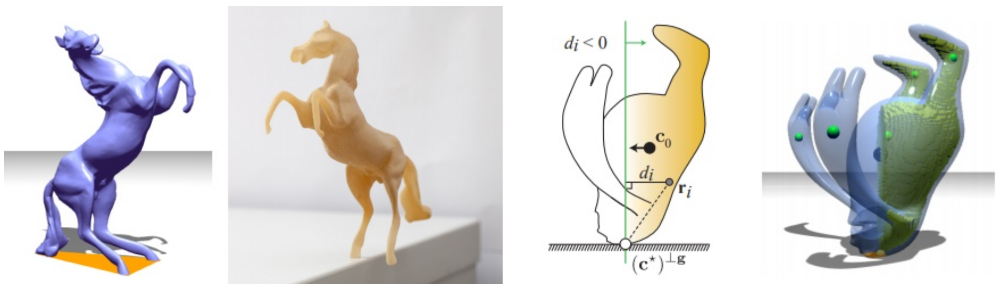


FIGURE 2.11: Results of the algorithm of Prévost et al. [2013].

Less than two years later Christiansen et al. [2015] propose an automatic method of balancing shapes, without the need for user interaction. This is achieved by hollowing and shifting the weight inside the model to keep the impact on appearance at a minimum. If shifting weight is not enough they deform the model near the base to rotate the entire object into a more balanced position. The difference in appearance with the original model is usually less than the method by Prévost et al.

2.4.2 Structural integrity

Telea and Jalba [2011] presented the first automatic method for assessing the printability of models. They define a model to be printable if the removal of thin areas does not result in critical topological or geometrical changes. Critical changes include the disappearance of notable detail, the disconnection of large fragments and the creation of large holes. First they voxelize the shape into a binary voxel model defined by Ω using the method of Nooruddin and Turk [2003]. This method can handle models with holes, double walls and self-intersecting meshes. This means that the input model does not need to be a watertight manifold surface, which solves a number of problems. They detect locally thin areas Θ by using multi-scale set erosion and dilation on the model. The difference between the result and the original model indicates where the model is too thin, since during erosion those areas are removed entirely, which is not recoverable by the dilation. This can be calculated efficiently using a top-hat transform and the Euclidean norm to compute the Euclidean distance transform.

Next they classify the voxels into four categories: thin (T), rump (R), interface (I) and boundary (B) using the following formulas:

$$
\begin{aligned}
R &= \Omega \setminus \Theta \\
I &= \{x \in \Theta | \exists y \in n_6(x), y \in \Omega, y \notin \Theta\} \\
B &= \{x \in \Theta | \exists y \in n_6(x), y \in \overline{\Omega}\} \\
T &= \Theta \setminus (B \cup I)
\end{aligned}
\tag{2.1}
$$

Where $n_6(x)$ denotes the 6 voxel neighbors of x. The printability of a shape is assessed by two metrics: an area-based and a geodesic length metric. The area-based metric uses the shared property of spikes, bridges and holes of long thin structures connected to the rump with small interfaces. The criticality is quantified by the ratio of the boundary area and the interface area. The geodesic length metric measures the eccentricity of an area, with high spikes having a high value as well as long bridges and large thin holes.

While this method is a good start for automatically assessing the critical areas in a 3D printed model, it is only based on thickness. A more useful model would also focus on structural integrity, including the effect of forces such as gravity or picking up the object with your hand. The paper also only focuses on analysis; the correction is left to humans to do manually. Based on the output of their method, it is not always trivial to see what correction to make. Furthermore, while the results were looked at by domain experts (with no reference to who that may be), the models were not actually printed to see if those areas marked as critical by their method would actually create problems in the printing process.

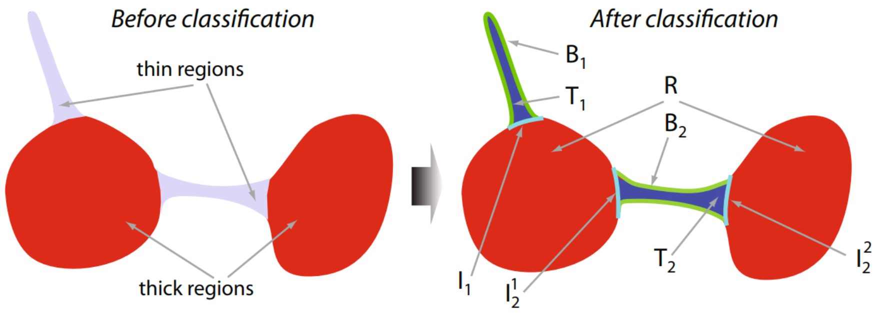


FIGURE 2.12: Voxel classificiation by Telea and Jalba [2011].

A year later Stava et al. [2012] present a system that does look at structural integrity by combining a lightweight structural analysis solver with 3D medial axis approximations. It can also automatically correct problem areas by use of hollowing, thickening and adding struts. They analyze structural stress caused by gravity and by humans pressing on the object to pick it up with two fingers. The model is iteratively analyzed and corrected until no problem areas remain, as shown in figure 2.13.

Since it is difficult to analyze all possible structural loads an object can withstand, the paper only focuses on the highly probable cases. These include gravity in the default upright position and in orientations calculated by the algorithm of Fu et al. [2008]. In addition the most likely pinch grip positions are calculated and analyzed. For every face center, three grip directions are considered: one in the direction of the object's center of mass, one that points to the center of mass projected to a plane that is parallel to the ground level, and another in the direction of the closest medial axis. These grip configurations are then given a score based on the difference with the normal of the surface area roughly the size of a finger, and on accessibility by humans using ambient occlusion. Several configurations with the highest values are chosen that are significantly different from each other.

The structural stability of an object depends on the printing technology and material used. The paper uses a homogeneous material, but the accuracy can be improved by considering what printer and material is going to be used. The model is first converted to a tetrahedral mesh [Si and TetGen, 2006] and then the stress is calculated using the Finite Element Method with quadratic tetrahedral elements [Hughes, 2012]. The structural load with the highest stress is selected and different correction methods are analyzed. The correction with the highest effectiveness and lowest impact on appearance is selected. The resulting model is then analyzed and corrected on repeat until no problem areas are found. The main drawback of this method is that a lot of simulation iterations are necessary for complex input models, so analysis and correction can potentially take a very long time. Some complex models give trouble too, such as enclosed structures and models with sharp spikes.

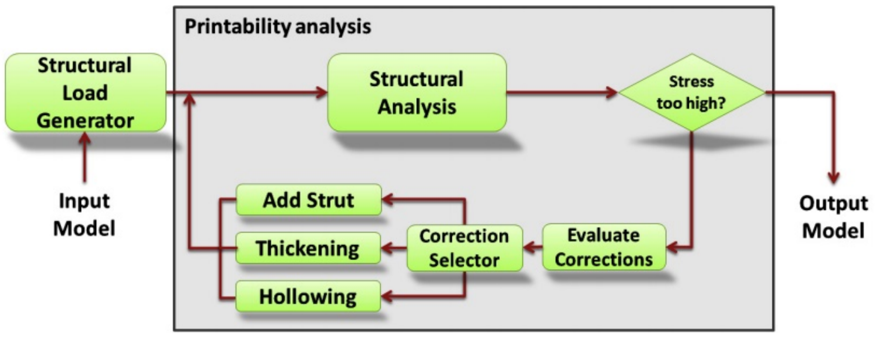


FIGURE 2.13: Overall schema of the system by Stava et al. [2012].

To this end Zhou et al. [2013] try to optimize the analysis part of structural integrity by focusing on what load causes the most damage, instead of focusing on the most probable loads. This ensures that the weakest parts of the object get addressed. For each weak region they calculate how to put the maximum stress on it. To optimize the analysis they use some approximations, which speeds it up by in some cases 10.000 times compared to brute force. The output is a weakness map that shows what parts of the object are most likely to break under circumstances like pinching the object and dropping it.

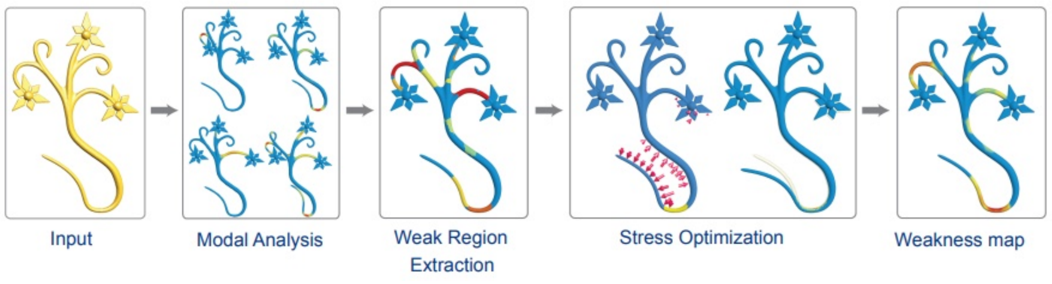


Figure 2.14: Algorithm pipeline of Worst-case Structural Analysis by Zhou et al. [2013].

The previously discussed methods all rely on the Finite Element Method to solve the structural analysis. This method is known to be quite slow and it requires a mesh without holes and intersections. Another analysis technique is to use cross-sections of the mesh. "Cross-sectional Structural Analysis for 3D Printing Optimization"[Umetani and Schmidt, 2013] uses bending momentum equilibrium to achieve interactive analysis speeds and to optimize the orientation to maximize the mechanical strength of the 3D printed object. "Cross section-based hollowing and structural enhancement" [Wang et al., 2017] minimizes material use by hollowing structurally strong cross-sections and enhances strength by reinforcing structurally weak cross-sections.

Chapter 3

Approach

Based on the literature study, we discovered that desirable properties of 3D printed objects that commonly cause issues are, as answer to **Q1a**: (1) it should consist of one connected component that touches the printing platform, (2) it should be able to stand without falling over and (3) it should be structurally sound. An application has been implemented that analyzes models on this list of requirements and shows users what areas of the model need to be improved. If the analysis shows no problems the object can be 3D printed. If the 3D print is structurally integral, it can be concluded that the range of capabilities in the application is sufficient to create a stable print out of that model.

This research project focuses on models created in the application Google Blocks [Google, 2017], which works by placing and manipulating primitives (e.g. spheres, blocks). The program focuses on analyzing models exported with Google Blocks on the desirable properties: connectedness, balance and strength. Separated components of the model are detected and visualized to the user. The center of mass is calculated and if that point does not lie above the vertices that touch the ground, the model visually falls over. The model is analyzed on strength and weak regions are visualized by coloring the model in those areas. Some basic model editing tools are also provided to prevent the user from having to switch between Google Blocks and the program too often. These tools include deleting components, adding struts, flattening the ground, adding a pedestal and thickening areas.

To thoroughly test if the range of capabilities is sufficient, a group of children (till the age of 14) from an elementary school in Almere got to design their own favorite playground equipment in VR. They did so in Google Blocks, and afterwards these designs were analyzed to determine if they can be 3D printed. This was shown to the children in VR. If areas with issues were discovered an attempt is made to fix them. This analysis and correction provided models that can be printed. All of the designs the children made were 3D printed on a small scale using the Original Prusa i3 MK3 printer. After that the children discussed their designs in class and picked one favorite design. In a future project this design might be printed in real scale and will be put in an appropriate location (schoolyard/park) in Almere.

FIGURE 3.1: Original Prusa i3 MK3 printer.

3.1 Research question

In addition to the first research question as described in Chapter 1, a second research question has been formulated with regards to the project introduced above. In this thesis a 'model' denotes a virtual representation whereas an 'object' denotes its physical counterpart.

Question 1b (Q1b) What functionalities are necessary to convert a model created in Google Blocks into a model that makes it possible for the Original Prusa i3 MK3 printer to print the corresponding object with desired properties as specified in Q1a?

Hypothesis 1 (H1) The functionalities that are necessary include algorithms to compute vertex connectivity and component intersection to ensure connectedness, convex hull and center of mass computation to ensure balance and thin areas detection to ensure strength. These functionalities together will make it possible for the Original Prusa i3 MK3 printer to print the object with desired properties as specified in Q1a.

3.2 Experiment

An experiment is conducted to answer the research question and verify the hypothesis. If the models were created or chosen by us then the risk exists of choosing only models that will work with our program. To obtain models with a variety of issues that properly test our program, the participants of this experiment were a group of children (of the approximate age of 12) from an elementary school in Almere. The children were not told what it takes to create 3D printable objects until after the first analysis by our program.

In a series of three sessions between 3 September and 14 September they got to design their own favorite playground equipment using the HTC Vive. They made groups of two, making 16 groups in total. In the first two sessions they used Google Blocks and in the third session they used our developed program to create a 3D printable design.

In the two weeks after these designs were 3D printed. After the printing supports were removed they were visually inspected for connectedness and balance. The strength of the objects was determined by drop tests. The objects were dropped on a hard surface and if they survived a drop test from table height (75 centimeter), they passed. Then in an additional classical session with the children the prints were handed out and a discussion was held to find out which design is the overall favorite. In a future project, this favorite design might be 3D printed in real size by a professional 3D printing company.

3.3 Time plan

The execution phase of the project consists of implementing the application, writing the thesis paper and conducting the experiment. This phase takes 91 days in total and is split into an implementation part (41 days) and an experiment part (50 days). The following tasks are completed in chronological order.

Implementation part (18 June - 17 August):

- View object in VR (scale, move) (10 days, in research phase)
- Visualize areas of object with edges and face colors (5 days)
- Detect if connected and visualize (4 days)
- Export to STL format (2 days)
- Balance analysis and simulate (10 days)
- Strength analysis (10 days)

- Correct areas with issues (10 days)

Experiment part, after summer holiday (3 September - 9 November):

- Sessions with children (3 days over 2 weeks)
- Write Implementation chapter (10 days)
- 3D print designs, inspection and drop test (5 days over 3 weeks)
- Write Experiment and Conclusion chapters (10 days)
- Discussion of best design and open day (2 days)
- Produce a dissemination (5 days)
- Prepare presentation (5 days)
- Process feedback and finalization (11 days)

Chapter 4

Implementation

This chapter explains how our analysis and correction software is implemented. This program is implemented in Unity3D and allows children to convert their Google Blocks models into 3D printable models. Figure 4.1 shows the overview of the printing process and what tools are used for what purpose. Models are designed in Google Blocks and saved as OBJ files. Our program imports those files and analyzes the model. Some issues can be corrected with the simple correction tools included in our program. For other issues the user needs to go back to Google Blocks and edit the original model. If our program approves the model, it can be exported as STL file, after which it can be imported in Autodesk Meshmixer. The model is scaled to be 10 centimeter in the largest axis and optionally printing supports are added. Then the model is saved as STL once more and imported into Slic3r Prusa Edition, in which we optionally add supports as well. After that the model is sliced, a process that converts models to 3D printer instructions, and saved as GCODE. After this the GCODE file is uploaded to the printer after which it can start printing.

Our program works in a step by step process, as can be seen in figure 4.2. The first step is to import the OBJ file. In the following three steps the model is analyzed on connectedness, balance and strength and the results are visualized for the user. Some simple correction tools are provided for each step. The model must be approved by the analysis in that area before the user can go to the next step. This ensures that the balance and strength tests can be performed on a connected object. The fifth and last step is to export the model to STL format so it can be 3D printed. The software can be run on a normal screen and in VR. To our knowledge, this makes it the first 3D print analysis program that works in virtual reality. All functionality has been developed by us, with exception of the SteamVR and VRTK plugin for VR support and the Runtime OBJ Importer plugin for importing OBJ files.

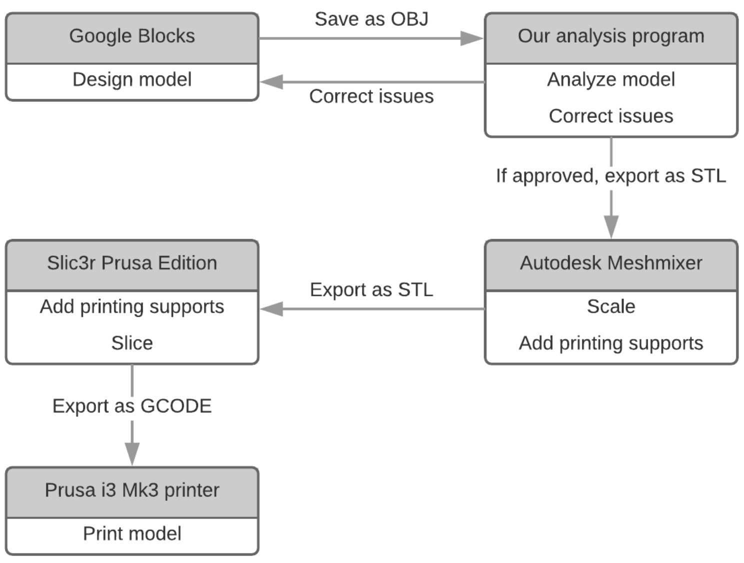


FIGURE 4.1: An overview of the total printing process.

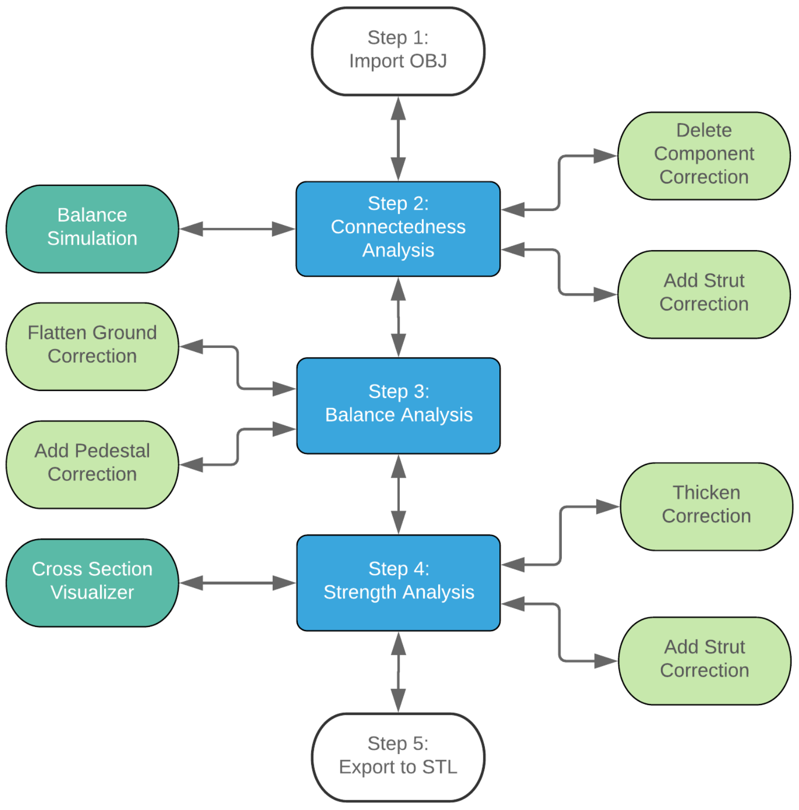


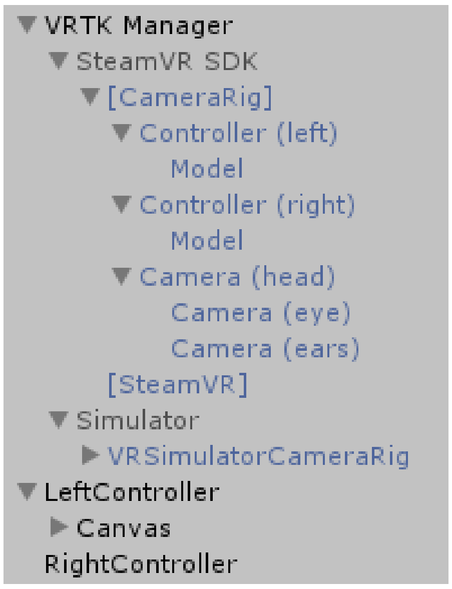
FIGURE 4.2: An overview of the steps within our analysis program. The main analysis steps are blue, the corrections are green and the extra visualizations are cyan.

Table 4.1: Features and their corresponding classes.

Feature	**Classes**
VR support	VRControl
Non-VR camera control	MouseOrbit
Step-by-step process	Stepper
Importing OBJ file	File Management/FileLoader File Management/ObjImport
Export to STL	File Management/STLExport
Common analysis features	Analyses/Analysis Analyses/AnalysesManager Analyses/PreAnalysis
Connectedness analysis	Analyses/ConnectednessAnalysis Analyses/TriangleTriangleIntersection
Balance analysis	Analyses/BalanceAnalysis
Strength analysis	Analyses/StrengthAnalysis
Common correction features	Corrections/Correction Corrections/CorrectionsManager
Add strut correction	Corrections/AddStrut
Delete component correction	Corrections/DeleteComponent
Flatten ground correction	Corrections/FlattenGround
Add pedestal correction	Corrections/AddPedestal
Thicken correction	Corrections/Thicken
Balance simulation	Corrections/BalanceSimulation
Cross section visualizer	Corrections/CrossSectionViewer

4.1 VR setup

To keep the experience consistent between our program and Google Blocks, our program can be run in VR using the HTC Vive. To make this possible the VRTK library[1] is used, together with the SteamVR plugin[2]. The setup can be seen in figure 4.3. The VRTK manager activates the SteamVR SDK or the simulator if the former is not available. The SteamVR SDK shows the controller models, puts a camera at the head, manages the play area and has several other functions. The simulator is a simulation of VR using the mouse and keyboard. It is clunky but good for testing. The non-VR mode of our software will not be using this, it will use the traditional input mechanics with a separate canvas.



```
▼ VRTK Manager
  ▼ SteamVR SDK
    ▼ [CameraRig]
      ▼ Controller (left)
          Model
      ▼ Controller (right)
          Model
      ▼ Camera (head)
          Camera (eye)
          Camera (ears)
      [SteamVR]
  ▼ Simulator
    ▶ VRSimulatorCameraRig
▼ LeftController
  ▶ Canvas
  RightController
```

FIGURE 4.3: VRTK setup.

In order to view the model more easily from different angles, the user can hold the grip button on both controllers and move them. This not only moves the model but it also rotates and scales it, all in one movement, depending on the relative position between the two controllers. This is a method that is used by Google Blocks as well, so the user will be familiar with it.

The position is calculated as follows:

```
Vector3 positionDiff = (posLeft + posRight) / 2f - initPos;
transform.position = initWorkspacePosition + positionDiff;
```

[1]https://vrtoolkit.readme.io/

[2]https://assetstore.unity.com/packages/templates/systems/steamvr-plugin-32647

`PosLeft` and `posRight` are the current controller positions and `initPos` is the average position between the two controllers at the start of the action. The workspace is an object that contains all objects that can be moved, including the model and visualizations such as lines. `InitWorkspacePosition` is the position of that object at the start of the action.

The rotation is calculated in the following way:

```
Vector3 direction = posRight - posLeft;
Quaternion rotationDiff = Quaternion.FromToRotation(initDirection,
        direction);
transform.rotation = initWorkspaceRotation * rotationDiff;
```

`PosLeft` and `posRight` are again the current controller positions and `initDirection` is the difference between the controller positions at the start of the action.

The scale is adjusted as follows:

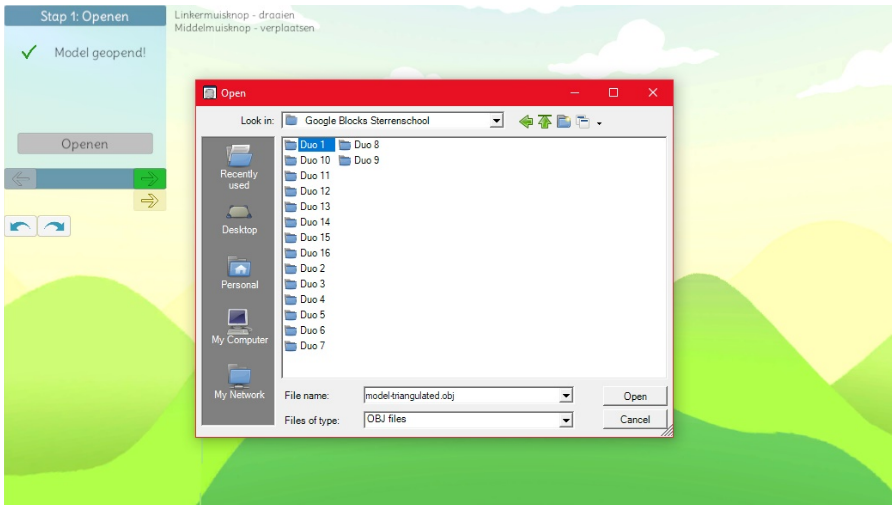
```
float scale = (posRight - posLeft).magnitude;
float scaleDiff = scale - initScale;
float newScale = initWorkspaceScale + scaleDiff;
transform.localScale = new Vector3(newScale, newScale, newScale);
```

`InitScale` is the initial length between the two controllers and `initWorkspaceScale` is the initial scale of the workspace object.

4.2 Step 1: Import OBJ

Google Blocks saves their models in OBJ format, so we need to be able to import that. A line in the file starts with one of the following:

- `v`: vertex coordinates in the form (x, y, z [,w]).
- `vt`: texture coordinates in the form (u, v [,w]).
- `vn`: vertex normals in the form (x, y, z).
- `f`: faces in the form (v1/vt1/vn1 v2/vt2/vn2 v3/vt3/vn3), where v1 is the vertex coordinate index of the first vertex, vt1 is the texture coordinate index of the first index, vn1 is the vertex normal index of the first index etc.



File name: model-triangulated.obj
Files of type: OBJ files

FIGURE 4.4: Importing an OBJ file.

- `mtllib`: specifies where the material library (.mtl file) is located.
- `usemtl`: indication to switch materials to the one with the provided name.

The OBJ importer in our software is a simplified version of Aaro4130's Runtime OBJ Importer[3]. The most important difference is that no separate vertices are created for each face, so faces that are neighbors share the same vertex. This was initially separated because face vertices can have different normals and texture coordinates, so each vertex was duplicated per face. This makes it very expensive to calculate connectivity, and normals and vertex coordinates are not needed in our case, so they are ignored. The materials are also imported to keep the visuals consistent compared to Google Blocks.

4.3 Step 2: Connectedness analysis

To analyze the connectedness of the model we iterate over every vertex in the model and group them into components and groups. We define the overall structure as follows:

1. Vertices are part of triangles
2. Vertices that are connected with triangles form a component
3. Components that intersect each other form a group

Unity meshes are defined with several arrays:

[3]`https://assetstore.unity.com/packages/tools/modeling/runtime-obj-importer-49547`

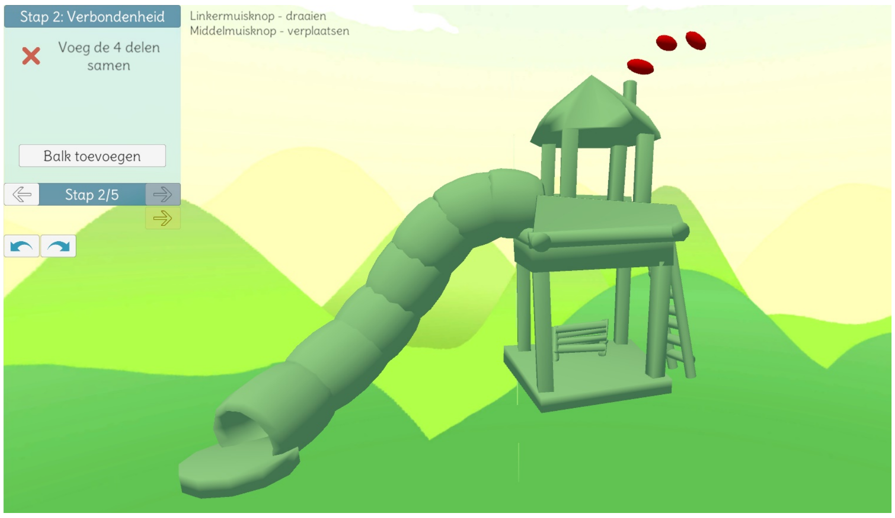


FIGURE 4.5: Connectedness analysis. Disconnected pieces are colored red.

- `Vector3[] vertices`: an array of vertices containing their position
- `int[] triangles`: every three elements describe a triangle, where every element is an index in the vertices array
- `Vector3[] normals`: normals per vertex. This array is exactly as long as the vertices array.
- `Color[] colors`: colors per vertex. This array is exactly as long as the vertices array.

Several steps are involved with finding the components and groups. First of all a 'neighbors array' is created with for every vertex a HashSet (as defined in C#) of integers. These HashSets are filled with the vertices that neighbor the vertex. This is calculated by iterating over every triangle. Every three vertices of the triangle are neighbors of each other, so each triangle adds 6 connections (from 'a' to 'b' and from 'b' to 'a', from 'b' to 'c' and back, and 'a' to 'c' and back).

Secondly we find the triangles that neighbor each vertex. This array also has a HashSet for every vertex, but instead of containing vertex indices inside the HashSet, it contains triangle indices. To fill this array with the appropriate data we again iterate over every triangle. The triangle index is added to the HashSet of every vertex in that triangle. This information will not be used for the rest of the connectedness analysis, but is used for strength analysis.

Now we have the data we need to calculate the components. Components are groups of connected vertices, where the connectedness is defined by the triangles (see Algorithm 1).

Algorithm 1 CalculateComponents

```
mark every vertex as unvisited
for all vertices, if unvisited do
    create new component
    enqueue current vertex for traversal
    while there is a vertex enqueued for traversal do
        remove vertex from queue
        add all unvisited neighbors to queue
        mark current vertex as visited
        add current vertex to component
    end while
end for
```

The next step is to intersect all components with each other to form the groups, but to do that quickly we first calculate the axis aligned bounding box of each component. These are saved in an array of `UnityEngine.Bounds`, which consist of a Vector3 center and a Vector3 extents, but can also be defined by setting the minimal and maximal point of the AABB. Calculating these is straightforward: we simply iterate over every vertex in the component and find the minimum and maximum x, y and z coordinates.

Before we calculate the groups, we also find and save the triangles per component. We do that now once for every component, so we do not have to do it later when we are iterating multiple times over every component.

After that we can find the neighbors of each component. This algorithm iterates over each pair of components and checks if their bounding boxes intersect (see Algorithm 2). If they do, a more precise intersection check is performed that checks each triangle of the first component to see if it intersects with any triangle from the other component. If any triangle intersects any other triangle from the other component, they are considered neighbors and their component index is added to the neighbors list. The triangle-triangle intersection algorithm (the function ComponentsIntersect) is based on Möller [1997].

Algorithm 2 CalculateComponentNeighbors

```
create a list of integers for every component
for i = index of every component do
    for j = index of every component do
        if j ¡= i then
            continue
        end if
        if boundingbox[i] intersects boundingbox[j] then
            if ComponentsIntersect(i, j) then
                add j to neighbors[i]
                add i to neighbors[j]
            end if
        end if
    end for
end for
```

Finally we can take the components that neighbor each other and put them into groups (see Algorithm 3). Ideally a model will have only one group, that means that everything is connected. If there are multiple groups, the user has to correct the model either using the correction tools provided or by going back to Google Blocks. The algorithm is similar to the one used for finding components.

Algorithm 3 CalculateGroups

```
mark every component as unvisited
for all components, if unvisited do
    create new group
    enqueue current component for traversal
    while there is a component enqueued for traversal do
        remove component from queue
        add all unvisited neighbors to queue
        mark current component as visited
        add current component to group
    end while
end for
```

For easy recognition the biggest group is colored green while all other groups are colored red. This is only an indication; it is still possible to edit the model so that another group is kept or several groups are combined.

4.4 Step 3: Balance analysis

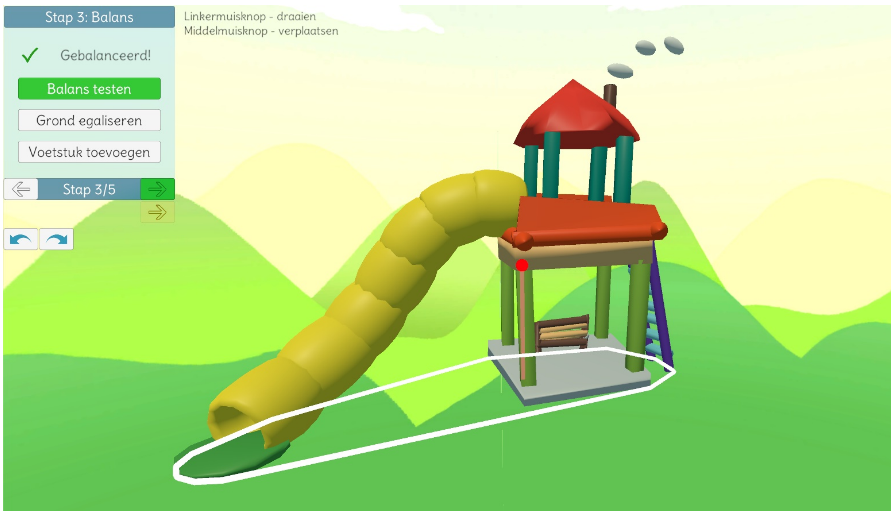


FIGURE 4.6: Balance analysis. The convex hull of the floor vertices is colored white. The center of mass is the red dot. If the center of mass (projected to the floor) is within the convex hull, the model is balanced.

To find out whether a model is balanced we use a combination of a convex hull of the floor vertices and the center of mass of the whole model. If the center of mass (projected onto the floor) is within the convex hull we conclude that the model is balanced. The calculation involves several steps.

The first step is to find out which vertices belong to the floor. We first find the lowest vertex and use that as a baseline. Then we find every vertex with a height difference less than a predefined threshold. These are put into a Vertex2 array. These are random points on the plane, this is not yet an outline which we need. The next step is to find the convex hull of those points. We use the Graham Scan algorithm for this, as it performs well and is relatively easy to implement [Graham, 1972]. The convex hull is drawn with white lines underneath the model when the user is at step 3.

In order to calculate the center of mass, we opted to mimic the way Blender calculates the center of mass[4]. It takes the weighted average of all the face centroids, where the weight is equal to the area of the faces. This results in the center of mass of an object that is hollow, which matches the way 3D printers usually only fill in 10 to 20% of the insides of models in order to save material and speed up the process.

As precalculation we calculate the face centroids and the face areas and put them into arrays (Vector3[] and float[] respectively). The following code is run for all triangles. v1, v2 and v3 are the vertices of the triangles.

[4]`https://troymcconaghy.com/2015/06/25/how-blender-calculates-center-of-mass/`

```
//Centroid is the means of the vertices
Vector3 faceCentroid = (v1 + v2 + v2) / 3;

//Area using Heron's formula
float a = (v1 - v2).magnitude;
float b = (v2 - v3).magnitude;
float c = (v1 - v3).magnitude;
float s = (a + b + c) / 2;
float faceArea = Mathf.Sqrt(s * (s - a) * (s - b) * (s - c));
```

To get the center of mass we add all face centroids times their area and we divide that by the sum of all areas. To find out if the model is balanced we adapted the PolyContainsPoint code from the Unity wiki[5]. This returns a boolean that when true means that the given point is inside the given polygon, which means our model is balanced. The center of mass is drawn as a red dot when the user is at step 3, with a transparent red line to the floor to quickly give the user an idea how high the center of mass is above the floor.

4.5 Step 4: Strength analysis

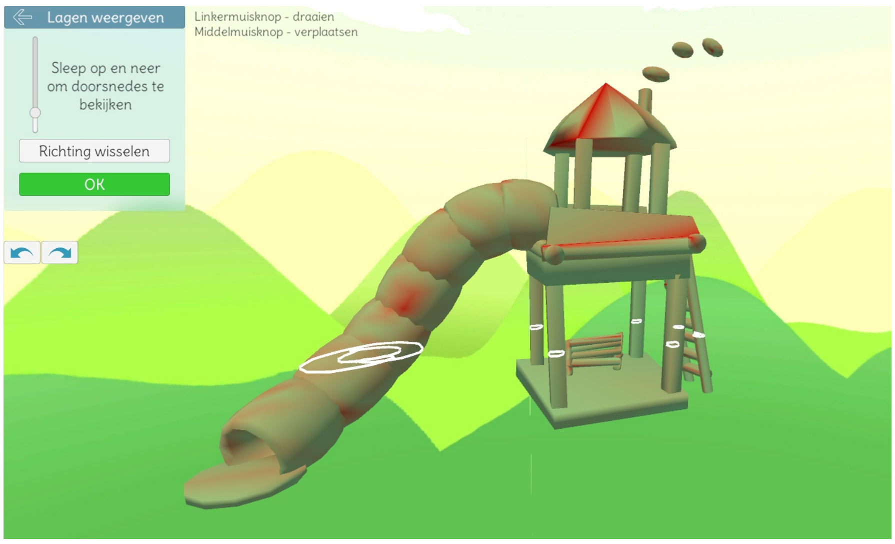

Lagen weergeven
Linkermuisknop - draaien
Middelmuisknop - verplaatsen
Sleep op en neer om doorsnedes te bekijken
Richting wisselen
OK

FIGURE 4.7: Strength analysis. Thin pieces are colored red. A cross section is colored white.

[5]http://wiki.unity3d.com/index.php/PolyContainsPoint

To find problematic areas of strength in the model, many cross sections are calculated and analyzed. The amount of cross sections depends on the crossSectionDistance parameter and the size of the model. For the x axis the amount of cross sections is calculated in the following way:

$$count = (highestX - lowestX)/crossSectionDistance$$

The highest and lowest value of each axis is calculated upon loading the model in the PreAnalysis class. At that point the model is also normalized to be 1 unit high. Or rather, the depth, width and height are compared and the longest side gets normalized to 1 unit, the other sides scale with it to be less than 1 unit. With a crossSectionDistance value of 1f and a model 1 unit wide, 100 cross sections will be taken. The same is done for the y axis and the z axis.

The height of an individual cross section is determined as follows:

$$height = lowestX + i * crossSectionDistance$$

where i is the index of the loop.

Cross sections can consist of multiple disconnected parts. To be able to find each part we need to iterate over every triangle in the mesh, see Algorithm 4. We perform an efficient triangle-plane intersection on the triangle. This method quickly returns if the triangle does not intersect the plane. If the triangle does intersect the plane, we get a list of intersection points. Usually we get two points, but in the case of a coplanar triangle that is the same height as the plane, we get three points. In the case the triangle does intersect the plane, we know we have found a new cross section part.

Before we continue with the next triangle, we will first find the whole cross section part. First we find all points that intersect with the plane and are connected to the intersection points we just found. This is done in FindConnectedGraph, which is explained in detail in the next paragraph. Because the mesh can be non-manifold, this is an essentially random set of points on the plane, but we only want the outline. This is calculated in FindOutline, which is described below. The cross section part is now complete, so we add it to the cross section, which is a list of cross section parts. Finally we mark each point in the cross section part as visited, so we can skip them in the original iteration at the start of the method.

Algorithm 4 CalculateCrossSection(axis, height)

```
make new cross section (list of cross section parts)
for all triangles do
    if triangle has been visited then
        continue
    end if
    set triangle as visited
    points ← TrianglePlaneIntersection(triangle, axis, height)
    if there are points then
        make new cross section part (list of points)
        graph ← FindConnectedGraph(points)
        crossSectionPart ← FindOutline(graph)
        for all points in graph do
            set point to visited
        end for
        add crossSectionPart to crossSection
    end if
end for
return cross section
```

In order to find all the points that are part of a cross section part, we work with a queue (see Algorithm 5). In each iteration we take a point from the queue, find the triangles that neighbor that point and intersect those triangles with the plane. The resulting new points are added to the graph and the queue. To find the triangle neighbors of the intersected point with the plane, each intersected point is a struct that remembers the two points of the triangle it came from. Now we find a list of triangles that contain the first point, and a list that contain the second point. If we intersect those two lists, we get the triangles that neighbor the edge, which are the triangles that neighbor the intersectionpoint. Each point also remembers their neighbor points inside the struct, which is used to find the outline.

Algorithm 5 FindConnectedGraph(startingpoints, axis, height)

```
make new graph (set of points), add startingpoints
make new queue, add startingpoints
while queue is not empty do
    point ← dequeue
    triangleNeighbors ← find triangles that neighbor the line that the intersectionpoint originates from
    for all triangleNeighbors do
        newPoints ← TrianglePlaneIntersection(triangleNeighbor, axis, height)
        if there are no points then
            continue
        end if
        for all newPoints do
            if graph does not contain newPoint then
                add newPoint to graph
                add newPoint to queue
            else
                update neighbors list of newPoint equivalent in graph
                update neighbors list of all existing neighbors of newPoint
            end if
        end for
    end for
end while
return graph
```

The graph of a cross section part that contains all the connected points is not always equivalent to the outline. The mesh can be non-manifold, in particular in the case of brushes in Google Blocks. This tool allows the user to create a tube of connected convex regular polygons (polygons that are equiangular and equilateral). With every step that a new polygon is added to the tube, the previous polygon (which is now invisible from the outside) is kept. This means that some edges of the tube have three connecting triangles instead of two (see figure 4.8).

To make sure that the strength can still be calculated regularly for non-manifold pieces of the cross section, only the outline of the graph is kept. To find the outline of a graph we start with the lowest point in the graph, and traverse the graph counter-clockwise (see Algorithm 6). Only points that are connected in the graph with the current point are considered as next point of the outline in each iteration. The point with the highest clockwise angle between the previous, current and potentially next point is chosen. For

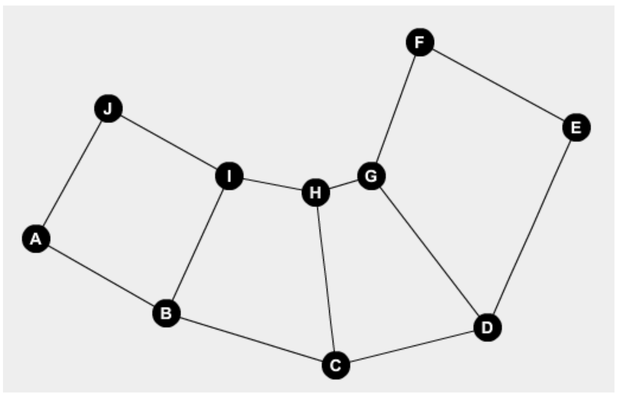


FIGURE 4.8: Graph of a non-manifold cross section part.

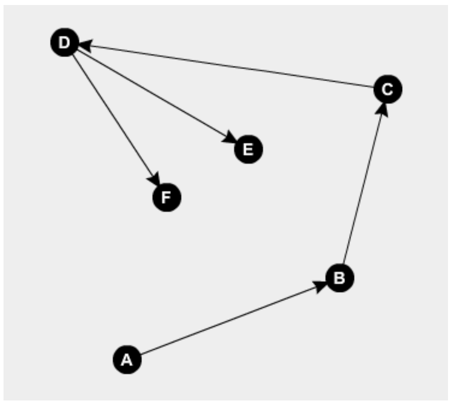


FIGURE 4.9: The process of finding an outline.

example, in figure 4.9 the clockwise angle between C, D and E is compared to the angle between C, D and F. The latter has a higher clockwise angle so F is chosen as next point of the outline.

Algorithm 6 FindOutline(graph, axis)

```
make new outline (list of points)
startPoint, currentPoint ← lowest point in graph (lowest y if plane axis is x or z,
lowest z if plane axis is y)
previousPoint ← 1 unit to the right of currentPoint
while true do
    maxAngle ← negative infinity
    for all neighbors of currentPoint do
        calculate clockwise angle between previousPoint, currentPoint and neighbor
        if angle > maxAngle then
            maxAngle ← angle
            nextPoint ← neighbor
        end if
    end for
    previousPoint ← currentPoint
    currentPoint ← nextPoint
    if currentPoint = startPoint then
        break
    end if
    add currentPoint to outline
    if number of points in outline ≥ number of points in graph then
        break
    end if
end while
return outline
```

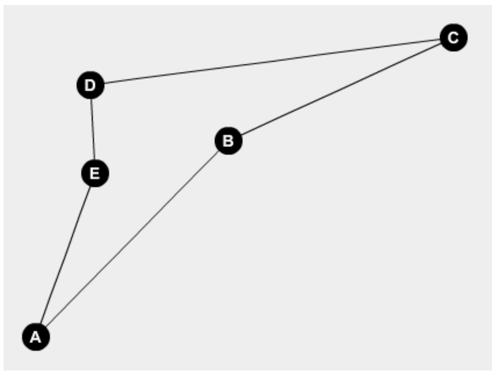


FIGURE 4.10: Graph of a cross section outline with sharp angles.

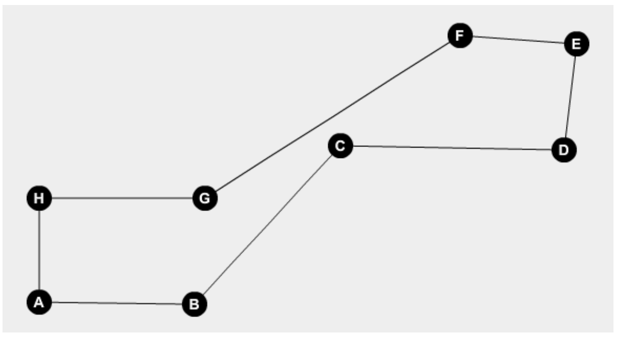


FIGURE 4.11: Graph of a cross section outline with a thin area without sharp angles.

Several cross section parts can overlap, so we looked at combining all cross section parts with a boolean operation. Due to time constraints we could not implement our own boolean operation, such as one based on Mei and Tipper [2013], so we looked at plugins available for the Unity engine. After trying several plugins, including SabreCSG by Sabresaurus Ltd[6] and CSG by Alan Baylis[7], this proved too difficult to implement within the available time, so the current version of our program does not include the union of cross section parts.

To calculate the strength per cross section part, we assign a weakness value to each point of the outline. We look for weak points that result in thin areas in two ways: sharp angles (figure 4.10) and distance between points and lines (point C in figure 4.11). If the angle between the previous, current and next point on the outline is too sharp, we add to the weakness a value that depends on the sharpness of the angle.

After that we compare each point to all lines. If the line is close to the point and it does not go in the direction of the point, the point gets a weakness value depending on the distance to the line. Finally, the weakness values are added together and it is stored in the source vertices: the vertices in the model that the cross section point originates from. The weakness is divided by two (each point has two source vertices) and scaled depending on the distance between the point and the source vertex.

Each vertex in the model now has a weakness value. Using a vertex color shader the vertices are colored from green to red depending on the weakness value. This allows the user to quickly spot where the model is weak, and correction tools are given to improve those areas. To check if the model gets approved or not, we loop over every vertex and find its weakness value. If any of these values are higher than the approvedWeakness parameter, the model is rejected. It also keeps count how many vertices exceed this value, and prints the number to the console.

[6]https://assetstore.unity.com/packages/tools/modeling/sabrecsg-level-design-tools-47418
[7]https://assetstore.unity.com/packages/tools/modeling/csg-82197

TABLE 4.2: Strength analysis parameters.

Name	**Type**	**Value**	**Description**
crossSectionDistance	float	0.01f	Distance between cross sections. Determines how many cross sections to calculate depending on mesh size.
sharpAngle	float	60f	Angle between two lines of a cross section that is considered a sharp angle and will create thin areas.
sharpAngleWeight	float	1f	How much the angle weighs towards weakness calculation. This is the maximum amount that gets added to weakness.
thinDistance	float	0.005f	Distance between lines in a cross section that is considered thin.
thinDistanceWeight	float	1f	How much the distance between two lines weighs towards weakness calculation.
approvedWeakness	float	100f	Maximum weakness a vertex can have before the mesh is rejected.

4.6 Step 5: Export to STL

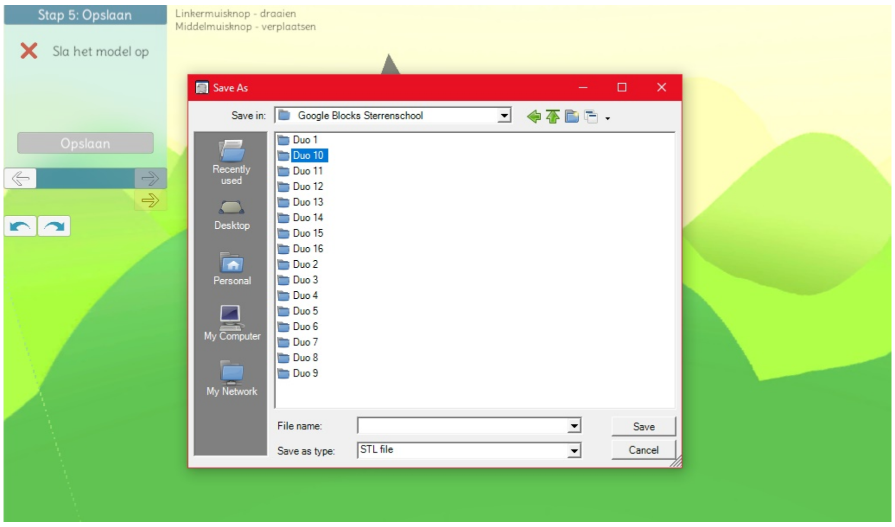


FIGURE 4.12: Exporting to STL format.

If and only if all the analyses approve the model, the model can be saved and exported to a 3D printable format. The STL format is chosen because of its universal use in 3D printing. An ASCII encoded STL file starts with the line

```
solid [name]
```

Then for each triangle the following lines are added:

```
facet normal n_i n_j n_k
    outer loop
        vertex v1_x v1_y v1_z
        vertex v2_x v2_y v2_z
        vertex v3_x v3_y v3_z
    endloop
endfacet
```

And the file is ended with:

```
endsolid [name]
```

Note that the normal for each triangle is required, which should be a unit vector pointing outward. In most cases when the normal is set to 0, the software will calculate the normal using the right-hand rule. Some software even ignore the normal entirely and always calculate it. In addition, vertex information cannot be shared. Since 3D printers require manifold objects, every vertex will be used at least twice. In conclusion, the STL format contains a lot of redundant data while not supporting colors or textures. For this reason the size of an STL file will be excessive when used with a model with thousands or millions of triangles.

The binary encoded version of STL reduces the file size drastically. It starts with an 80-character header which is usually ignored. After that are 4 bytes for an unsigned integer in little-endian order. This contains the number of triangles in the model. Each triangle is described by 50 bytes: twelve 4-byte floating-point numbers and 2 bytes for the 'attribute byte count', which should be zero. The first 3 numbers of the twelve floating-point numbers describe the normal and the other 9 describe the X, Y and Z coordinate for each of the 3 vertices of the triangle. After the last triangle is described the file ends. The full file can be described like this:

```
UINT8[80] - Header
UINT32 - Number of triangles

foreach triangle
REAL32[3] - Normal vector
REAL32[3] - Vertex 1
REAL32[3] - Vertex 2
REAL32[3] - Vertex 3
UINT16 - Attribute byte count
end
```

4.7 Corrections

To make it easier for the user to get their model approved, a number of basic corrections have been implemented. These are only there to help the user in simple cases, and are not guaranteed to be sufficient to get the model approved by the program. For more advanced corrections the user is expected to go back to Google Blocks. Furthermore these corrections are not the most robust, occasionally break and sometimes even crash the program. Implementing the analyses correctly was a higher priority, and the corrections were considered stretch goals from the beginning. Nevertheless they did prove useful in the experiment, especially the balance related corrections. Table 4.3 shows an overview of the corrections, whether they work in and outside of VR and their related analyses.

These corrections are implemented according to the command pattern, which allows for undo and redo functionality. In VR this can be accessed with the touchpad on the right controller, where pressing the left side of the touchpad activates the undo functionality, and the right side activates the redo functionality. Outside of VR there are two buttons in the interface with curved arrows which do the same.

Table 4.3: Correction overview.

Correction	**In VR**	**Outside VR**	**Related analyses**
Delete component	Yes	No	Connectedness
Flatten ground	Yes	Yes	Balance
Add pedestal	Yes	Yes	Balance
Add strut	Yes	No	Connectedness and strength
Thicken	Yes	No	Strength

4.7.1 Delete component

This correction deletes components (Dutch: Deeltje verwijderen). Especially useful in the connectedness step, in the case the user forgot to remove a floating piece in Google Blocks. When the trigger is pressed on the right controller, it finds the nearest vertex with respect to the top of the controller. Then it finds the component that the vertex belongs to, as calculated during the connectedness analysis phase. It deletes the triangles and vertices that belong to the component.

Doing this causes problems, however, because triangles store vertices by their index, and a lot of these indices are now shifted. To solve this problem we update the triangle references every time we remove a vertex. This is not the most performant way and unfortunately did not solve the problem entirely: some triangles were still being removed that should not be removed. Due to time constraints no other solution was found nor implemented. Ultimately this correction did not prove robust enough and was hidden from the program entirely.

4.7.2 Flatten ground

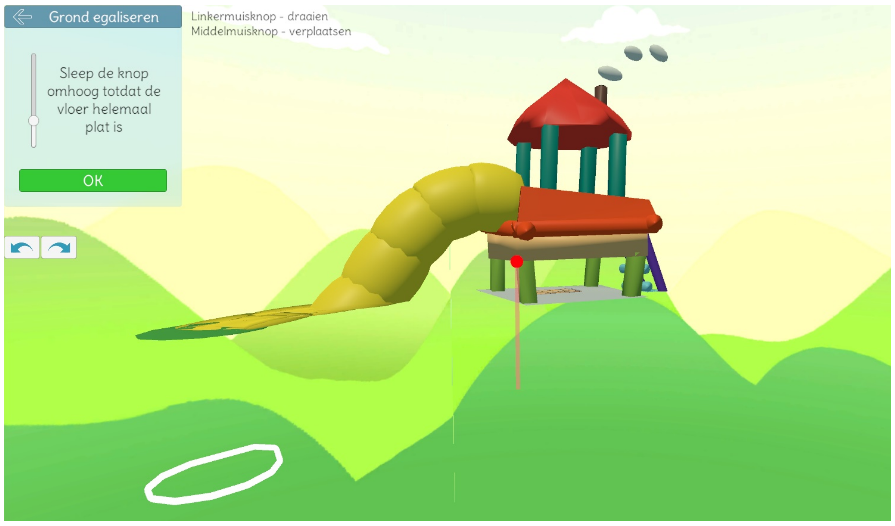

Grond egaliseren
Linkermuisknop - draaien
Middelmuisknop - verplaatsen
Sleep de knop omhoog totdat de vloer helemaal plat is
OK

FIGURE 4.13: The flatten ground correction.

This correction flattens the ground for balance purposes (Dutch: Grond egaliseren). The user is presented a slider which can be dragged up to flatten more of the model. Any time the slider is moved, the program will look through all the vertices of the model, and move the vertices that are lower than the threshold up. The slider works relative to the height of the model. It represents the percentage of model that will get flattened: when at the bottom, no vertices will be moved and when at the top, all vertices will be moved to the top y value.

4.7.3 Add pedestal

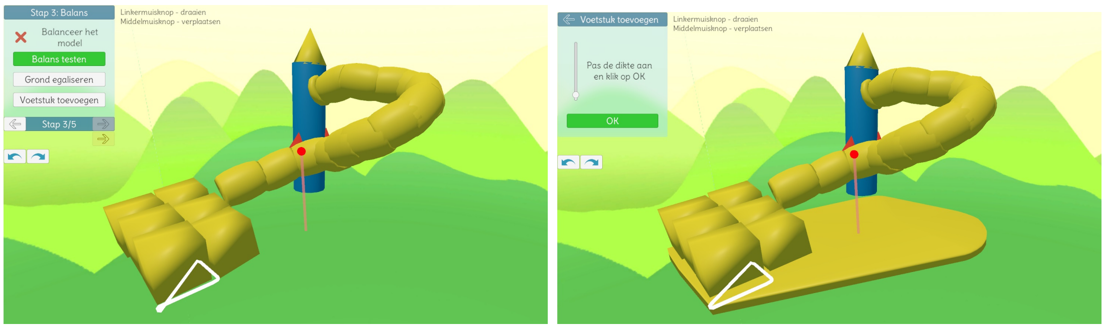


FIGURE 4.14: Adding a pedestal. Left: model without pedestal. Right: model with pedestal.

This correction adds a pedestal for balance purposes. It works with a similar slider to the one used in the flatten ground correction, which this time determines the height of the pedestal. On the corrections activation a convex hull is made from the whole model, not just the ground vertices like in the balance analysis. This guarantees the pedestal touches the model and the model will be balanced. It is not guaranteed to be strong enough, this will be inspected in the strength analysis step. The center of the convex hull is calculated as well. Every time the slider is moved, the old pedestal is removed and new vertices and triangles are added to the model. The sides of the pedestal are created first and then the top and bottom are created with a triangle fan with the previously calculated center of the convex hull as common point.

4.7.4 Add strut

This correction allows the user to add struts in VR. It is used for connecting loose components in the connectedness step and for strengthening the model. When the trigger is pressed on the right controller a strut is made on the top of the controller. This defines the starting point. As long as the trigger is held down, the user can move the endpoint. This gets visually updated in realtime, and as soon as the trigger is released the strut is finalized and the model is reanalyzed.

4.7.5 Thicken

The final correction allows the user to thicken parts of the model. This correction also only works in VR. When the user holds the trigger of the right controller, vertices close to the top of the controller gets pushed out. The longer the user holds the trigger, the more they are pushed out. This is visible in realtime for the user in VR.

4.8 Balance simulation

One of the requests from Stichting KleurInCultuur was to add a visual means to show the children why an unbalanced model would not work in the real world. We decided to add a button that makes the model fall over if it is unbalanced (Dutch: Balans testen). This is easier to understand for children than the white line and red dot in the balance analysis step.

It is implemented as a correction, because that allows for easy switching of modes and already has activation, update and finish code within. On activation it searches for the closest point on the convex hull compared to the center of mass. This is the point the model will be rotated around, with the axis of rotation equal to the edge of the convex hull. Every update the model gets rotated around this point more and more, with a speed that gets increased by an acceleration value. When the model has rotated 180°, it simply stops. There is a button to replay the animation, and a button to go back to the balance analysis step.

4.9 Cross section visualization

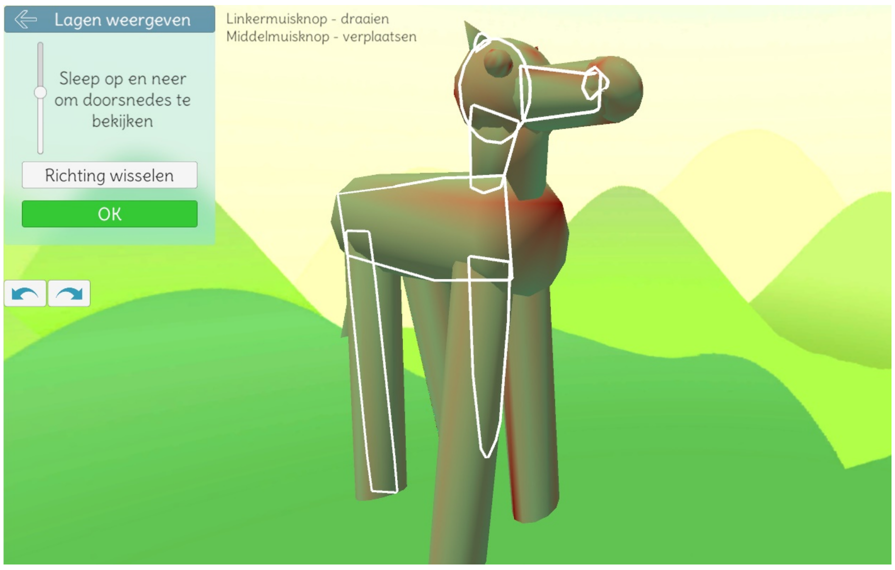

Lagen weergeven
Sleep op en neer om doorsnedes te bekijken
Richting wisselen
OK
Linkermuisknop - draaien
Middelmuisknop - verplaatsen

FIGURE 4.15: Cross section visualizer.

Mostly to debug but also to show the inner workings of the strength analysis, we added a mode to view the cross sections in any of the 3 axes on any part of the model (Dutch: Lagen bekijken). Like the balance simulation, this mode was also implemented as a correction for our own convenience. The user is presented with a slider to move the height of the cross section and a button to switch the axis. Then the cross section is visualized on the model with white lines. This is the same method that the strength analysis uses to find thin parts of the model.

4.10 Optimization

During the implementation of the connectedness analysis it became clear that some optimizations were necessary. Some of the test models (made by colleagues at ING in Google Blocks) showed an analysis time of more than 100 seconds while most other models were analyzed in under 5 seconds. After adding some code that allows for checking the time particular pieces of code take (using `Time.realtimeSinceStartup`), it became clear that the culprits were the CalculateComponents and CalculateTrianglesPerComponent methods.

In the worst case CalculateComponents took 33.3 seconds. Luckily, by switching from a List to a Queue data structure for the vertex traversal in the while loop (algorithm 1), that time was reduced to 0.91 seconds, including the profiling code mentioned earlier. Before it was changed to a Queue we continuously grabbed the first item from the List, meaning that all the other items in the list needed to be moved forward one index. This takes $O(n)$ time, whereas a Queue can retrieve and remove it's first element in $O(1)$ time.

The importance of picking the right data structure was shown once more in the CalculateTrianglesPerComponent method. In the worst case this method took 81.4 seconds to complete, which was reduced to 0.018 seconds (!). The code to check if a vertex is in a component changed from linear to constant time by storing components in HashSet form instead of Lists. This piece of code is performed for every triangle for every component, so that performance increase adds up.

The total connectedness analysis time of the model that took the longest was reduced from 113.64 seconds to 0.96 seconds. Other models were sped up from 4.13 seconds to 0.11 seconds, and from 2.44 seconds to 0.19 seconds. This performance improvement allowed for a better balance and strength analysis.

Chapter 5

Experiment

5.1 Setup

To answer research question **Q1b** an experiment is performed to test our program. Children are asked to design playground equipment in Google Blocks, after which these models are analyzed and 3D printed. Then the prints are tested for desired properties, as defined in the answer to **Q1a**. The children are not told what those desired properties are beforehand, in order to get models with a variety of issues that properly test our program.

After approaching a few elementary schools in Almere we chose Sterrenschool de Ruimte based on their enthusiasm and availability of desired dates. Three sessions were held in the first two weeks of the schoolyear in which the 35 children got to experience working with virtual reality. This was done in groups of two and occasionally three children, forming 16 groups in total. In those three sessions the children had 15 minutes per group to complete their assignment for that day. Class 8a and 8b were chosen for this project, containing children of the approximate age of 12 years old.

The first session was an introduction to VR and Google Blocks, getting familiar with the controls and thinking about what they wanted to create next time as their playground equipment. The second session was dedicated to creating their playground equipment in Google Blocks and finishing their design. In the third session they used our analysis program to analyze their own models for 3D printability.

The experiment consisted of the following parts:

- Session 1 (4 September): Introduction to Google Blocks, start on designing playground equipment.

- Session 2 (6 September): Finish playground equipment in Google Blocks.
- Session 3 (11 September): Children use our program to analyze their own models.
- Print the designs (12 September - 8 October): Print and test the prints for desired properties.
- Open day (21 September): Help Stichting KleurInCultuur with showing VR at the elementary school.
- Session best design (9 October): Hand out the prints and discussion in class to choose the best design.

FIGURE 5.1: Open day at Sterrenschool de Ruimte.

5.2 Session 1

In the first session the children were introduced to our project and Google Blocks.

8:30 - Set up VR sets in Auditorium. Laptops and VR sets were provided by Stichting KleurInCultuur.

9:00 - Introduction in class. Amanda from Stichting KleurInCultuur introduced the class to our project. The children made groups beforehand.

9:30 to 12:15 - Group sessions in the Auditorium. We had two groups per 15 minutes, one supervised by me and the other by Amanda. Amanda introduced the kids to three painters: Mondriaan, Kandinsky and Dalí. Then the children chose one of them and got to make playground equipment in the painter's style. Each kid got 5 minutes to play in VR and try Google Blocks after which the other kid of the group could go as well. In the case of groups of 3 they all got 3 minutes.

During the day we noticed that there was very little time for each child to work in VR. Some groups started with their playground equipment already but others were still struggling with the basics. We mostly spent time explaining the following features of Google Blocks: placing a block, dragging to create bigger blocks, switching to different primitives (sphere, block, pyramid). We noticed that the eraser tool was very intuitive, all children immediately understood they had to touch the objects they wanted to erase and then press the trigger button.

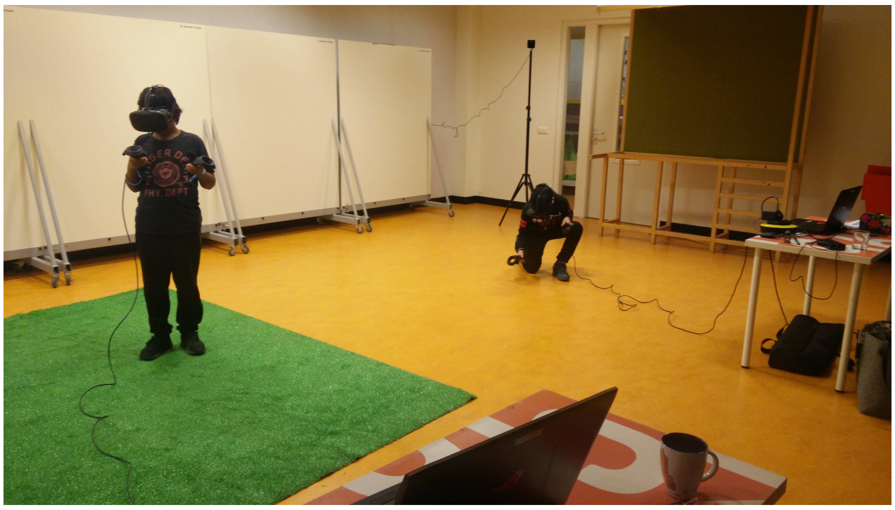


FIGURE 5.2: Session 1 at Sterrenschool de Ruimte.

5.3 Session 2

In the second session a few days later the children continued with Google Blocks to create a playground equipment design they made up.

8:30 - Setting up VR sets in Auditorium.

9:00 - 11:45 - Group sessions. We had two groups per 15 minutes again, one supervised by me and the other by Mannie from KleurInCultuur.

We noticed it went a lot better than the first session, the majority knew immediately what to do and managed to create some good models. Some would turn out very hard to print but that was part of the plan. We purposefully did not explain to the children what properties are necessary to 3D print their model so that we could get models with a variety of issues that properly test our program.

The models the children produced can be found on Poly, a website to share 3D assets for VR and AR made by Google: `https://poly.google.com/search/sterrenschool%20les%202`

5.4 Session 3

In the third session the children used our program to analyze their designs and correct them. Corrections were made using our program in simple cases and in Google Blocks in other cases.

8:30 - Setting up VR sets in the Auditorium.

9:15 - Explanation of the program in class. We gathered class 8a and 8b in one room and showed our program on the big screen (in 2D). We explained the program step by step so the children would not be overwhelmed when they would see our program in VR for the first time in their limited timeslots.

9:30 - 13:30 Group sessions. This time we had one group come over per 15 minutes. We still set up two VR sets in case some groups would exceed their timeslots. This turned out very handy as most of the time the two VR sets were in use. Some groups were done in 5 minutes, others had to go back and forth between our program and Google Blocks to fix all of the issues that our program showed them.

Some of the feedback that we gathered mentioned that the balance simulation (the falling animation) was popular, and the background was nice and looked even better than Google Blocks (thank you Jenna!). Our software hanged sometimes; the experience could be smoother with less waiting or with a better indication that the program is analyzing the model. Switching between Google Blocks and our program could also be smoother. For now we had to manually find the save files and open them in our program. Overall the program was intuitive and the children knew what they had to fix to make their model 3D printable.

FIGURE 5.3: The experiment in progress at Sterrenschool de Ruimte. Left: Amanda from Stichting KleurInCultuur.

5.5 3D printing process

After the models were gathered, analyzed by our program and exported to STL format, it was time to print them. 8 of the 16 models were deemed printable by our program, as seen in table 5.2. Since the strength analysis did not meet our intuition, we decided to try to print the models that were not approved in the strength analysis anyway. 14 of the 16 models were approved, ignoring the strength analysis.

At this point the STL files were not completely ready to print yet: they still needed printer supports. 3D printers cannot print in the air: a platform or previous layers of filament is needed below. Note that our program analyzes the properties of the completed print without the printer supports. The variety of printer supports and their effects are outside of the scope of this project and are solely used to be able to print the models.

We used two ways of adding printer supports: with Slic3r Prusa Edition [Alessandro Ranellucci, 2016] and with Autodesk Meshmixer [Autodesk, 2016]. Slic3r adds thin walls below a large part of the model, as seen in figure 5.4. Autodesk Meshmixer adds solid columns that are attached to the points of the model that need it the most, as seen in figure 5.5. These columns can also be added manually for increased precision. Both ways have their up- and downsides, and it depends on the model which one works best. Models with large flat platforms would require too many columns so Slic3r works better in that case, whereas models that require supports in only a small area of the model work better with Meshmixer supports.

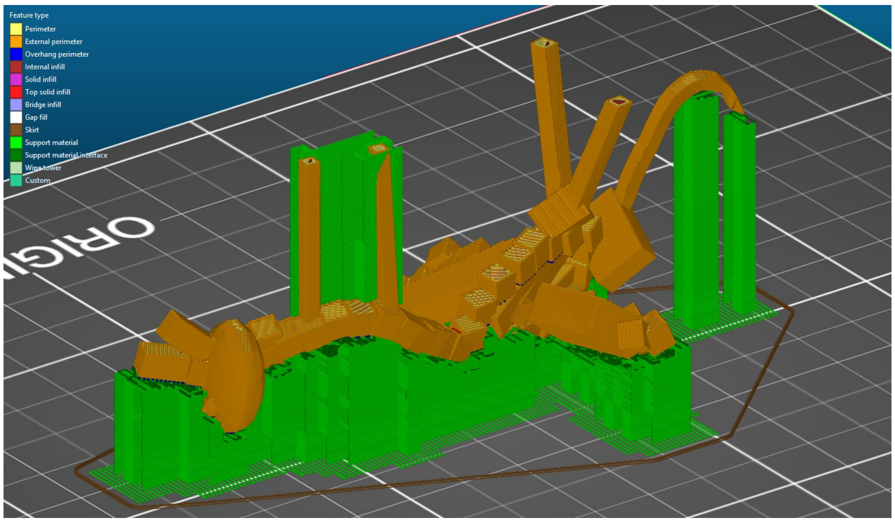


FIGURE 5.4: Adding supports in Slic3r. Model 12 (upside down).

The STL files were first opened in Autodesk Meshmixer and scaled using the Analysis →Units/Dimensions tool. The longest axis was made 10 centimeter. Then some models get printer supports inside Meshmixer, this is done with the Analysis →Overhangs tool. Some of the settings were adjusted based on what works best for that model. After that all models were exported to STL format.

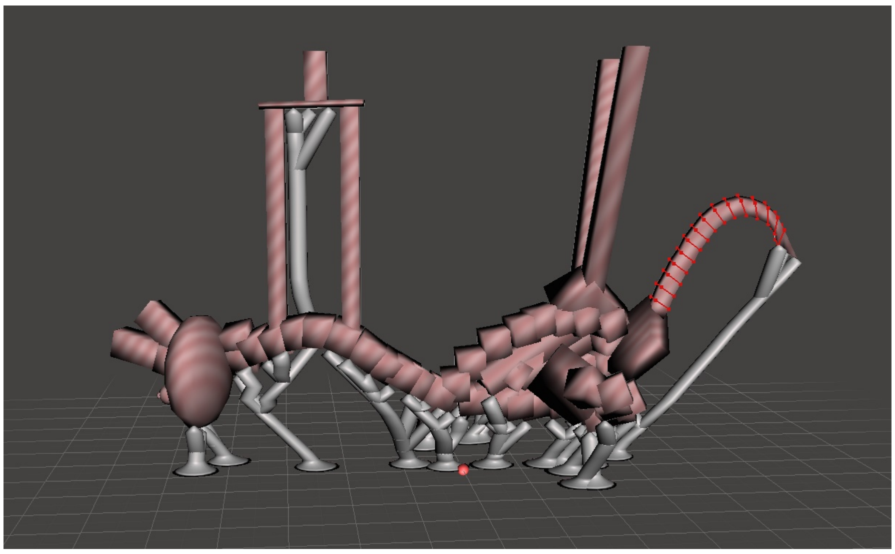


FIGURE 5.5: Adding supports in meshmixer. Model 12 (upside down).

These STL files exported from Meshmixer (and properly scaled) were then imported in Slic3r Prusa Edition. The models that did not get Meshmixer supports will now get supports. Layer height is kept at 0.15mm and fill density is kept at 20% for all models. Interface layers were set to 0 in cases where the supports would be difficult to remove. The supports were visually inspected using the 'Slice now' feature and the Preview panel. The model can be viewed layer by layer here, so it is easy to check if there are areas that are still missing supports. If everything is correct it can be exported to G-code, a format that contains the travel path for the printer head, the speed and other commands that the printer understands.

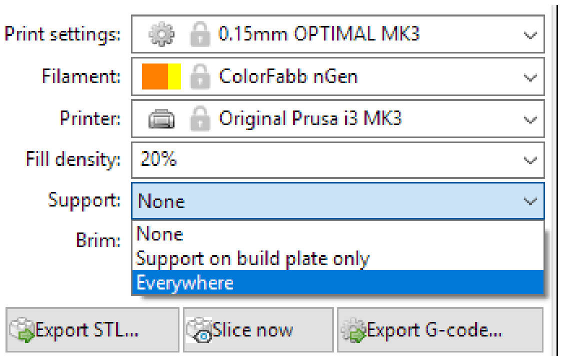

Print settings: 0.15mm OPTIMAL MK3
Filament: ColorFabb nGen
Printer: Original Prusa i3 MK3
Fill density: 20%
Support: None
Brim:
None
Support on build plate only
Everywhere

FIGURE 5.6: Slic3r Prusa Edition settings.

The models were printed with the Original Prusa i3 Mk3 printer using the ColorFabb nGen orange filament over the course of 3 weeks. This filament operates at a printer head temperature of 240 °C and a platform temperature of 85 °C. Strength-wise it is in between the commonly used ABS and PLA filament types. It is not as tough as ABS but also does not come with the unpleasant fumes during printing.

9 of the 14 connected models were printed correctly the first time. 4 models needed a second print to improve some rough areas by adding more supports and changing other settings. One model needed to be printed 5 times before the print succeeded. That was model 12, and eventually succeeded by printing it upside down with Slic3r supports.

5.6 Discussion of best design

In this final session the prints were handed out and the class chose a best design. This design might be 3D printed in real size in a future project.

10:45 - Introduction to the class together with Amanda from Stichting KleurInCultuur.

10:50 - Handing out the prints. Calling out the names of the groups one by one, then one of the group comes forward and receives their print.

11:00 - Every group prepares a pitch presentation. They describe what their playground equipment represents, what you can do with it, what painter style was used (Kandinsky, Mondriaan or Dalí), and why they think their playground equipment is the best.

11:10 - Every group holds their pitch presentation.

11:30 - The class gets to vote on their favorite design. This is done using pieces of paper on which they write the name of their favorite design, in order to eliminate group pressure.

11:40 - 11:50 Amanda and I count the votes and announce the top three. The winner is model 9 (as seen in table 5.1).

Model	**Votes**
Model 9	14
Model 13	5
Model 2	4
Model 7	3
Model 1	2
Model 11	2
Model 5	1
Model 15	1

TABLE 5.1: Votes from the children.

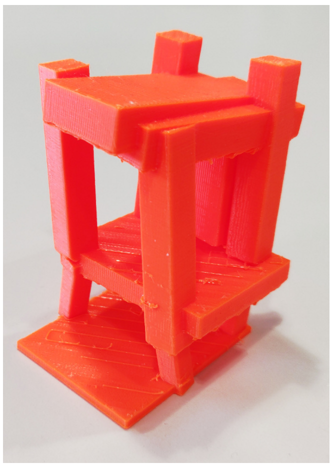


FIGURE 5.7: Model 9, the best design as voted by the class.

5.7 Results

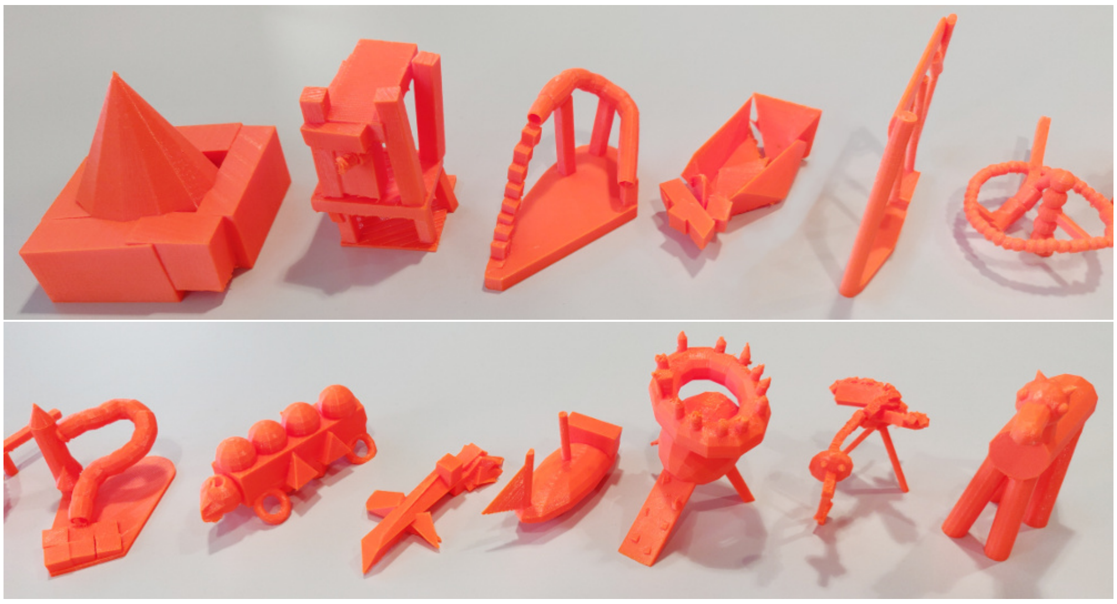


FIGURE 5.8: Printed designs.

7 of the 16 models were approved by our program. Since unbalanced and weak models can still be printed, we tried to print all connected models: 14 of the 16. Of these 14 models, 7 were correctly printed without any issues and 7 were printed with minor issues. All models where able to be printed. See table 5.2 for the results per model. The columns under 'Program' indicate the analysis results of our program on the model.

The columns under 'Tests on printed model' indicate the results of the physical tests we performed on the models, as described in the following paragraph.

The prints were tested on connectedness, balance and strength. Connectedness was tested by observing whether the print remained in one part after removing the printer supports. Balance was tested by placing the print on a flat surface in the orientation as seen in Google Blocks and our program, and observing whether or not the print falls over. If they were still standing upright after one minute the print is considered in balance.

Strength was tested by performing drop tests from a height of 75 centimeter, the height of a standard table, onto a hard surface. If the print did not break into multiple pieces and was not considerably deformed (as observed with the naked eye), the print is considered strong enough for everyday use.

Of the prints that were printed without any issues or with minor issues, 13 were connected and did not fall apart after removing the printer supports. 14 models did not fall over when placing them on a flat surface, and 7 models survived the drop test.

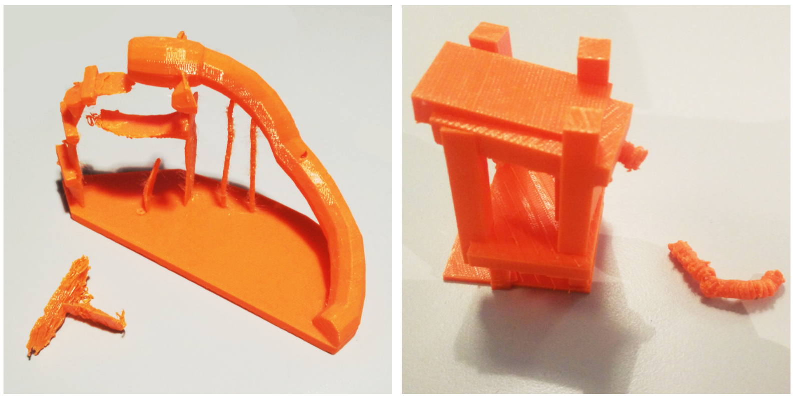


Figure 5.9: Printed models with disconnected parts. Left: Model 4. Right: Model 9.

Model 4 is one of the models that was not fully connected after removing the print supports. This is because the model contains really thin areas that the printer did not manage to print. This depends on the printer and the slicer software. These thin areas that cause issues for connectedness were already indicated in the strength analysis, so when the user fixes the strength of the model, this connectedness issue will be fixed at the same time.

Model 9 has tubes that form a slide that got disconnected during the removal of the printer supports (see figure 5.9). Being more careful on the second print resolved that issue, and this second print was used for the tests. As mentioned before, printer supports and their effects are not included in this research. In any case, that part of the print turned out not strong enough for the drop test, as expected.

All designs are balanced as can be seen in figure 5.8. 7 designs of the 14 that could be printed were not strong enough. In those cases pieces of the design broke off because of the drop. Some examples can be seen in figure 5.10.

We can compare these results to the predictions of the analyses of our program. We do this separately for each property, as seen in table 5.3. The connectedness analysis correctly predicted the outcome in 92.9% of cases, where the one time that it did not predict correctly was due to print accuracy and thin areas. We observe that the balance analysis correctly predicted everything. Still, we can not conclude it is perfect considering the sample size. We only got to test our program on 16 models which means there

TABLE 5.2: Results of the program and the physical tests per model.

Model	**Prediction of program**			**Tests on printed model**		
	Connected	**Balanced**	**Strong**	**Connected**	**Balanced**	**Strong**
1	Yes[1]	Yes[2]	No	Yes	Yes	Yes
2	Yes[1]	Yes[2]	No	Yes	Yes	Yes
3	Yes	Yes	Yes	Yes	Yes	No
4	Yes[1]	Yes	No	No	Yes	No
5	Yes	Yes[3]	Yes	Yes	Yes	Yes
6	No[5]	-	-	-	-	-
7	Yes	Yes[2]	Yes	Yes	Yes	Yes
8	Yes	Yes[3]	Yes	Yes	Yes	No
9	Yes	Yes[3]	Yes	Yes	Yes	No
10	Yes	Yes[2]	Yes	Yes	Yes	Yes
11	Yes	Yes[3]	Yes	Yes	Yes	Yes
12	Yes[1]	Yes[4]	No	Yes	Yes	No
13	Yes[1]	Yes[4]	No	Yes	Yes	No
14	No[5]	-	-	-	-	-
15	Yes	Yes[2]	Yes	Yes	Yes	No
16	Yes[4]	Yes[4]	No	Yes	Yes	Yes
Overall	14/16 (87.5%)	14/14 (100%)	8/14 (57.1%)	13/14 (92.9%)	14/14 (100%)	7/14 (50%)

[1] After (re)moving separated components in Google Blocks
[2] After using the Flatten Ground correction
[3] After using the Add Pedestal correction
[4] After adding struts in Google Blocks and using the Flatten Ground correction
[5] Not connected and too many edits required to continue

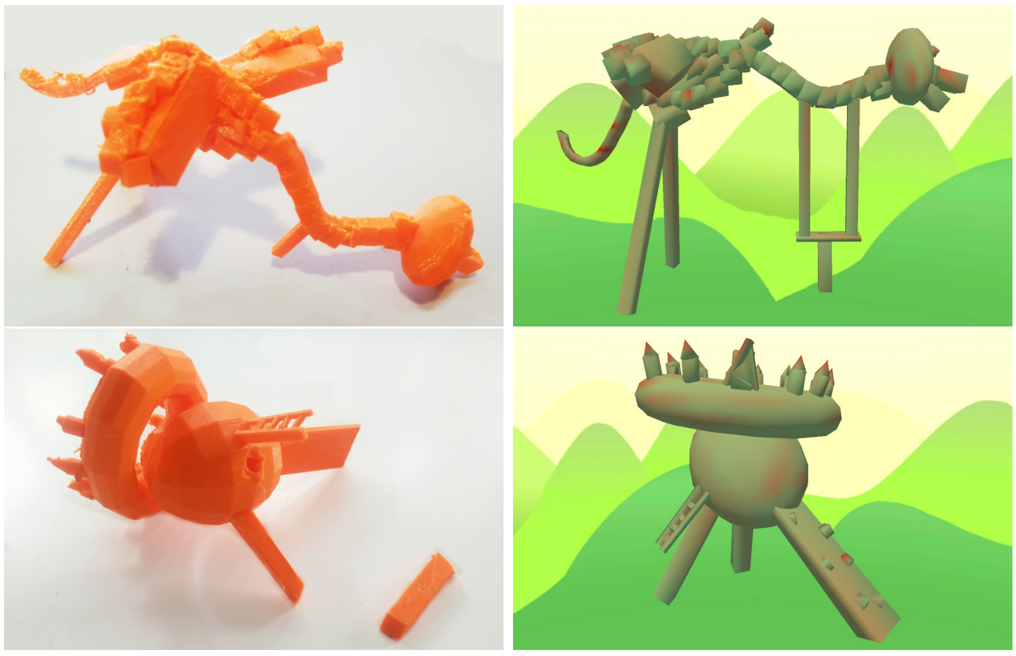


Figure 5.10: Printed models that did not survive the drop test. The weak area was not predicted correctly. Top: Model 12. Bottom: Model 13.

Table 5.3: Correctness of predictions by our program.

Connected	**Balanced**	**Strong**
92.9% (13/14)	100% (14/14)	50% (7/14)

could be edge cases where that functionality is not sufficient. The strength analysis shows some issues and only predicted the strength of the physical print correctly in 50% of cases. Furthermore in cases where it did predict issues with strength, it often did not predict what part of the model would be weak correctly, as can be seen in figure 5.10.

Chapter 6

Discussion & Conclusion

6.1 Discussion

Now that we have made the program (chapter 4) and put it to the test (chapter 5), it is time to answer the research question Q1b and hypothesis H1 as stated in chapter 3:

Question 1b (Q1b) What functionalities are necessary to convert a model created in Google Blocks into a model that makes it possible for the Original Prusa i3 MK3 printer to print the corresponding object with desired properties as specified in Q1a?

Hypothesis 1 (H1) The functionalities that are necessary include algorithms to compute vertex connectivity and component intersection to ensure connectedness, convex hull and center of mass computation to ensure balance and thin areas detection to ensure strength. These functionalities together will make it possible for the Original Prusa i3 MK3 printer to print the object with desired properties as specified in Q1a.

As you may recall from the answer in chapter 3, the desired properties are: (1) it should consist of one connected component that touches the printing platform, (2) it should be able to stand without falling over and (3) it should be structurally sound.

As discussed in the Results section of the previous chapter, in 92.9% of the tested models the algorithms to compute vertex connectivity and component intersection were sufficient to ensure property 1. In addition, convex hull and center of mass computation was sufficient for all tested models to ensure property 2. As for property 3, thin area detection was only sufficient in 50% of cases. With tweaks to our parameters and by merging the cross section parts of each layer with each other this could be improved but it would still miss cases where, for example, the majority of the weight ends up on one strut. What functionality is necessary instead to ensure property 3 would require more research.

These results show that algorithms to compute vertex connectivity and component intersection to ensure connectedness and convex hull and center of mass computation to ensure balance are sufficient for the vast majority of the tested models. Still, we can not conclude this is sufficient for every case. We only got to test our program on 16 models which means there could be edge cases where that functionality is not sufficient. This means that Q1b can only be partially answered according to the data we have gathered: the functionality of thin area detection is not sufficient to make it possible for the Original Prusa i3 MK3 printer to print the object with the third property as specified in Q1a.

Since the Original Prusa i3 MK3 is very similar to most FDM printers and the properties of nGen filament lies in between the commonly used PLA and ABS plastic types, the conclusion can be generalized to apply to most FDM printers and to the most commonly used plastic filaments. In addition, Google Blocks saves models in OBJ format, the conclusion can be further generalized to apply to any 3D model that can be saved in the commonly used OBJ format. This means that the functionality of thin area detection is not sufficient to make it possible for most FDM printers to print the object with the third property as specified in Q1a. And while we can not say it with certainty, algorithms to compute vertex connectivity, component intersection, convex hull and the center of mass are likely sufficient to ensure connectedness and balance for the majority of 3D models that can be saved in OBJ format.

There were a couple of things that went better than expected and there were a few things that could be improved in the future. We will now discuss these here.

The center of mass algorithm, considering it is an approximation that only takes the surface of the model into account, has served considerably well. This can be explained by the fact that all prints have had their infill set to the default of 20%, meaning that only the surface is printed in solid plastic and the insides are filled for only 20%.

Additionally, we initially aimed to print 10 of the 16 models in our limited timeframe but we managed to print 13 in the end. This exceeded our expectations also because we did not tell the children what it takes to 3D print a model and what to pay attention to. This was on purpose, to get models with a variety of issues that properly test our program.

Our program could be improved in a number of ways, mostly related to corrections. We had to scrap the 'delete component' correction, because we did not have enough time to fix the bugs it created. The program also sometimes hangs when adding in many supports with the 'add strut' correction, and when raising the pedestal too high with the 'add pedestal' correction. Furthermore all corrections could have more visual feedback built into them, like an indication where it works from in the 'delete component', 'add strut' and 'thicken' corrections and what it is going to affect in the 'delete component' and 'thicken' corrections. Corrections like 'add strut' could be more intuitive, for example by removing the laser, as for some of the children it took a long time to figure out the supports were created directly at the controller position, not where they are aiming.

Another improvement to our program is in the parts where it recalculates all analyses. This causes the program to hang for a few seconds because the analyses calculate a lot of stuff, but when closing the balance simulation or cross section display, this is not necessary. It could also recalculate partially when closing a correction. A more intuitive way of showing that the program is busy analyzing would go a long way in improving the experience and increasing the patience of the user.

When performing the experiment a couple of minor issues came to light, which could easily be improved, should a similar experiment be done in the future. The 15 minute sessions turned out too short, as it takes time to switch between groups, explain the purpose of the day's session and put on the headset. Having 3 children per group is not advised because in this limited time some children did not get to work in VR during some of the sessions. The painters could also have been introduced beforehand by the teacher in class. This was done in previous sessions outside of this project by KleurInCultuur, but did not happen this time because the sessions took place in the first week of the school year.

The drop test experiment could have been more thorough. We only dropped each model one time but survival depends, among other things, on the angle it falls on. Dropping it multiple times would improve the quality of the results, but the conclusion would still be the same. With a more thorough experiment more models would likely turn out not strong enough, which can only confirm that thin area detection is indeed not sufficient for strength analysis.

6.2 Conclusion and future work

Our literature study shows that desirable properties of 3D printed objects that commonly cause issues are: (1) it should consist of one connected component that touches the printing platform, (2) it should be able to stand without falling over and (3) it should be structurally sound. To find out what functionalities are needed to successfully analyze these properties we implemented a program with the functionalities that we considered the most likely to be successful. These functionalities are algorithms to compute vertex connectivity and component intersection to analyze the first property, convex hull and center of mass computation to analyze the second property and thin areas detection to analyze the third property. We implemented this program in the Unity engine with a step-by-step process and a number of basic corrections for the most common types of issues. It can be run on a normal screen and in VR and includes a balance simulator and a cross section visualizer. This makes it the first 3D print analysis program that works in virtual reality.

We performed an experiment with 35 elementary school children and asked them to create models in virtual reality without stating what the desired properties are. This way our program could be thoroughly tested in a real-world scenario. Then the models were printed and tested for desired properties. The results show that our program analyzed the connectedness 92.9% of the time, the balance 100% of the time and the strength of the print 50% of the time. This shows that computing vertex connectivity, component intersection, convex hull and center of mass computation is sufficient to make it possible for the Original Prusa i3 MK3 printer to print the object with the first and second property in most cases, but we can not conclude that it is sufficient in all cases. This is because we only got to test our program on 16 different models, and thus edge cases could exist where this functionality is not sufficient. However, we can conclude that thin area detection is not enough to successfully analyze strength in all models. While further research is needed to draw more conclusions, this is a good first step into gaining understanding of what it takes to convert models created in virtual reality to 3D prints.

This thesis showed that there are several areas in the fields of virtual reality and 3D printing that would benefit from getting a more extensive research project in the future. One of those areas is to research what is necessary to analyze the strength of a printed object based on its virtual model. We showed that thin area detection is not sufficient, but as of yet we do not know what is necessary.

A bigger and more conclusive experiment is also necessary to be able to conclude that computing vertex connectivity, component intersection, convex hull and the center of mass is either sufficient or insufficient for connectedness and balance analysis.

Another interesting area of research has to do with the supports necessary for 3D printing. Automatic printer supports is a difficult task and still requires manual checking if every part of the model is supported, which often is not the case. A balance needs to be struck between being able to print the model and being able to easily remove the printing supports afterwards. Too few supports might result in a failed print and too many supports can make it almost impossible to remove the supports without breaking the model.

A smaller scope of this as of yet unsolved area of research is to analyze the models where it would be difficult to add supports. Some models require more supports than others, but more importantly some models require support in difficult to reach places, such as high up away from the rest of the model or in the middle of the model where it would be difficult to remove the support without using soluble supports. It is also interesting to research for what type of models the Meshmixer type of supports are more beneficial and for what type of models the Slic3r type is better. Up until now this is done mostly on intuition and human experience alone.

Another potential project could be to create a metric that indicates how easy it is to knock an object over, and to create the ability to calculate this for a virtual model. Our program used a binary system to answer whether a model is balanced or not, and while this worked well enough for our project, the real world is more complicated than that. This could potentially be calculated using the distance from the center of mass to the convex hull and the height of the center of mass itself.

Additional research could also be done on the effect of virtual reality on the proficiency of inspecting models for print analysis. Humans might very well be able to inspect virtual models better and spot issues quicker when they can walk around the model and scale them up to be as large as they want, as opposed to being limited to screen size and the mouse and keyboard as input devices.

Bibliography

3DSystems (2005). Charles w. hull executive bio. https://www.3dsystems.com/sites/default/files/downloads/3D-Systems-Charles-W-Hull-Executive-Bio.pdf.

Alessandro Ranellucci, P. R. (2016). Slic3r prusa edition. https://www.prusa3d.com/slic3r-prusa-edition/.

Anthony (2014). High-performance physics in unity 5. https://blogs.unity3d.com/2014/07/08/high-performance-physics-in-unity-5/.

ASTM (2018). Additive manufactoring overview. https://www.astm.org/industry/additive-manufacturing-overview.html.

Autodesk, I. (2016). Autodesk meshmixer. http://www.meshmixer.com/.

Bolier, W., Hürst, W., and van Bommel, G. (2017). Drawing in a virtual 3d space. https://dspace.library.uu.nl/handle/1874/353003/.

Bykat, A. (1978). Convex hull of a finite set of points in two dimensions. *Information Processing Letters*, 7(6):296–298.

Chakravorty, D. (2018). The 4 most important 3d printer file formats – simply explained. https://all3dp.com/3d-printing-file-formats/.

Chan, T. M. (1996). Optimal output-sensitive convex hull algorithms in two and three dimensions. *Discrete & Computational Geometry*, 16(4):361–368.

Chand, D. R. and Kapur, S. S. (1970). An algorithm for convex polytopes. *J. ACM*, 17(1):78–86.

Christiansen, A. N., Schmidt, R., and Bærentzen, J. A. (2015). Automatic balancing of 3d models. *Computer-Aided Design*, 58:236 – 241. Solid and Physical Modeling 2014.

Clouds, W. (2018). File types used in 3d printing. http://ss.whiteclouds.com/3dpedia-index/file-types-used-3d-printing/.

Eddy, W. F. (1977). A new convex hull algorithm for planar sets. *ACM Transactions on Mathematical Software (TOMS)*, 3(4):398–403.

Fu, H., Cohen-Or, D., Dror, G., and Sheffer, A. (2008). Upright orientation of man-made objects. *ACM Trans. Graph.*, 27(3):42:1–42:7.

Galyean, T. and F. Hughes, J. (1991). Sculpting: An interactive volumetric modeling technique. 25:267–274.

Google (2016). Tilt brush official website. `https://www.tiltbrush.com/`.

Google (2017). Google blocks official website. `https://vr.google.com/blocks/`.

Graham, R. L. (1972). An efficient algorith for determining the convex hull of a finite planar set. *Information processing letters*, 1(4):132–133.

Hesse, B. (2015). Abs or pla: Which 3d printing filament should you use? `https://www.digitaltrends.com/cool-tech/abs-vs-pla-3d-printing-materials-comparison/`.

Hopcroft, J. and Tarjan, R. (1973). Algorithm 447: Efficient algorithms for graph manipulation. *Commun. ACM*, 16(6):372–378.

Hughes, T. J. (2012). *The finite element method: linear static and dynamic finite element analysis.* Courier Corporation.

Jarvis, R. (1973). On the identification of the convex hull of a finite set of points in the plane. *Information Processing Letters*, 2(1):18 – 21.

Kirkpatrick, D. G. and Seidel, R. (1986). The ultimate planar convex hull algorithm? *SIAM journal on computing*, 15(1):287–299.

Limited, G. S. (2017). Gravity sketch official website. `https://www.gravitysketch.com/`.

Lorensen, W. and E. Cline, H. (1987). Marching cubes: A high resolution 3d surface construction algorithm. 21:163–.

MasterpieceVR (2017). Masterpiecevr official website. `https://www.masterpiecevr.com/`.

McGraw, T., Garcia, E., and Sumner, D. (2017). Interactive swept surface modeling in virtual reality with motion-tracked controllers. In *Proceedings of the Symposium on Sketch-Based Interfaces and Modeling*, SBIM '17, pages 4:1–4:9, New York, NY, USA. ACM.

Mei, G. and Tipper, J. C. (2013). Simple and robust boolean operations for triangulated surfaces. *CoRR*, abs/1308.4434.

Möller, T. (1997). A fast triangle-triangle intersection test. *Journal of graphics tools*, 2(2):25–30.

NIH (2018). What file formats are used in 3d printing? `https://3dprint.nih.gov/faqs/1781/`.

Nooruddin, F. S. and Turk, G. (2003). Simplification and repair of polygonal models using volumetric techniques. *IEEE Transactions on Visualization and Computer Graphics*, 9(2):191–205.

Oropallo, W. and Piegl, L. A. (2016). Ten challenges in 3d printing. *Engineering with Computers*, 32(1):135–148.

Prévost, R., Whiting, E., Lefebvre, S., and Sorkine-Hornung, O. (2013). Make it stand: Balancing shapes for 3d fabrication. *ACM Trans. Graph.*, 32(4):81:1–81:10.

RadialGames (2016). Fantastic contraption official website. `http://fantasticcontraption.com/`.

Si, H. and TetGen, A. (2006). A quality tetrahedral mesh generator and three-dimensional delaunay triangulator. *Weierstrass Institute for Applied Analysis and Stochastic, Berlin, Germany*, page 81.

Stava, O., Vanek, J., Benes, B., Carr, N., and Měch, R. (2012). Stress relief: Improving structural strength of 3d printable objects. *ACM Trans. Graph.*, 31(4):48:1–48:11.

Telea, A. and Jalba, A. (2011). Voxel-based assessment of printability of 3d shapes. In Soille, P., Pesaresi, M., and Ouzounis, G. K., editors, *Mathematical Morphology and Its Applications to Image and Signal Processing*, pages 393–404, Berlin, Heidelberg. Springer Berlin Heidelberg.

Umetani, N. and Schmidt, R. (2013). Cross-sectional structural analysis for 3d printing optimization. In *SIGGRAPH Asia 2013 Technical Briefs*, SA '13, pages 5:1–5:4, New York, NY, USA. ACM.

Vanderkay, J. (2015). 3mf consortium launches to advance 3d printing technology. `https://3mf.io/3mf-consortium-launches-to-advance-3d-printing-technology/`.

Wang, W., Li, B., Qian, S., Liu, Y.-J., Wang, C. C. L., Liu, L., Yin, B., and Liu, X. (2017). Cross section-based hollowing and structural enhancement. *The Visual Computer*, 33(6):949–960.

Zhou, Q., Panetta, J., and Zorin, D. (2013). Worst-case structural analysis. *ACM Trans. Graph.*, 32(4):137–1.