ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Špilar** Jméno: **Vojtěch** Osobní číslo: **519775**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačové grafiky a interakce**

Studijní program: **Otevřená informatika**

Specializace: **Počítačové hry a grafika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Vizualizace 3D tisku pomocí vrhání paprsku

Název bakalářské práce anglicky:

Visualization of 3D Printing via Ray Tracing

Jméno a pracoviště vedoucí(ho) bakalářské práce:

prof. Ing. Vlastimil Havran, Ph.D. Katedra počítačové grafiky a interakce

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **12.02.2025** Termín odevzdání bakalářské práce: _____

Platnost zadání bakalářské práce: **20.09.2026**

_____
podpis vedoucí(ho) ústavu/katedry

_____
prof. Mgr. Petr Páta, Ph.D.
podpis proděkana(ky) z pověření děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

_____
Datum převzetí zadání

_____
Podpis studenta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Špilar** Jméno: **Vojtěch** Osobní číslo: **519775**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačové grafiky a interakce**

Studijní program: **Otevřená informatika**

Specializace: **Počítačové hry a grafika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Vizualizace 3D tisku pomocí vrhání paprsku

Název bakalářské práce anglicky:

Visualization of 3D Printing via Ray Tracing

Pokyny pro vypracování:

Analyzujte programy pro vizualizaci 3D modelu během 3D tisku a konverzi 3D modelu do GCODE pro tiskárny typu FDM (fused deposition modeling). Na základě této analýzy navrhněte a v jazyce ANSI C++ implementujte počítačový simulátor tisku na 3D tiskárně typu FDM pro vstupní program v jazyce GCODE (program pro 3D tisk v GCODE je výstupem softwarových nástrojů pro přípravu 3D tisku, např. Prusa Slicer, Cura) a jeho konverzi do objemového či hraničního modelu vhodného pro zobrazování pomocí vrhání paprsku. Pro tento průběžně v čase měnící se prostorový jednobarevný model tištěného 3D objektu analyzujte možné způsoby zobrazování se zaměřením na vrhání paprsku. Po konzultaci s vedoucím práce vyberte vhodný způsob zobrazování tištěného modelu a to buď s využitím vhodné knihovny pro vrhání paprsku na CPU jako je Intel Ospray [1] nebo navrhněte a implementujte vlastní datovou strukturu a odpovídající algoritmus na CPU. Zvolte vhodný osvětlovací model pro zobrazení, do výpočtu zobrazení využijte kromě barvy pokud možno i odrazivost materiálu tiskové struny nejlépe s využitím měření zkušebního výtisku konkrétních reálných materiálů pro isotropní BRDF [3]. Program otestujte alespoň na dvanácti různých modelech různého tvaru a velikosti bez opakování tvaru objektů (alespoň 4 objekty o objemu nejméně 10cm3, 4 objekty o objemu nejméně 100cm3 a 4 objekty o objemu nejméně 1000cm3). Naměřte čas běhu programu a spotřebu paměti pro převod GCODU do 3D modelu k zobrazení a výpočtu obrázku z něj. Na základě profilovací analýzy implementovaný algoritmus optimalizujte se zaměřením v první řadě na celkový čas výpočtu (konverze a zobrazování) a v druhé řadě co se týče kvality a fotorealističnosti spočtených obrázků ve srovnání s fotografiemi reálného 3D tisku.
Pro alespoň dva různé tvary 3D objektů (o objemu alespoň 100cm3) nasnímejte sekvenci fotografií případně video z reálného 3D tisku pro konkrétní 3D tiskárnu a vizuálně porovnejte pořízené snímky z reálného prostředí se snímky spočítanými programem pro pokud možno nejpodobnější nastavení polohy kamery jak v reálném prostředí tak v rámci vizualizace 3D tisku.

Seznam doporučené literatury:

[1] Intel OSPRAY, The Open, Scalable, and Portable Ray Tracing Engine, https://www.ospray.org/
[2] Cezner Lukáš, Měření povrchu BRDF s využitím LIGHTEC Minidiff V2 pro využití v programování her, bakalářská práce 2023, ČVUT FEL.
[3] Prusa: Basics of 3D printing with Josef Prusa, free E-book, URL: https://www.prusa3d.com/page/basics-of-3d-printing-with-josef-prusa_490/

FAKULTA ELEKTROTECHNICKÁ
FACULTY OF ELECTRICAL ENGINEERING
Technická 2
166 27 Praha 6

PROHLÁŠENÍ

Já, níže podepsaný

Příjmení, jméno studenta:	Špilar Vojtěch
Osobní číslo:	519775
Název programu:	Otevřená informatika

prohlašuji, že jsem bakalářskou práci s názvem

Vizualizace 3D tisku pomocí vrhání paprsku

vypracoval samostatně a uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací a Rámcovými pravidly používání umělé inteligence na ČVUT pro studijní a pedagogické účely v Bc a NM studiu.

Prohlašuji, že jsem v průběhu příprav a psaní závěrečné práce nepoužil žádný z nástrojů umělé inteligence. Jsem si vědom důsledků, kdy bude zjevné nepřiznané použití těchto nástrojů při tvorbě jakékoli části mé závěrečné práce.

V Praze dne 23.05.2025

Vojtěch Špilar
.....
podpis studenta

F3

Faculty of Electrical Engineering
Department of Computer Graphics and Interaction

Bachelor's Thesis

Visualization of 3D printing via Ray Tracing

Vojtěch Špilar

Květen 2025

Supervisor: prof. Ing. Vlastimil Havran, Ph.D.

Acknowledgement / Declaration

I would like to thank profesor Vlastimil Havran for supervising this thesis and for his very helpful constructive criticism and advises that greatly improved this work.

I declare that the present thesis was composed by myself and that the work contained here is my own. I also confirm that I have only used the specified resources

Abstrakt / Abstract

Cílem této práce je generování náhledů 3D tištěných objektů z instrukcí pro 3D tiskárnu zvaných G-code pomocí ray tracingu. Je zde popsán problém převodu instrukcí G-code na geometrickou reprezentaci tisknutého modelu. Dále jsou diskutovány problémy vykreslování geometrikcé remprezentace generované z G-code instrukcí použitím ray tracingu na CPU s naměřenými daty o odrazivosti materiálů používaných 3D tiskárnou pro dosažení photo-realismu

This thesis focuses on rendering images using ray tracing technique from instructions for a 3D printed called G-code. Problem of converting G-code instructions into geometrical representation of a 3D printed part.This thesis also discusses the problem of rendering the geometrical representation generated from G-code using CPU and utilizing measured reflectance data of materials used by a 3D printer in the rendering process to achieve photo-realistic results

Contents /

/ Figures

Chapter 1
Introduction

3D printing is a widely spread additive manufacturing process used in both fast prototyping and production. To create a 3D printed object, instruction for the 3D printer are needed. Those instructions are called G-code. Each line of G-code describes single elementary action that the 3D printer in succession reads and performs. This results in long and not very readable files. Finding errors in these files directly is very difficult. To solve this problem, software generating previews of the 3D printed part exists. The generated previews offer higher readability thanks to color coding and the possibility to see how the creation of 3D printed part progresses through the process. This previews are rendered using simplified graphics, rendering images in real time.

However, some applications may require photo-realistic previews of the 3D printed part. Previewing the final look of a 3D printed object in photo-realistic detail can be achieved by exporting the geometry and using different physically based rendering software. This software commonly requires graphical accelerator which are not available on every computational device.

1.1 goals

First goal of this thesis is to parse G-code program which was generated for a 3D printer by a slicer software. Evaluate the parsed G-code using a simulation. The simulation should examine relevant G-code commands and predict behaviour of the 3D printer. Based on the prediction of behaviour the simulation should approximate the result of the 3D printing process with a geometrical representation. This geometrical representation of the 3D printed object can be used to render a preview.

Second goal of this thesis to create renderer that uses ray tracing technique to generate the previews. Ray tracing generates images by casting rays into a scene and collecting color information based on what objects they hit. During traversal of the scene rays are accumulating color information resulting in rendered image. The renderer should be run on CPU and the renderer should use the geometric representation generated by our simulation of the 3D printing process.

Third goal is to measure reflectance of some materials used by a 3D printer using a scatterometer and to use this data to increase photo-realism of resulting images. Reflectance describes how light interacts with a material based in angle of incidence at which the light hits the object and at what viewing angle are we observing.

Chapter 2
3D Printing

3D printing is an additive manufacturing process, in this process material is deposited in layers in order to create the final object. To use 3D printers, a virtual model is required which is then converted to information about layers which the 3D printer reads and according to them deposits material. This manufacturing method has low startup time and costs compared to subtractive or formative manufacturing. In subtractive manufacturing a chunk of material is cut until desired geometry is reached. This wastes material and the cutting tool must reach all positions where cutting is needed, which may result in the need of a machine capable of movement along more axes, adding complexity, time and cost. Formative manufacturing such as injection molding or stamping creates objects by forming or molding material into the final geometry. It requires creation of molds or dies in order to produce parts, the production can be faster and cheaper, but creating molds or dies increases startup costs and time, as can be seen in figure 2.1. The ability of 3D printers to create complex geometries with low startup costs, quickly producing little waste is ideal for low volume production or prototyping [1].

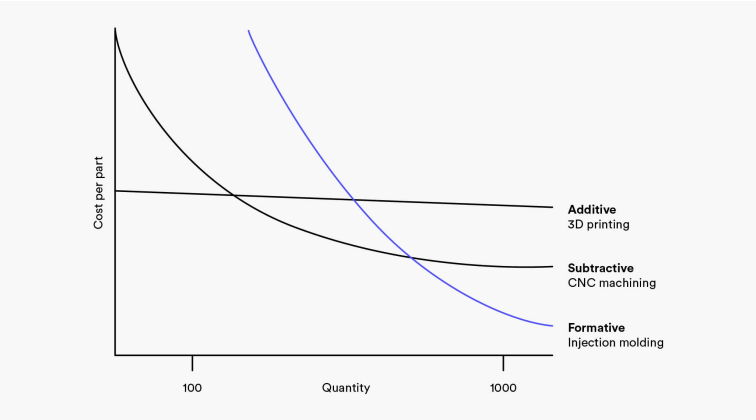

Figure 2.1. Rough approximation of cost per part. Image taken from [1]

2.1 Fused Deposition Modeling 3D Printing

Fused Deposition Modeling 3D printing (FDM 3D printing), also known as fused filament fabrication 3D printing (FFF 3D printing), is a process where material is deposited through a nozzle onto a printing platform. The material is deposited layer by layer creating the final 3D object. The FDM principle is depicted in figure 2.2. The material used is mainly different types of thermoplastics in the form of thin spooled cord called filament. The filament is pushed into part of the printer called extruder by an external motor or pulled in directly by the extruder. In the extruder, the material is heated up and deposited through a thin nozzle onto the printing platform (also referred to as the bed) of the printer or onto previously deposited layers where it is cooled and

solidifies. The extruder is moved using the 3D printer's motion system which can come in different types [2].

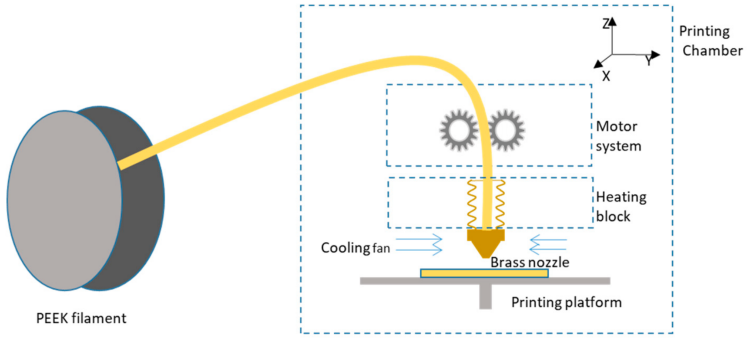

Figure 2.2. FDM 3D printer diagram. [3]

2.2 Types of 3D printer movement systems

There are 3 main types of 3D printer movement systems, as depicted in figure 2.3. Cartesian is the most common system where the extruder moves along three linear mutually perpendicular axes. There are also systems where the printer platform moves along the z axis and the extruder only moves in the x/y plane. The Delta movement system uses three independently controlled vertical rails with rigid rods linked to the extruder to move it. Lastly the least common movement system is the Polar system, where the extruder moves along two linear perpendicular axes and deposits material onto a printer platform that spins around its center [2].

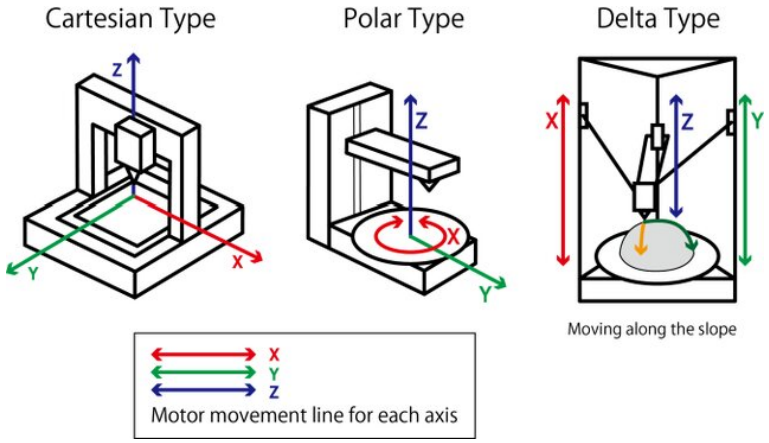

Figure 2.3. Types of FDM 3D printer. [4]

2.3 Industrial and desktop FDM 3D printers

FDM 3D printers can be assigned to two categories based on the scale of production, prototyping (desktop) and industrial (professional) respectively shown in figures 2.4 and 2.5. Industrial FDM 3D printers are more expensive but offer higher efficiency, speed, the ability to print materials requiring higher temperatures and the ability to print using multiple different materials at once to create an 3D object, they are designed for repeatability and reliability. Desktop FDM 3D printers require more user maintenance and calibration and may not be able to print some materials [5].

Figure 2.4. Desktop 3D printer Original Prusa MK4S. Image taken from [6].

Figure 2.5. Industial 3D printer Fortus 450mc. Image taken from [7].

2.4 Infill

Thanks to incremental layering of material, it is possible to create internal structures inside the printed object. If there is no need for the inside of an object to be solid, there is a possibility to fill the inside with hollow structures called infill. This makes the object lighter and saves material, however it can also decrease its rigidity compared to a solid object. Density and pattern of infill can be adjusted based on the requirements [8]. Examples of different infill patterns with different infill ratio percentages are shown in fırures 2.6 to 2.9. The ratio states how much of the internal volume is occupied by the generated infill structure.

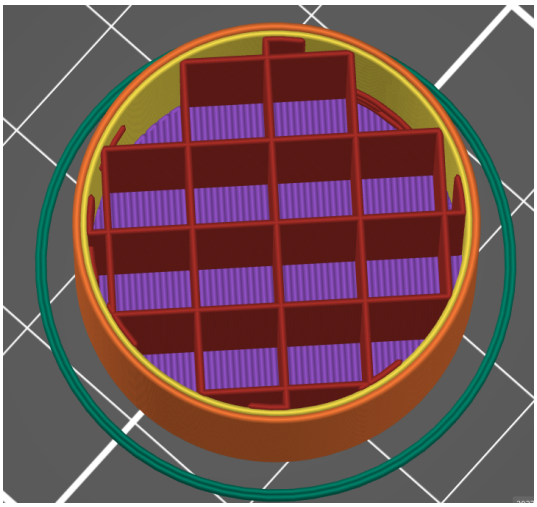

Figure 2.6. Grid pattern infill with 15% ratio.

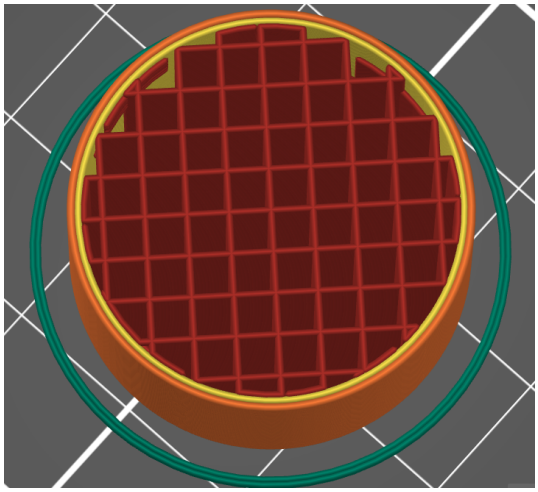

Figure 2.7. Grid pattern infill with 30% ratio.

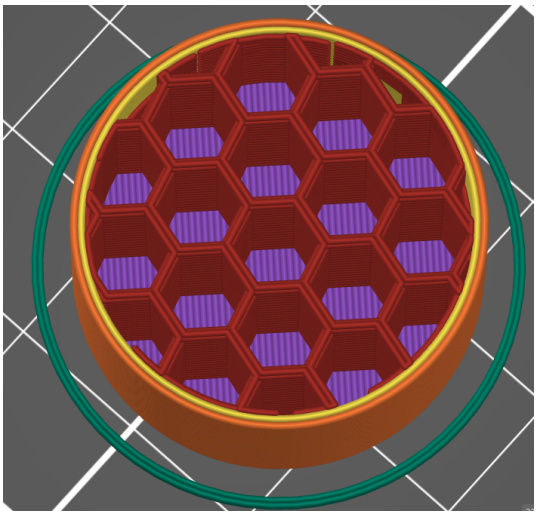

Figure 2.8. Honeycomb pattern infill with 15% ratio.

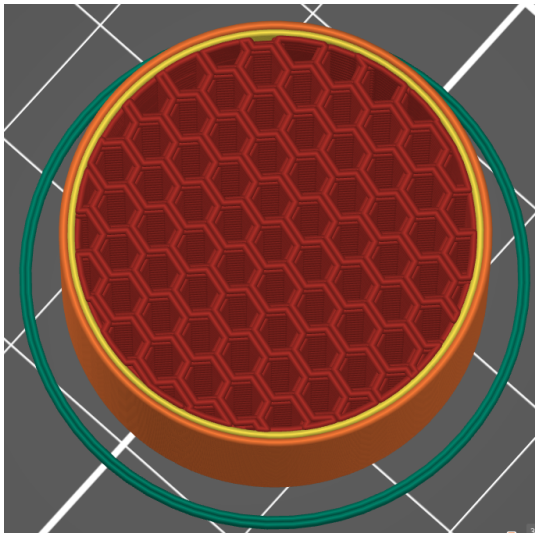

Figure 2.9. Honeycomb pattern infill with 30% ratio.

2.5 Overhangs and supports

In the 3D printing process layers are gradually stacked on top of each other and when the material is extruded without a solid base underneath it can drop down due to gravity before completely cooling and solidifying. Next layers will also be impacted due to the lowered position of the previous layer. This occurs when the material is not extruded directly above the previous layer, it can be seen in figure 2.10. Part of the layer that is not above the previous layer is called overhang.

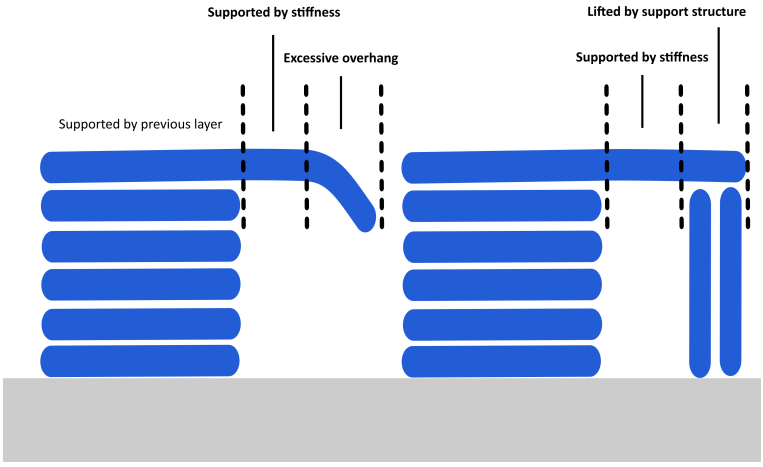

Figure 2.10. Overhangs in 3D printing process.

If overhangs are too large, support structure to provide a solid base for the viscous material to deposit onto is needed. This structure is printed along with the printed object and needs to be removed afterwards [9].

2.6 Multi-Material FDM 3D printing

Some FDM 3D printers support printing using multiple materials. That allows combining properties of different materials for mechanical parts, printing of multicolored objects or for easier support structure removal with soluble materials. Multi-material printing increases complexity for preparation of the 3D print and for the printing process itself where change of material may be performed multiple times each layer, where each material may require different settings of the printer [10]. We will describe two ways of printing with multiple materials, single exturder changing material and multiple extruders each with its assigned material.

2.6.1 Single extruder

If a 3D printer supports multi-material printing and has only one extruder, the filament is pulled from the extruder and swapped for a different one which is then inserted into the extruder. After new material is inserted it is mixed with some melted residue from the previous material inside the extruder, because of that material needs to be purged before continuing printing or some part of the model will be made from a mix of two materials [10].

2.6.2 Multi extruder

Problem of a single extruder needing to purge material can be solved by using a 3D printer with multiple extruders, where each material is assigned to a single nozzle. This generally increases the cost and complexity of the machine. Some 3D printers have multiple extruders on a single axis. Another possibility is that the moving mechanism of a 3D printer changes which extruder is attached to it during the printing process [10].

2.7 Filament Materials

FDM 3D printers can use multiple materials of filament, but not every 3D printer can use every material. Some materials require higher temperature inside the extruder, usually between 180-250°C. Some materials need slower printing, different nozzles or to be printed in heated space to prevent warping or cracking that can occure due to stresses induced by faster cooling and shrinking of higher layers that are more distant from heated 3D printer platform. Materials have different mechanical properties, hardness, stiffness, adhesion between layers and more.

2.7.1 Commonly used

Most materials for FDM 3D printing result in rigid objects. Most common are PLA, PETG and ABS. PLA is easy to use material compatible with most 3D printers, it has low printing temperature, usually does not warp or crack without heated chamber but it is less durable than ABS and PETG. PETG is sturdy with a smooth surface, but more hygroscopic. That means it absorbs more moisture from air and the result of wet materials can be worse surface finish of printed objects and worse adhesion between layers [11]. ABS is hard material but it is more prone to warping if it is not printed in a heated chamber [12].

2.7.2 Flexible

Flexible materials are made from TPEs (thermoplastic elastomers) which is a blend of plastic and rubber. These materials are difficult to print and require good control of the printing process. TPU (thermoplastic polyurethane) and TPC (thermoplastic co-polyester) are common materials that fall under TPEs. TPUs materials are generally more durable and can offer high resistance to abrasion, oils, chemicals and high and low temperatures. TPC materials offer high-temperature and UV resistance [12].

2.7.3 Soluble

Some materials can be dissolved using some solution, this is useful for support structures that can be dissolved instead of removal using abrasives and cutting tools. HIPS (High impact polystyrene) is tough and flexible material, it is easily machined and painted and can be dissolved using limonene solution. Other soluble material is PVA (Polyvinyl alcohol), it is soluble in water which allows use of soluble support structures, but it needs to be stored in a dry environment or the moisture in the air can damage it. PVB (polyvinyl butyral) has similar properties to PLA and can be smoothed using Isopropyl Alcohol vapors [12].

2.7.4 High performance

High performance materials such as PEEK (Polyaryletherketone) or PEI (Polyetherimide) offer higher strength to weight ratio, heat and chemical resistance, lower flammability compared to other cheaper materials. Disadvantages are higher cost and more difficult printing process that may need 3D printer with more capabilities (higher printing temperature, heated chamber) [12]

Chapter 3
G-code

Geometric code (G-code) [13]) is a simple programming language that does not require complex logic used for computer numerical control machining (CNC) which refers to computerized operation of machining tools used in manufacturing. G-code consists of instructions that the microcontroller in the CNC machine can read, interpret and then pass it to relevant machine part. G-code works in combination with machine code (M-code). G-code controls functions related to movement of machine parts and M-code controls functions related to other functions of the machine, for example instructions to pause the program, coolant flow or controlling heating elements. Different manufacturers may customize the format of the G-code or implement only some instructions [14].

3.1 G-code for FDM 3D printing

FDM 3D printers are also CNC machines and use G-code to control the machining process. A G-code program for a 3D printer can be generated using a lower level library like Mecode [15] where every action is coded in a different programming language (Python) and then translated into G-code and exported. More common practice is generating G-code using specialized software dedicated to generating instructions for 3D printer from virtual 3D model [16–17]. Example of Mecode:

```
from mecode import G
g = G()
g.move(10, 10)  # move 10mm in x and 10mm in y
g.arc(x=10, y=5, radius=20, direction='CCW')  # counterclockwise arc with
a radius of 5
g.meander(5, 10, spacing=1)  # trace a rectangle meander with 1mm spacing
between passes
g.abs_move(x=1, y=1)  # move the tool head to position (1, 1)
g.home()  # move the tool head to the origin (0, 0)
```

Different 3D printers may work with different implementations of G-code so the correct implementation must be selected before generating the G-code.

3.2 Syntax

Each line of G-code contains one instruction starting with its identifier, after that come arguments starting with argument identifier then value of the argument. For example

```
G1 X20.3 Y19.2 E0.8
```

is instruction for linear movement G1 with arguments X = 20.3, Y = 19.2 and E=0.8.

3.3 Most used instructions in FDM 3D printing

3.3.1 G28 - perform homing routine

This instruction tells the 3D printer to go to the edges of its axes until it hits the end. Homing routine is used so that the printer knows its position after starting a program.

3.3.2 G1 - linear movement

This instruction tells the 3D printer to move the extruder to position X, Y, Z at given feedrate F (speed of the movement) and move material in the extruder by the amount E. This instruction makes up the majority of the G-code program for 3D printers. Most printers support “sticky coordinates”, which means that if the argument is missing it is taken from the last G1 instruction containing the missing argument. . The coordinates in G1 command can be read by the machine either as absolute or relative, where absolute is a coordinate system in relation to the machine and relative coordinates state distance to move from the current position of the tool. The coordinate system used by machines G-code interpreter is chosen by other g-code commands (G91 - all relative, G90 - all absolute , M83 - E relative, M82 - E absolute) [18].

3.3.3 G28 - Move to Origin (Home)

This instruction tells the 3D printer to move the extruder to predefined origin (Home), flags X, Y and Z specify along which axes will be the extruder set to its origin. Moving extruder to origin is used to determine starting position of the extruder [18].

3.3.4 M104 and M109 - extruder heating commands

M104 instruction starts heating the extruder while allowing other commands to be performed immediately after. M109 instruction waits until the extruder reaches a given temperature. Printers using Makerbot firmware use command M133 for heating the extruder while this command in other firmwares sets PID I limit value [18].

3.3.5 M140 and M190 - bed heating commands

Analogically to extruder heating commands M140 starts heating the printing platform and does not wait and M190 waits until the printing platform reaches a given temperature [18].

3.3.6 M106 - set fan speed

M106 instruction sets the speed of the cooling fan that is cooling the extruded material [18].

Chapter 4
Slicer

Slicer is a software that converts digital 3D models into printing instructions for a given 3D printer represented by G-code according to settings specified by the user, such as layer height, speed, extruder temperature, support structure settings, infill settings, wall thickness etc.. Most commonly used slicers are Cura and PrusaSlicer, shown respectively in figures 4.2 and 4.1 with preview of sliced cylinder that is 25 mm tall and has radius of 14 mm. The two slicers are free and open-source [19].

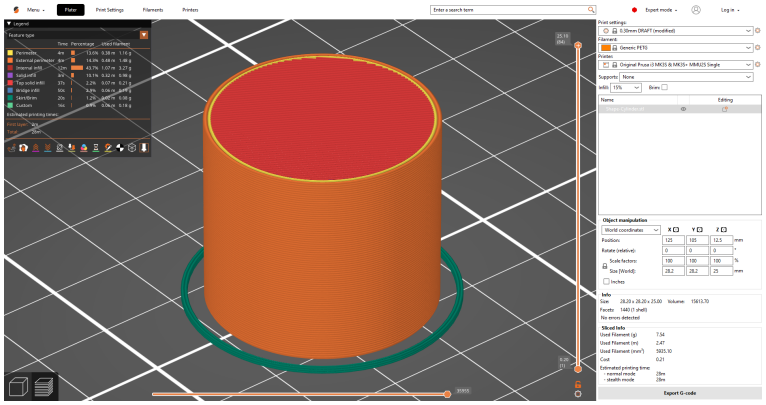

Figure 4.1. PrusaSlicer

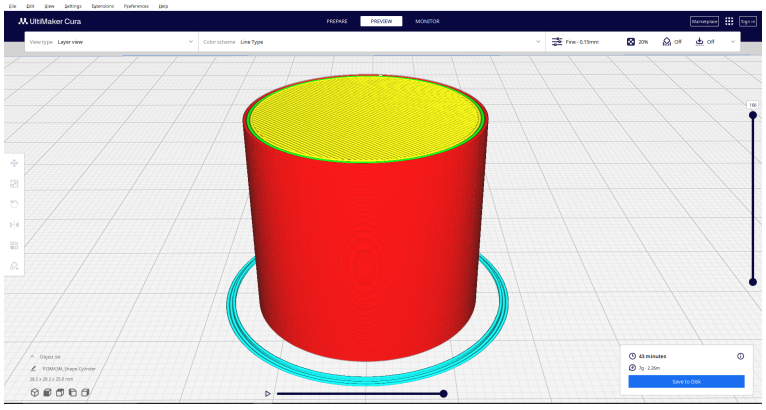

Figure 4.2. CuraSlicer

4.1 input files

Different CAD softwares used for creation of virtual 3D models use their own formats to store information about a given model, ".blend" for Blender, ".sldprt" and ".sldasm" for SolidWorks. To avoid implementing many different formats, slicers work with multiple standardized 3D model formats as which CAD softwares can export the model.

4.2 Commonly used standardised 3D model formats

4.2.1 .stl format

Stereolithography format [20] know as STL represents a 3D object as a triangular mesh and stores information for each triangle facet of the 3D model, each triangle is described by 3 positions of its vertices and vector specifying the normal of the face represented by the triangle. There is no standard support for colors or textures of the 3D object.

4.2.2 .obj format

Wavefront OBJ [21] represents a 3D object as a mesh composed of polygonal faces. Each vertex must be defined by 3 floating point values representing its position, but more information can be added, normal and texture coordinates of the vertex. N-Polygonal faces are then defined by n integer values pointing to what vertices the face contains. Geometry can then be split into multiple groups each defining its vertices and faces.

4.2.3 .3mf format

3D manufacturing format [22] is a 3D printing format that can hold more information than just the geometry of a 3D model. 3D printer profile, generated support structures, units in which the model was created, color and texture information and more in XML format. It is widely used across CAD and slicer softwares.

4.3 3D Print Preview

Preview of 3D printing is important for increasing effectiveness and error prevention. Tools used for preview are usually implemented in slicer software, others are accessible as online service. Previews generated by these tools are not photorealistic but generated using raster graphics or by rendering lines along the path of the extruder during the 3D printing process. These tools allow for quick check of printer settings assigned to different parts of the 3D model thanks to colored parts and legend describing what each color means.

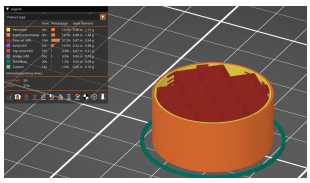

Figure 4.3. Preview in middle of printing process in PrusaSlicer.

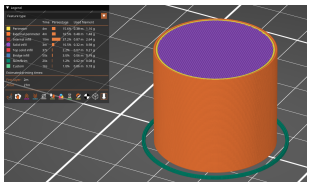

Figure 4.4. Preview of completed part in PrusaSlicer.

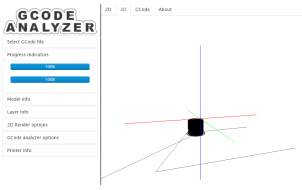

Figure 4.5. Preview in GcodeAnalyzer `https://gcode.ws`

Photorealistic preview can be generated by exporting 3D triangle mesh used by slicer for rasterized preview as .obj file and by using this file in some rendering engine. From the commonly used slicers mesh can be exported from PrusaSlicer and Superslicer. Prusaslicer is open-source and the algorithm for generating the mesh can be found on github [23] in file src/slic3r/GUI/3DScene.cpp. Extruder path along which material is being deposited is represented by line segments from which positions and normals of vertices are calculated. These vertices create a rhombus which approximates the profile of the layer in a given position. These profiles are then connected by triangles creating 3D triangular mesh.

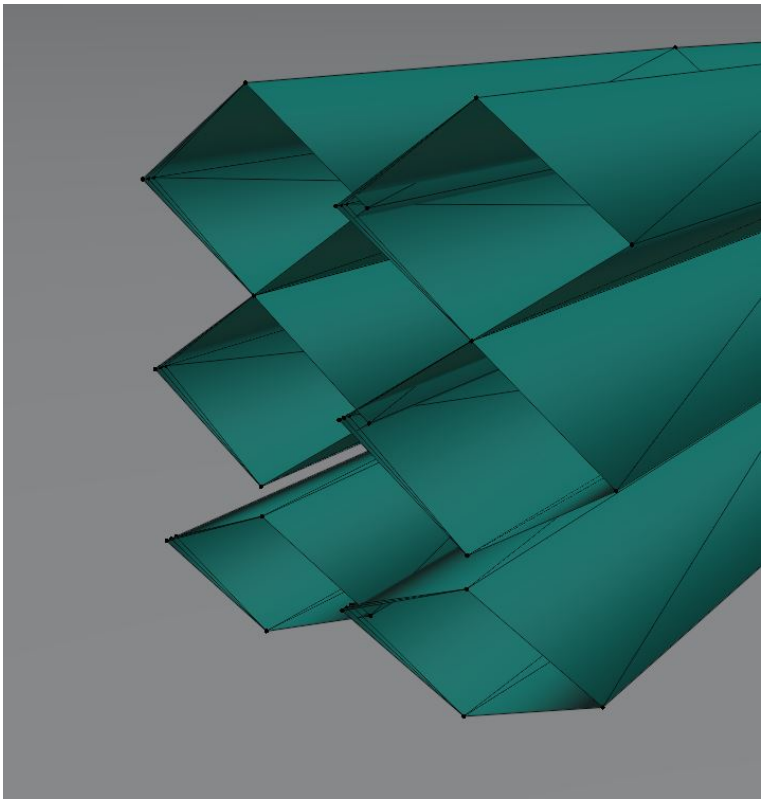

Figure 4.6. PrusaSlicer exportovaná mesh v Blenderu.

Disadvantage of this method is increasing complexity of the 3D mesh, every movement of the extruder during which material is being deposited adds additional geometry including infill structures and support structures. If the same 3D model is scaled up, the number of layers during the 3D printing process increases and so does the time needed to render a photorealistic image. Following images were generated in Blender with materials provided by PrusaSlicer during the export using renderers EEVVEE and cycles and in Fusion360 with material Plastic-Glossy (Yellow). The small model is a cylinder of height and diameter of 20mm the medium model has height and diameter of 75mm and the large model has height and diameter of 150mm. EEVVEE is rendering engine designed to be used for previews and for close to real time performance while sacrificing some quality, cycles is designed to be precise and physically based while taking more time to render images. Mesh was generated from differently scaled cylinders with the same 3D printer settings for slicing. Figures 4.7 to 4.15 contain caption with information about what software and scaled model was used to render the image and how long the rendering took in seconds.

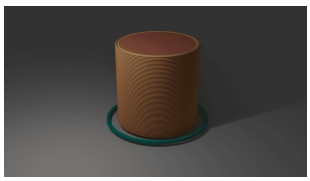

Figure 4.7. Blender-EEVVEE small model 11s.

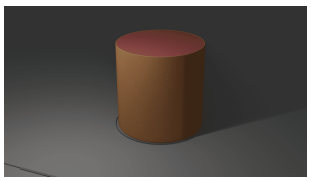

Figure 4.8. Blender-EEVVEE medium model 12s.

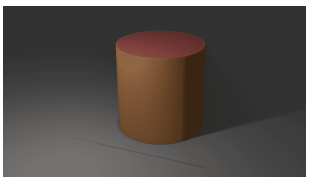

Figure 4.9. Blender-EEVVEE large model 15s.

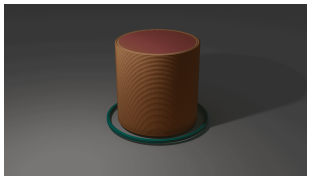

Figure 4.10. Blender-cycles small model 76s.

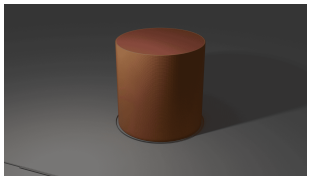

Figure 4.11. Blender-cycles medium model 124s.

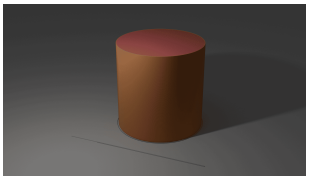

Figure 4.12. Blender-cycles large model 276s.

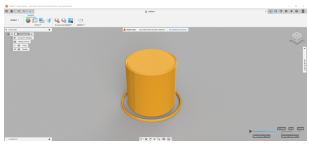

Figure 4.13. Autodesk Fusion360 small model 30s.

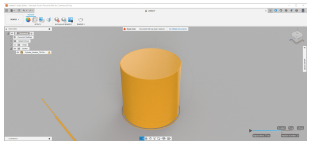

Figure 4.14. Autodesk Fusion360 medium model 37s.

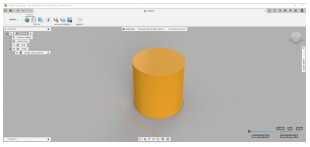

Figure 4.15. Autodesk Fusion360 large model 38s.

4.4 3D Geometry from G-code

If there is no option to export a triangle mesh from slicer software or the mesh does not have desired properties it is possible to simulate the 3D printing process based on G-code program and create a new virtual 3D model of the printed part.

4.4.1 Geometry from point cloud

From G1 and G0 commands extrusion arguments are read, based on their value simulation can differentiate whether during the movement during which material is being deposited or the movement is only travel between different parts of 3D printed model. It is possible to get point cloud of lines from the data of extruder travel, that is then possible to convert to desired file format using other software like MeshLab [24–26].

4.4.2 Geometry from extruder trajectory

Similarly as in previous section G1 and G0 commands are read to find trajectory composed of straight line segment along which is the extruder depositing material. G1 and G0 commands are interchangeable, usually the G-code only contains the G1 command and if it contains both, G0 is usually reserved for travelling moves along which there is no material deposited [27]. To find these trajectories algorithm that is described by figure 4.16 is used.

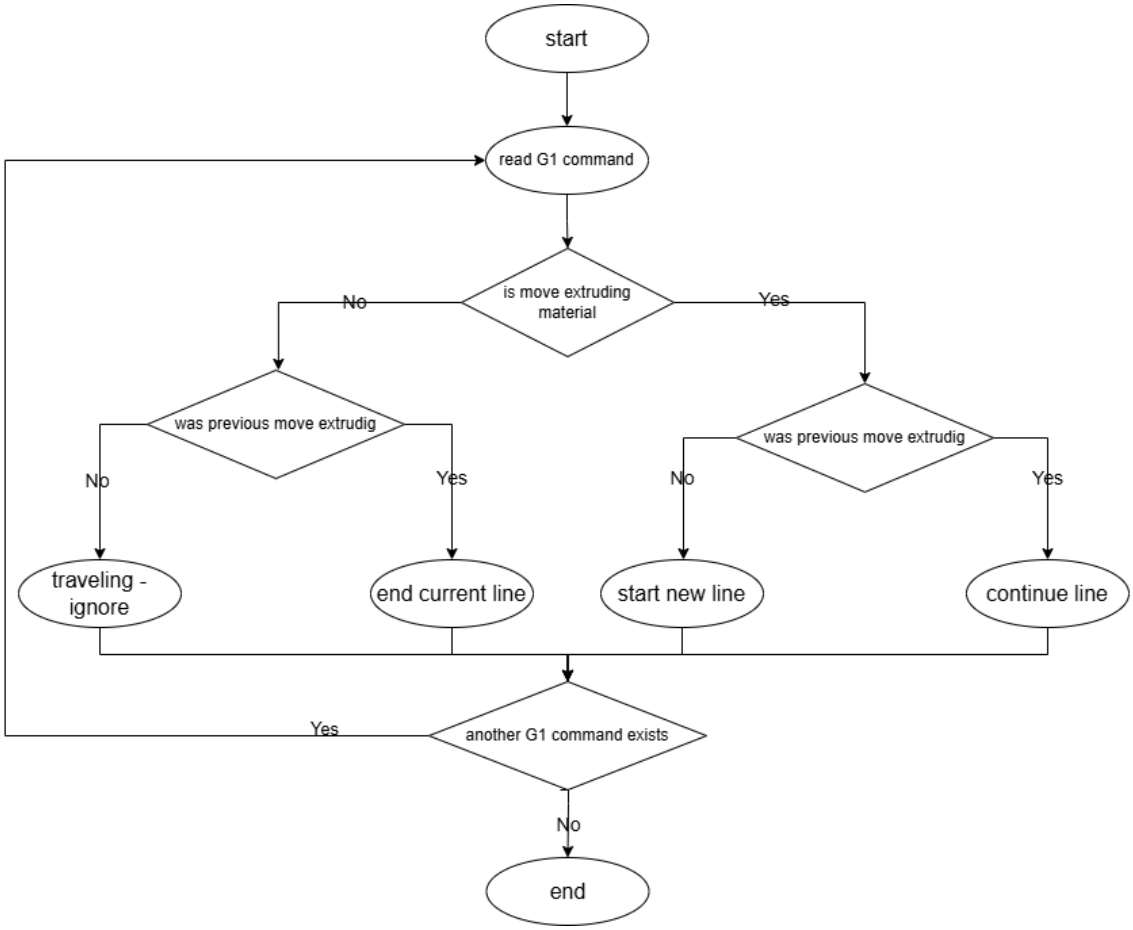

Figure 4.16. Line generating algorithm.

Series of points between which the extruder continuously deposits material is referred to as a line. Based on the extrusion value of the previous command is determined if the current line should be continued, ended, or if there is no current line new if new one should be started. This algorithm is repeated until all G1 (and G0) commands in G-code file are read. Along with the information about position of the extruder it is also useful to save extrusion value corresponding to each point for later calculations.

Chapter 5
Geometry from lines

To approximate shape of extruded material shape of the cross section of the extruded material is extruded along the lines generated by the algorithm 4.16. The shape is created from 3 parts, two circles approximating squished out material and a rectangle that represents material flattened by flat bottom of the extruder nozzle. It is shown in figure 5.1.

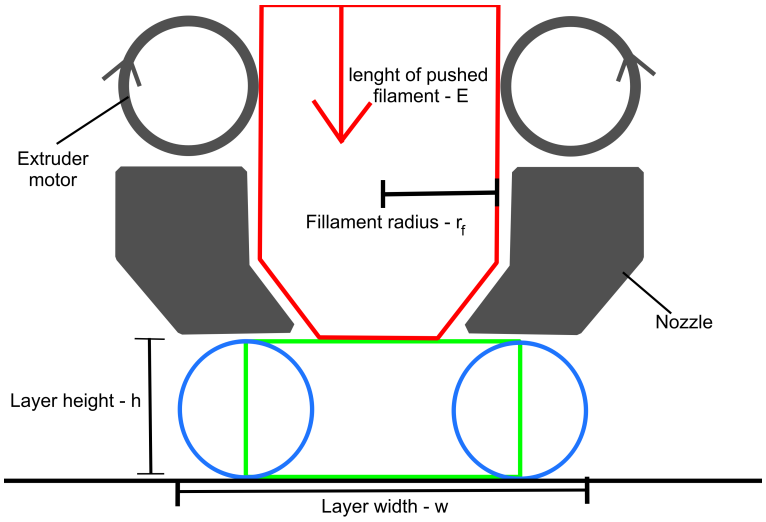

Figure 5.1. Cross section of extruded material.

Layer height can be measured from the difference between previous layer and current height of the extruder. Equation to calculate layer width is possible to calculate using the equation (1)

$$V_{in} = V_{out} \tag{1}$$

For each line segment V_{in} is defined by the length of filament pushed into the extruder (derived from extrusion values of the G1 command) and the diameter of the filament (usually 1.75 mm). V_{out} can be then split into the volume of rectangular box created by extruding the rectangular part of the cross section geometry 5.1 and two half cylinders created similarly by extrusion of profile along the line segment. The layer width w is calculated as:

$$w = \frac{\pi \cdot r_f^2 \cdot E}{l \cdot h} - h \cdot (\frac{\pi}{4} - 1) \tag{2}$$

Where r_f is radius of the filament , E is length of filament pushed into the extruder, l is length of the line segment between two points defined by two consecutive G1 instructions, it can be seen in figure 5.2 and h is height of the extruded layer.

By extruding the cross section from one point of the lie segment to the other we get rectangular geometric primitives representing the extruded material shown in figure 5.2

that are not connected by their front and back face as shown in figure 5.2 in the top view.

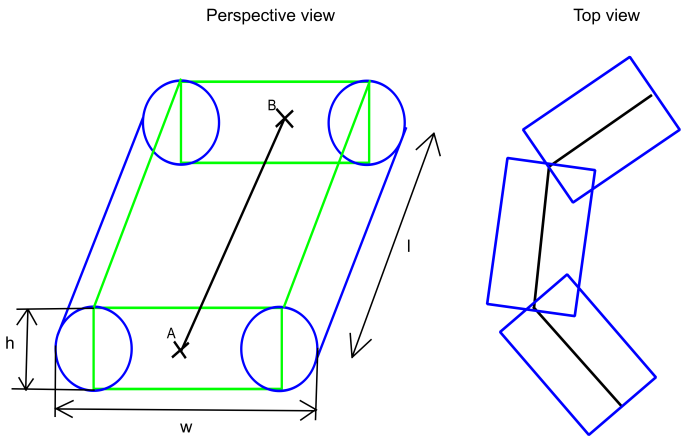

Figure 5.2. Geometric shape generated by extruding the profile along a line segment.

To create smooth transitions between neighbouring primitives the geometry is cut/extended based on two cutting planes each at one of the points defining the line segment, A and B. Each plane is defined by according normals n_1 and n_2 as shown in figure 5.3. The normals of the planes are parallel with x/y plane and are in such direction that the planes intersect with points where would the most outside parts of the two neighbouring primitives intersect if they were infinite (A_i, B_i).

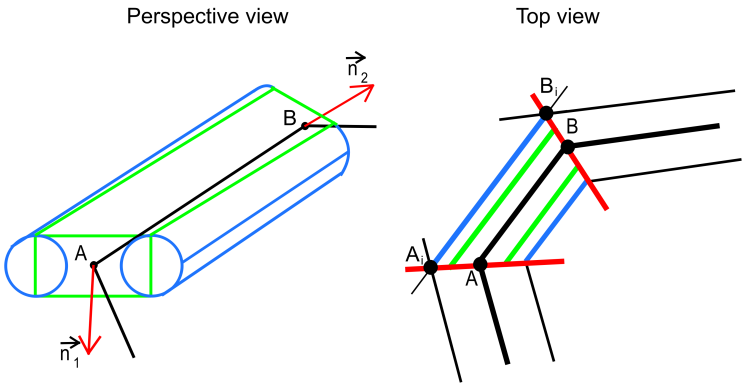

Figure 5.3. Geometry created by extruding cross section along a line segment and cut by two planes.

Unless the cutting planes intersect inside the geometry volume is preserved so the equation (2) holds true. Generated primitives can be described by the two points of the line segment, normals of the cutting planes, layer width and layer height.

5.1 Tessellation

Rendering software and libraries mainly work with triangular meshes composed of vertices connected by triangles, to convert generated primitives to triangular mesh process called tessellation is used. Points are placed around the outer surface of each face of the primitive. Each two neighbouring points create a trapezoid with two corresponding vertices on the other side of the primitive. This trapezoid can be then split into two triangles as seen in 5.4

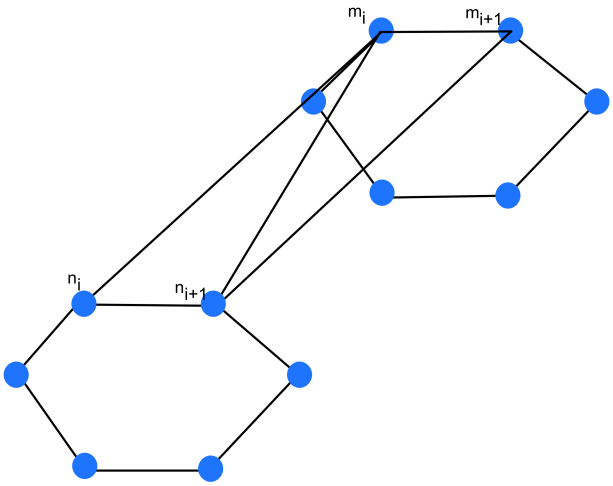

Figure 5.4. Creation of triangles from point along faces of the primitive.

Number of points on each face of the primitive determines how closely the generated triangle mesh will resemble the original geometric primitive as shown in 5.5.

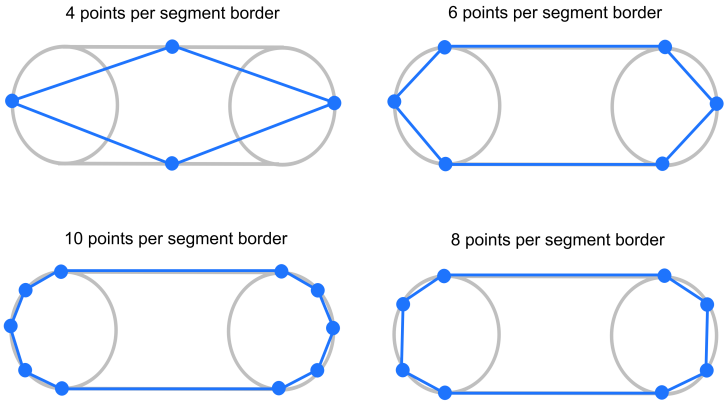

Figure 5.5. Approximation of primitive by generated triangular mesh.

Flat faces on top and bottom of the primitive do not require more than two triangles to be precisely represented, so for further tessellation only vertices touching the cylindrical parts of the primitive are required. Increasing number triangles generated by tessellation increases required memory to store the newly generated mesh which can increase time needed to render images using this geometry.

Chapter 6
Raytracing

Ray tracing [28] is an image rendering technique that simulates light traveling as rays from light sources in a scene. Rays are reflecting and refracting along their path through the scene and some of them will enter the eye/camera contributing to the resulting image. Most of the rays casted by light sources never hit the camera or reflect so many times their light intensity is so low the change in resulting image is negligible. To reduce the number of calculations needed rays are cast in reverse direction, starting in the camera and traveling through the scene collecting light information from each interaction with objects in the scene. For each pixel in the rendered image one or more rays are cast in a direction going through a pixel of an image projected on the near plane of the camera (near plane is a minimally sized rectangle at a selected distance that completely covers the view of an eye/camera). This is shown in figure 6.1. When a light ray collides with an object in the, multiple new rays can be cast from the point of contact to collect additional light information. Light information from these newly created rays is then accumulated and passed to a previous ray.

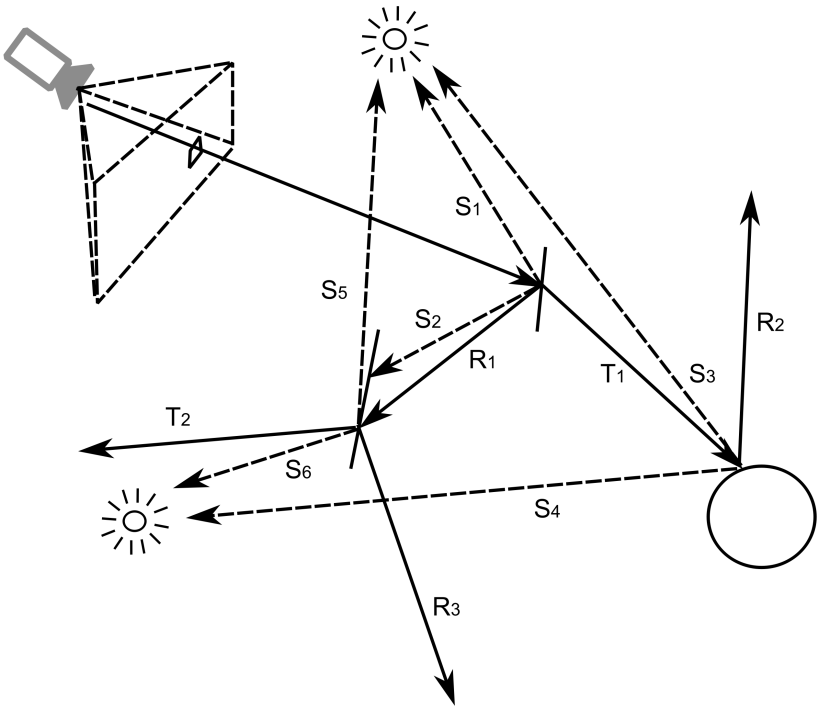

Figure 6.1. Ray tracing diagram.

6.1 Simple ray tracing

One ray is spawned for each pixel, if the ray hits an object two or three rays are spawned returning light values that are then added together. Shadow ray is cast in the direction of each light source to determine which light sources contribute to the light value of the previous ray by checking if there is a direct clear way between the point of contact and the light source. Reflection ray is cast in the direction incoming ray would reflect. The reflection ray behaves the same as the ray cast from the camera, it collects light information if it hits an object and adds it to the incoming ray from which the reflection ray was generated. Refraction ray behaves similarly to reflection ray but is only cast if a hit object is not opaque. Refraction ray is cast in the direction the incoming ray would take if it was refracted through the object it collided with. After each reflection or refraction ray hits an object this process can be repeated and cast more rays into the scene [28]. In the figure 6.1 ray E is cast from camera and propagates through a scene. Rays marked S are shadow rays, rays marked R are reflected rays and rays marked T are refracted rays passing through transparent objects.

6.2 Improving image quality

For higher quality of rendered images, more rays can be cast for each pixel with a small offsets in the direction defined by pixel position on the near plane of the camera, multiple reflection/refraction rays can be cast after collision in pseudo random directions based on the material of the hit object. More complex methods can be used to evaluate incoming light from multiple rays. Rendered images without enough rays cast for each pixel can appear noisy, this effect can be reduced using de-noising algorithms.

6.3 Bounding Volume Hierarchy tree

To reduce time needed to render an image, hierarchical data structure called Bounding Volume Hierarchy (referred to as BVH) can be used. BVH tree lowers the number of primitives that need to be checked when evaluating which primitive will be hit first by a ray. Main idea behind the hierarchical data structure is that only geometric primitives that are close to the trajectory of a ray are evaluated. Geometric primitives are wrapped in axis aligned boxes that can be evaluated faster than complex primitives. Nearby primitives are grouped together creating a node described by a bounding box containing all its children. Then nearby nodes are again grouped together to a new node gradually creating a tree hierarchy. Now when evaluating rays trajectory, the root node of the BVH tree is checked first, if ray hits the parent node all of its children nodes are evaluated and again if one of them is hit the algorithm repeats itself until leave node is reached where the more complex primitives are evaluated [29].

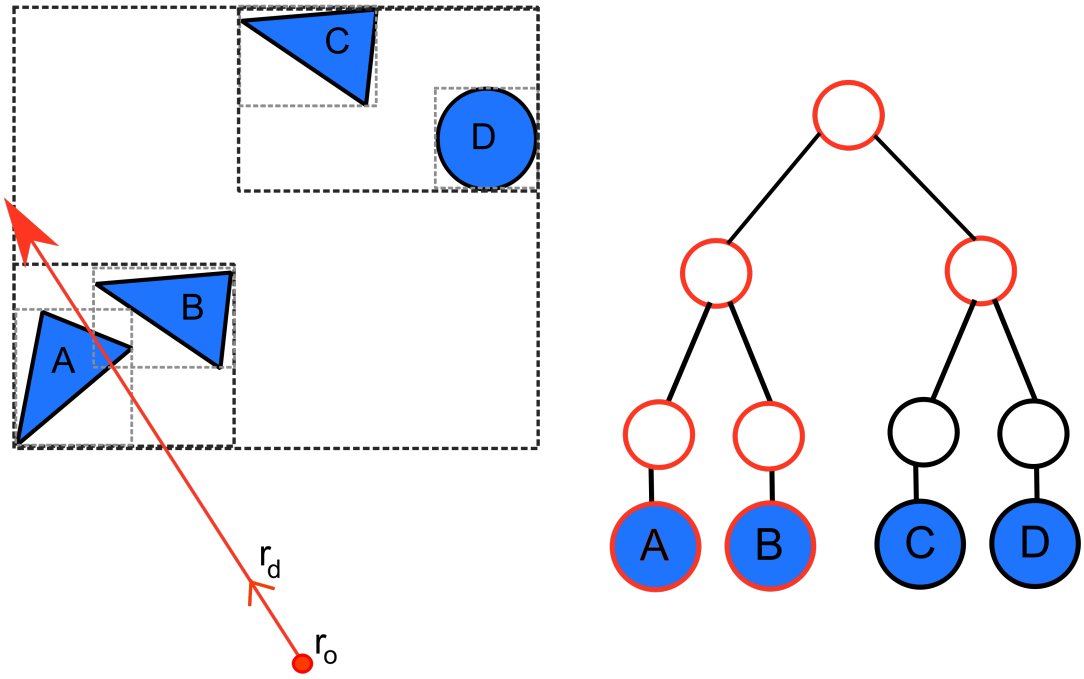

Figure 6.2. Traversing BVH tree diagram.

In the diagram 6.2 BVH tree is constructed from primitives A, B, C, D. White nodes contain bounding box encapsulating all of the nodes children. Blue nodes represent the actual geometric primitives used in a scene. Using the BVH traversal algorithm for the red ray only the nodes with red outline (A and B) are checked [29].

Primitives with large volume bounding boxes compared to their volume can lead to ineffective BVH tree because probability of false positives increases with higher ratio of bounding box volume to the primitives volume.

6.4 Ray tracing library

Ray tracing is a library containing implementations of ray tracing algorithms that are often highly optimized, photo-realistic and supporting many different devices and architectures. Using these libraries it is possible to write software utilizing ray tracing without the need for an implementation from scratch. The libraries can offer pipelines that allow a high level of customization of the rendering process.

6.5 Intel OSPRay

Intel OSPRay is an open source, portable and scalable ray tracing library. Purpose of OSPRay is to provide a powerful and easy-to-use rendering library that can be used to easily create ray tracing based applications. OSPRay can be utilized for both rendering using CPU and rendering using GPU. OSPRay builds on top of Intel Embree, Intel Open VKL and Intel Open Image Denoise. Intel Embree is high-performance ray tracing library incorporating high quality and performative data structures and algorithms. Intel Open VKL is a collection of high performance volume computation

kernels, containing algorithms and data structures commonly used in ray tracing. Intel Open Image Denoise is an open source library with denoising filters for images rendered with raytracing, that allows the creation of images of comparable quality with fewer samples of a scene [30].

To render an image in Intel OSPRay we need to specify the type of renderer that will be used for the image generation, world that can contain geometric models, volumetric models and lights and specify camera to capture the world.

Intel OSPRay offers materials for rendered objects that can be chosen based on desired performance and final look of the rendered object. Each material can be customized by changing values of its properties. For example, for the "Principled" material we can choose base color, edge color, how metallic the material is, how rough the material is and many more options. All of these properties can be specified by a texture for even more control of the resulting look. This allows for a high degree of customizability but shading technique is not possible without additional modules [31].

6.5.1 Renderer

Renderer is the central object for rendering in Intel OSPRay. Different renderers implement different features and support different materials. OSPRay renderers support adaptive accumulation which speeds up rendering by splitting image into regions and sampling pixels multiple times only in regions where variance is higher than specified threshold. OSPRay contains 3 renderers, SciVis renderer, Ambient occlusion renderer and Path tracer. SciVis Renderer is a fast raytracer for scientific visualization, it supports volume rendering and ambient occlusion. Ambient occlusion renderer supports only a subset of SciVis renderers features to gain performance. Lights are not supported, the main shading method is ambient occlusion. Path tracer supports soft shadows, indirect illumination, volumes with multiple scattering and realistic materials [30].

6.6 NanoRT

NanoRT is a ray tracing kernel [32] implementing BVH data structure allowing efficient ray intersection evaluation. It supports custom geometric primitives that can be implemented to work with the implemented BVH data structure. It is cross platform and can utilize OpenMP parallel programming API to build BVH tree taking advantage of parallelization.

6.6.1 Custom primitive

To create a custom primitive definition of data representing the primitive is needed. Next class resolving the bounding box calculations and a class with a predicate function used build BVH tree. Lastly Class that resolves ray intersections and class that holds data about the intersection for current ray traversal [32].

Chapter 7
Extrusion primitive

In this chapter we discus process of creating custom geometric primitive representing our geometric model of 3D printed material shown in figure 7.1. This geometric primitive then can be used while rendering image using ray tracing.

7.1 Data

Data needed to represent the geometry of extruded material shown in figure 7.1 are position of two points (A and B), normals of the two cutting planes ($\vec{n_1}$ and $\vec{n_2}$), layer width w and layer height h. The distance l between points A and B can be additionally saved in the data so there is no need to calculate it later during intersection testing.

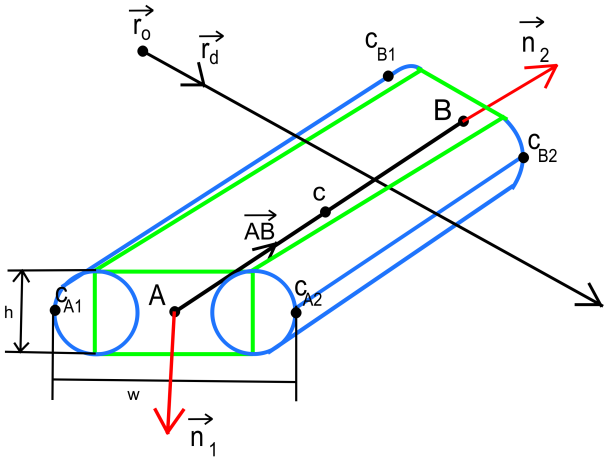

Figure 7.1. Geometry created by extruding cross section along a line segment and cut by 2 planes.

7.2 Predicate

For NanoRT to build the BVH tree it needs predicate that tells it which bounding boxes should be grouped together. NanoRT uses surface area heuristic (SAH) during BVH building process. The idea behind SAH predicate is to minimize probability of ray that passes through a parent node to hit both of its children (assuming binary partitioning) by minimizing surface area of the bounding boxes containing the children [33]. NanoRT uses greedy algorithm with use of this predicate to group the primitives into nodes. NanoRTs SAH predicate for triangles returns true if position given to it is behind the center of a primitive along a specified axis. Refactoring this function for the primitive 7.1 is trivial, only change of the calculation of the center is needed resulting

in equation (1) where c is the calculated center, A and B are points of the line segment defining the primitive.

$$c = \frac{A + B}{2} \tag{1}$$

7.3 Bounding box

To calculate the bounding box of extrusion primitive calculation of the corners of the extrusion primitive is necessary. These points can be seen in figure 7.1 marked C_{A1}, C_{A2}, C_{B1}, and C_{B2}. To calculate these points we need tangents to the normals of the cutting planes. Normals of the cutting planes are aligned with the direction between point A and B so the z coordinate will always be 0. With this knowledge constructing tangent t_i is trivial resulting in equation (2).

$$t_i = (-n_{i_x}, n_{i_y}, 0) \tag{2}$$

Where n_{i_x} and n_{i_x} is x and y values of the appropriate normal of a cutting plane. Last remaining step is to calculate the distance from the point to the edge of the primitive in the direction of tangent. Using trigonometry we get equation (3) where d is the distance from the edge segment point to its corresponding normal n, and $\vec{AB}$ is normalized direction from point $\vec{A}$ to point $\vec{B}$.

$$d = \frac{w}{2 \cdot \vec{n} \cdot (\vec{AB})} \tag{3}$$

Using this knowledge we can calculate the corners and find minimum and maximum in x and y axis. Minimum and maximum in z axis is derived by adding/subtracting half of layer height from z coordinate of point A or B.

7.4 Ray intersector

When calculating intersection of a ray with extruded primitive we need to find signed distance t the ray travels from its origin r_o in a direction r_d before it hits the extrusion primitive, then using equation (4).We can get the intersection point.

$$p = r_o + r_d \cdot t \tag{4}$$

The primitives outer surface not including caps can be split into four simple shapes that are then cut by the two cutting planes (later referred to as sub-primitives), two infinite cylinders with axis in direction $\vec{AB}$. Two infinite planes bounded on sides by edges in direction $\vec{AB}$ called flats. The sub-primitives are shown in figure 7.2 moved away from each for better readability.

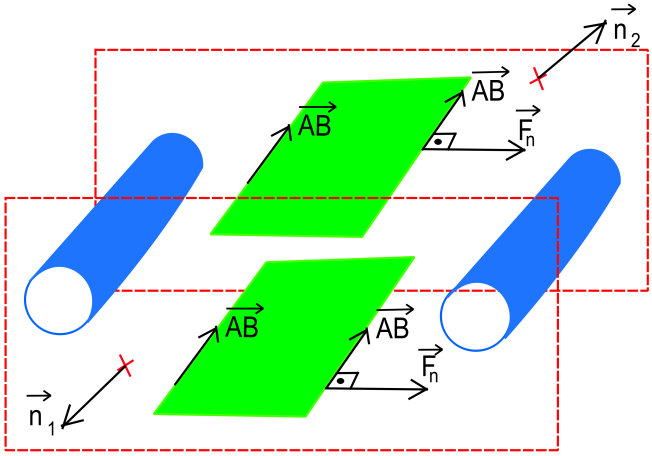

Figure 7.2. Extrusion primitive divided into 4 sub-primitives.

Signed distance is calculated for each of these sub-primitives then the smallest positive one is selected as the result of the ray and extrusion primitive intersection.

7.5 Cylinder intersection

Intersection between infinite cylinder and a ray can be derived from the equation (5) describing cylinder.

$$\|p - ((p - c_o) \cdot c_d) \cdot c_d\|^2 = r^2 \tag{5}$$

p is point on the surface of the infinite cylinder, c_o is point on the axis of the cylinder, c_d is direction of the cylinders axis and r is the radius of the cylinder. Now if p is substituted for using (4) we get (6).

$$\|t(r_d - r_d \cdot c_d \cdot c_d) + (r_o - c_o - (r_o - c_o) \cdot c_d)\|^2 = r^2 \tag{6}$$

Which can be simplified to get (7).

$$\|t \cdot A + B\|^2 = r^2 \tag{7}$$

This is solved as a quadratic equation:

$$t_{1,2} = \frac{-b \pm \sqrt[2]{b^2 - 4ac}}{2a}, \tag{8}$$

where

$$\begin{aligned} a &= A \cdot A \\ b &= 2 \cdot A \cdot B \\ c &= B \cdot B - r^2 \end{aligned} \tag{9}$$

To calculate the signed distance using our data we replace c_d with $\vec{AB}$ and c_o is calculated using equation (10).

$$c_o = \vec{A} + \frac{w - h}{2} \cdot \vec{t_n} \tag{10}$$

7.6 Flat top/bottom intersection

The extrusion primitive has two flats, at its top and bottom with two always parallel edges in the direction of the line segments $\vec{AB}$ used to create the primitive. Because the flats are always aligned with x/y plane we use equation (11) to see when the ray is in the same height as is the flat surface using equation where F_z is z coordinate of the flat.

$$t = \frac{F_z + h - r_{o_z}}{r_{d_z}} \tag{11}$$

Then the expresion (12) is used to check if the ray intersection is inside the primitive if it was infinitely long.

$$((p - F_1) \cdot F_n) \cdot ((p - F_2) \cdot F_n) < 0, \tag{12}$$

where p is point on rays trajectory in time t, F_1 and F_2 are points on each parallel edge of the flat F_n is a vector perpendicular to the direction of both edges aligned with x/y plane and it is shown in figure 7.2.

7.7 Intersection without caps

Now with information about hits of the sub-primitives the minimal time t is chosen and if point p derived using (4) falls between the 2 cutting planes, which is true if (13) is True ,with the assumption that the dot product of 2 normals of the planes is more than zero (one of the 2 normals can be flipped to satisfy the assumption).

$$((p - P_1) \cdot P_{1n}) \cdot ((p - P_2) \cdot P_{2n}) < 0, \tag{13}$$

where P_1, P_2 are points on each plane (points A and B describing the primitive in 7.1 and P_{2n}, P_{2n} are according normals of the two cutting planes. If the point with minimal time t does not satisfy the equation (13) the next smallest time is chosen and the procedure is repeated until a point satisfying it is found or until there are no more intersections.

7.8 Cap intersection

If the intersection between ray and cap is needed it can be checked by finding intersection of the ray and each of the cutting planes. Then for each plane the intersection point between the plane and the ray is checked if it is inside any of the 2 infinite cylinders or if is inside the rectangular box highlighted in green 7.1. Intersection between plane and ray is calculated using equation (14).

$$\frac{t = (p_0 - r_0) \cdot p_n}{r_d \cdot n} \tag{14}$$

Then equation (5) is used to check if the intersection point is inside any of the 2 cylinders and to check if it inside the rectangular box modified version of expression used for the flats (15) is used.

$$((p - e_1) \cdot e_{1n}) \cdot ((p - e_2) \cdot e_{1n}) < 0 \tag{15}$$

where e_1, $e2$ are replaced by 2 points on the intersection of plane with the left and right edge of the rectangular box. e_{1n} is replaced by the tangent of the plane parallel with the x/y plane and lastly the intersection points z coordinate must be between the z coordinates of the 2 flats.

However when rendering the geometry created from the 3D printer's G-code it is an uncommon occurrence to see any of the caps of the extruded lines, so it can be skipped to save performance.

7.9 Intersection data

Any intersection data structure must hold the id of the hit primitive and the signed distance t. With the primitive 7.1 the type of sub-primitive that was hit is also saved i.e.top/bottom flat, left/right cylinder or front/back cap. After traversing the BVH tree and successfully finding intersection with a primitive, more information can be calculated like tangent and normals at the intersection point. Tangent is either the direction between points A and B if the circumference of the primitive was hit. If one of the caps was hit the tangent is the same as the tangent of the according cutting plane. Normals are also calculated based on the type of hit sub-primitive. The normals is up/down direction if the hit sub-primitive is one of the top/bottom flats. It is the same as normal of the hit cutting plane if one of the caps was hit or it is the normalized direction from the projection of the intersection point onto the according cylinders axis to the intersection point.

Chapter 8
Measuring reflection data

In this chapter we discus how reflection data of a material can be measured to approximate bidirectional reflectance distribution function and how to use the measured data to render images.

8.1 Bidirectional Reflectance Distribution Function

The Bidirectional Reflectance Distribution Function is a theoretical concept describing relationship between the luminance of incoming light and the illuminance of the reflected light in any given direction of incoming light (θ_i, ϕ_i) and any direction of reflected light (θ_d, ϕ_d) define by equation [34]. All the variables are shown in figure 8.1 in coordinate space defined by normal $\vec{n}$, tangent $\vec{t}$ and bi-tangent $\vec{bt}$ of surface described by the BRDF.

$$BRDF(\theta_d, \phi_d, \theta_i, \phi_i) = \frac{dL(\theta_d, \phi_d)}{dE(\theta_i, \phi_i)} \tag{1}$$

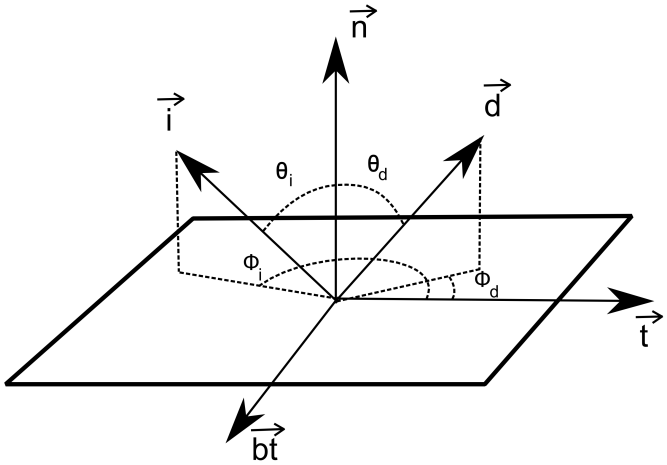

Figure 8.1. BRDF coordinates diagram.

The BRDF is often approximated using analytical mathematical function. For example ideal matte surface called Lamberian surface [35] can be described with function (2),

$$I(\theta) = I_d cos\theta \tag{2}$$

However more realistic and complex materials need more complex definitions of their BRDF or the BRDF can be measured for many different directions to create point cloud of data from which the values of BRDF can be read. To measure the required

data machine called scatterometer is used, specifically Mini-Diff V2 [36] for purposes of this work. This data can be then used in different domains like multimedia industry, simulation software or the car industry.

8.2 Mini-Diff V2

Mini-Diff V2 measures light reflected from a sample placed under the machine as shown in figure 8.2. Mini-Diff V2 measures materials using incoming light from four different angles of incidence θ_i (0°, 20°, 40°, 60°) for red, blue and green light with constant ϕ_i. So the measured BRDF is isometric, meaning it returns same value for all ϕ_i which is set to 0.

Figure 8.2. Mini-Diff V2. Image taken from [36]

Before measuring a sample of a material calibration is needed [36]. To calibrate BRDF measurement black and white reference samples are used before the start of each measuring session. The reference samples are depicted in figure 8.3.

Figure 8.3. Samples used to calibrate Mini-Diff V2. Image taken from [36]

Measurement is performed by placing a sample of a material under the machine and starting the measuring procedure in the provided software Mini-Diff with option to select which of the three lights will be measured and for which of the four directions of incoming light will the measurement be performed. It is possible to select multiple options at once for a measurement [36].

Figure 8.4. Samples used to measure BRDF.

In the 3D preview show in figure 8.5 of green PLA measurement the measured value for green light from incoming light under 0° is represented by both height and color of the shown shape. In the 2D view show in figure 8.6 the same sample is presented but in RGB preview composed from all three measured wavelengths.

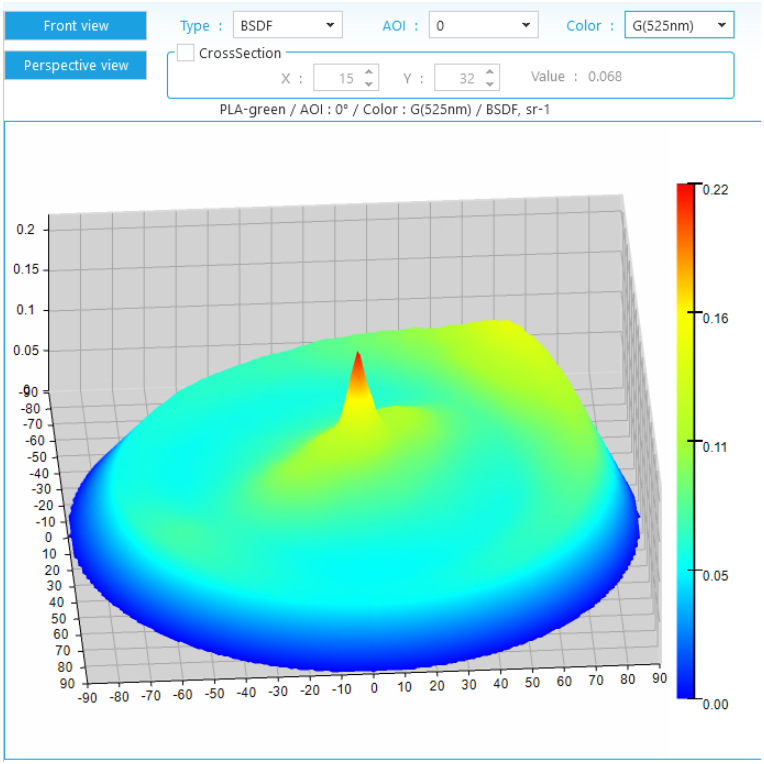

Figure 8.5. 3D preview in Mini-Diff software.

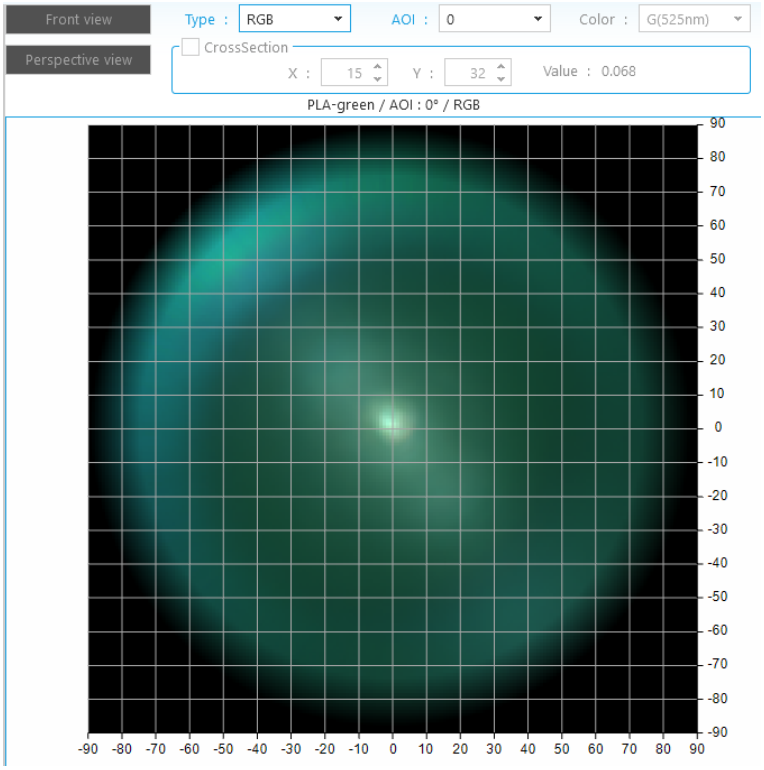

Figure 8.6. 2D preview in Mini-Diff software.

8.3 Textures from measured BRDF data

To use measured data to render images it is converted to textures. To convert the data application brdf_gui developed by Lukáš Cezner was used [37]. Data exported from the Mini-Diff software are separated to files by color of measured light (red, green, blue), so first brdf_gui is used to merge these files together. This merged file the brdf_gui application converts to a texture into EXR format. EXR format support custom meta data for generated image. In the generated textures meta data is the information about the tiled layout, what color channels are used and what angles of incoming light θ_i were used for the measurements. The result of the measurement of green PLA shown in figures 8.5 and 8.6 measurement is converted into a texture can be seen in figure 8.7.

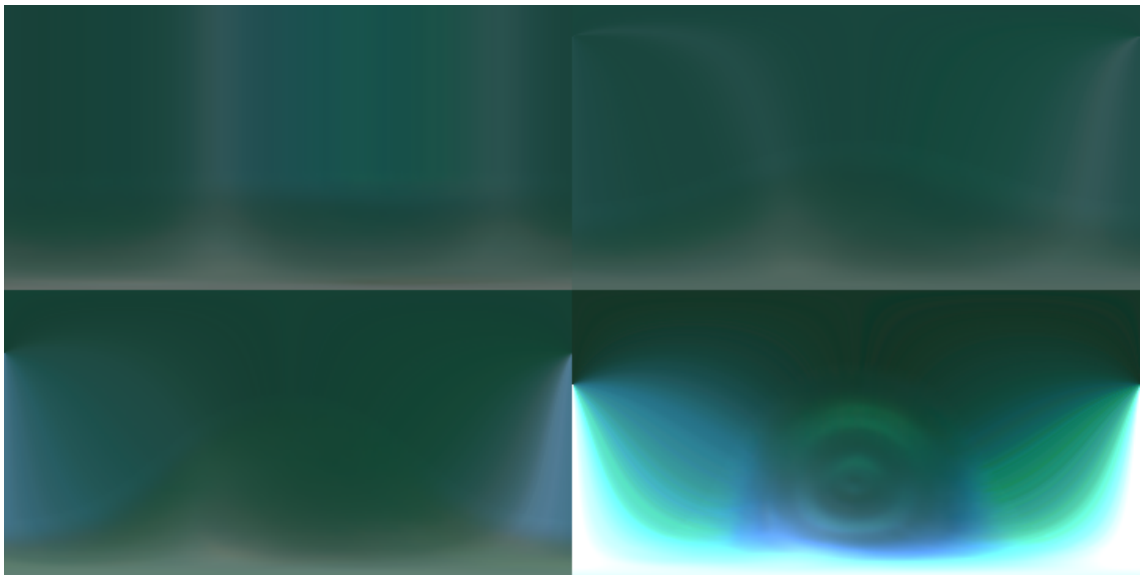

Figure 8.7. Texture created from green PLA sample.

Created texture is composed out of 4 tiles each corresponding to one of the measured incoming light angles θ_i. Each tiles horizontal coordinate corresponds to values of angle ϕ_d of the reflected light. Vertical coordinate then corresponds to values of angle θ_d [37].

8.4 Measured textures in NanoRT

Nanort engine traverses ray through a scene and returns intersection information containing rays direction, normal and the tangent of the intersected surface. Combined with information about direction of incoming light uv coordinates of the texture shown in figure 8.8 can be calculated.

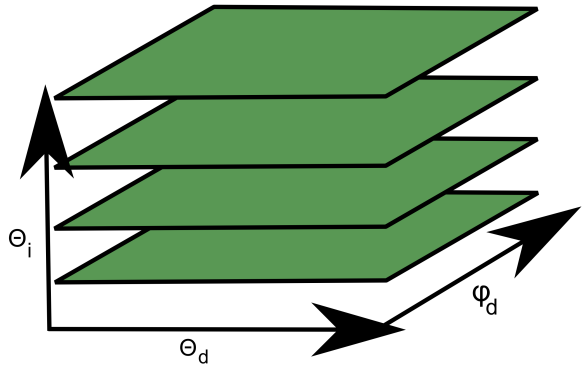

Figure 8.8. 3D texture created from BRDF measurement.

Using coordinate system defined by normal, tangent and bi-tangent previously defined in figure 8.1 with ϕ_i set to 0. Angle θ_i between normal and incoming light is calculated as:

$$\theta_i = acos(\vec{n} \cdot \vec{l}) \tag{3}$$

Angle θ_d is calculated similarly as:

$$\theta_d = acos(\vec{n} \cdot \vec{1}) \tag{4}$$

and lastly angle ϕ_d is calculated between projection of $\vec{i}$ and $\vec{d}$ on a plane with normal $\vec{n}$ using function atan2. Projections are calculated using:

$$\vec{v_p} = \vec{v} - \vec{n} \cdot (\vec{v} \cdot \vec{n}) \tag{5}$$

where $\vec{v_p}$ is vector $\vec{v}$ projected on plane with normal $\vec{n}$. The function atan2(x, y) returns angle $0 - 2\pi$ between positive x axis and vector (x, y), resulting in equation:

$$\phi_d = atan2(\vec{n} \cdot (\vec{d_p} \times \vec{i_p}), \vec{i_p} \cdot \vec{d_p})$$

where $\vec{d_p}$ is vector $\vec{d}$ projected on plane with normal $\vec{n}$ and $\vec{i_p}$ is direction to incoming light $\vec{i}$ projected on the same plane.

8.4.1 Direct lighting

During direct lighting color value is calculated for each individual light that shines on the intersected surface, then these values are added together. The lights are approximated as infinitely small at infinite distance. In this approach light reflected from the environment is not taken into account resulting in less accurate result, shown in figure 8.9.

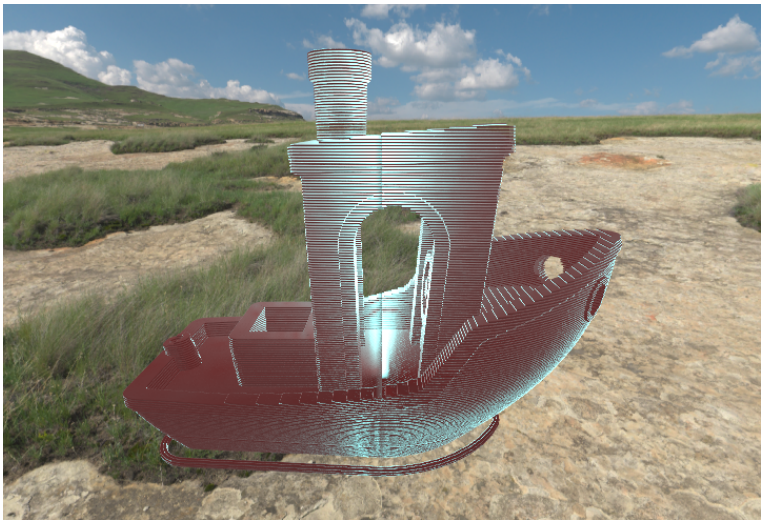

Figure 8.9. Red benchy rendered with indirect lighting.

8.4.2 Environment mapping

During environment mapping light is sampled across the whole hemisphere around the normal of the intersected surface. In each direction light value is read from environment map. Environment map is a 2D texture that can be sampled using spherical coordinate of longitude and latitude. The environment map is created by capturing incoming light in every direction in the real world and saving it to the 2D texture. Example is show in figure 8.10.

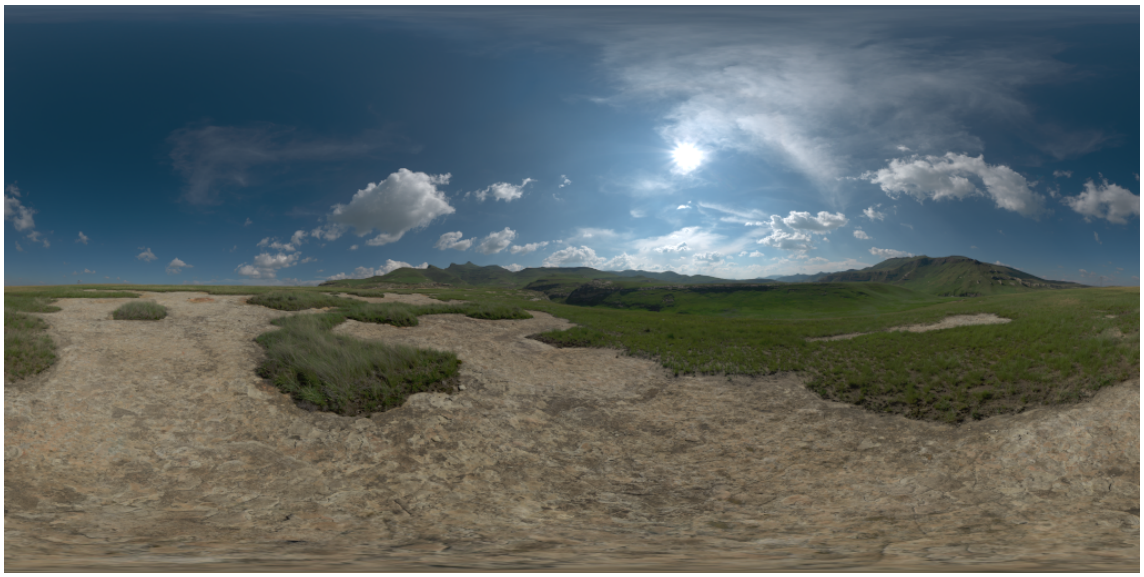

Figure 8.10. Enviroment map [38].

To sample light coming from a direction, the direction is transferred to spherical coordinates. Value is then selected from the texture based on x coordinate across the whole texture ranging from 0 to 2π. and y coordinate across the whole texture ranging from 0 to π.

To calculate the color value of the reflected light the equation (6) for integration around the whole hemisphere of incoming light is used.

$$L_{out}(\vec{\omega_{out}}) = \sum_{i=0}^{i_{max}} \sum_{j=0}^{j_{max}} w_{i,j} \cdot f(\vec{\omega_{i,j}}, \vec{\omega_{out}})) \cdot L_{in}(\vec{\omega_{i,j}}) \cdot cos\theta_j \tag{6}$$

where as can be seen in figure 8.11 $\vec{\omega_{out}}$ is the direction towards the camera, $w_{i,j}$ is the weight of the light sampled in chunk with center in direction $\vec{\omega_{i,j}}$, θ_j is angle between the normal and sampled chunk direction and lastly function f samples the BRDF texture and L_{in} returns value of accumulated light of the chunk read from the environment map. To properly calculate incoming light each pixel contained in the chunk must be evaluated. For this reason we downsample the environment map to increase performance.

After accumulating value from all pixels in area between ϕ_{i-1}, ϕ_{i+1} and θ_{j-1}, θ_{j+1} we calculate the weight $w_{i,j}$ of the sample. To calculate the weight the surface area of the sampled chunk of the hemisphere is needed which can be seen in figure 8.11. $\omega_{i,j}$ is a direction to the middle of the sampled chunk.

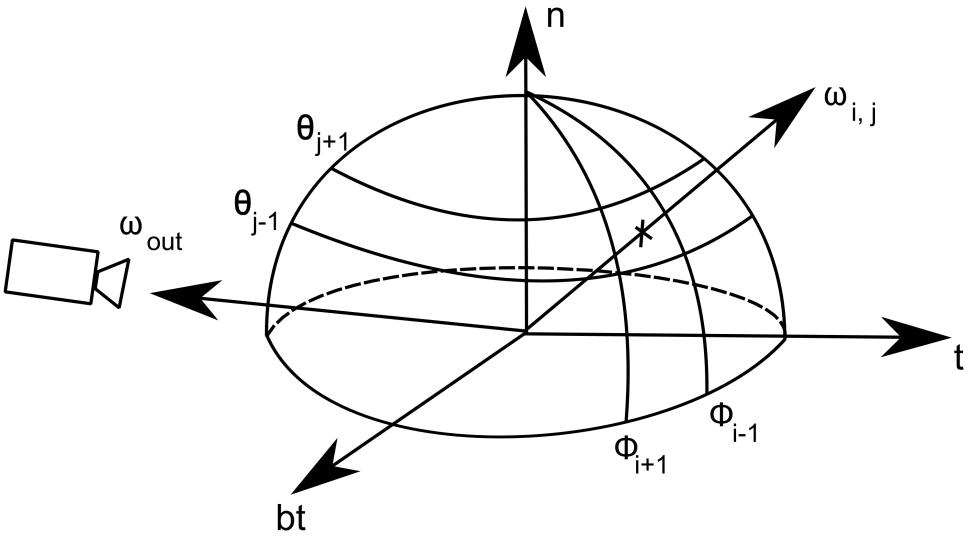

Figure 8.11. Sample of hemisphere around intersected surface.

To calculate the surface we first calculate the horizontal ring surface between current and sampled next angle where we are sampling direction in linearly interpolated halfway point between these 2 angles θ_j 8.11.

$$S_j = 2 \cdot \pi[(1 - \frac{cos\theta_j + cos\theta_{j+1}}{2}) - (1 - \frac{cos\theta_j + cos\theta_{j-1}}{2})] \tag{7}$$

then in similar fashion the surface is divided by the surface of the vertical strip 8.11.

$$w_{i,j} = S_j \cdot \phi_{i-1} \frac{\phi_{i+1} - \phi_{i-1}}{2 \cdot 2 \cdot \pi}$$

resulting image is rendered slower due to computational complexity of numeric integration but with more realistic results 8.12.

Figure 8.12. Benchy rendered using environment mapping.

Chapter 9
Results

To compare performance of NanoRT and OPSRay based renderers same shader called ambient occlusion is used for both types. Ambient occlusion calculates how much is each intersected surface affected by ambient lighting. Points surrounded by a large amount of geometry are displayed as darker ones and the other way around as shown in figure 9.1. To approximate the result of integrating across the whole hemisphere multiple checks if a secondary ray has a clear path from the impacted point through the scene without intersecting without any other geometry are performed in random directions [39]. Both OSPRay and NanoRT renderers were configured to test one random direction per hit with infinite maximal travel distance for the test ray and to sample 256 times per each pixel, models and camera position are identical.

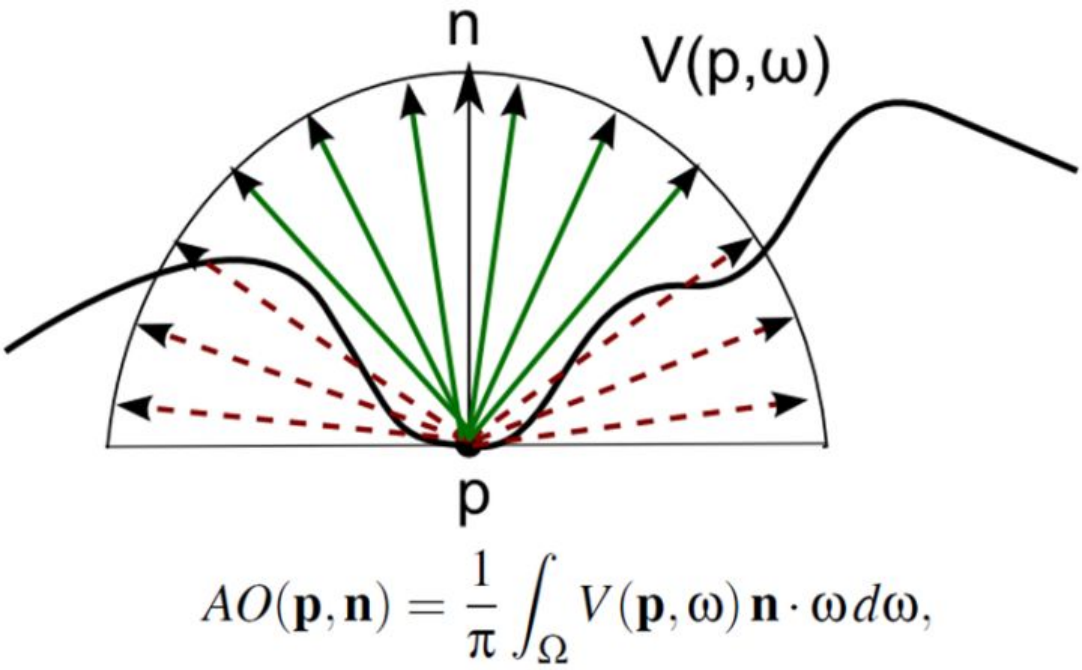

Figure 9.1. Definition of ambient occlusion shader. Image taken from [39]

In the results each plot has a legend describing what renderer and what geometry was used. First half of each entry is self explanatory, OSPRay if renderer using Intel OSPRay library and nanort if the renderer using NanoRT was used. Second part describes geometry used to approximate extruded plastics shape. Quad means tessellation with 4 vertices per segment border. Hex is for 6 vertices, Poly for 10 and Prim means use of extrusion primitive created for NanoRT.

Images generated using ambient occlusion shader are shown in figures 9.2 to 9.4 using small model Benchy [40], medium model Stanford Dragon [41] and big model Cute Raccoon [42].

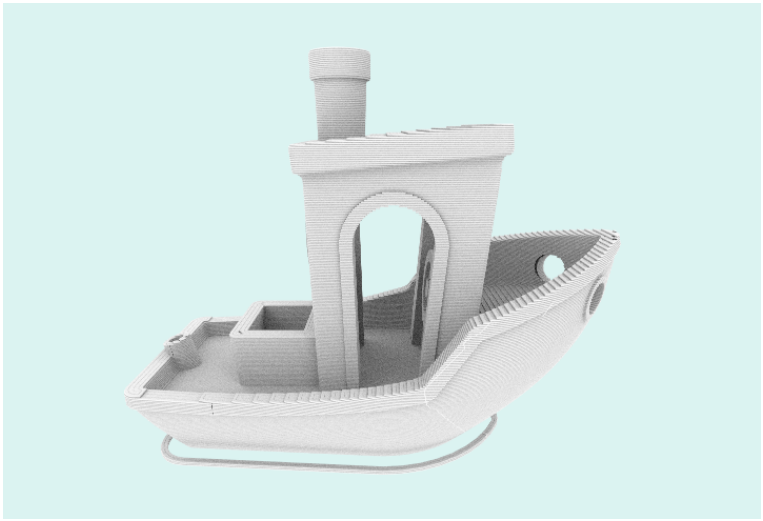

Figure 9.2. Rendered Benchy model using AO shader. Volume of the model is 15.6 cm^3.

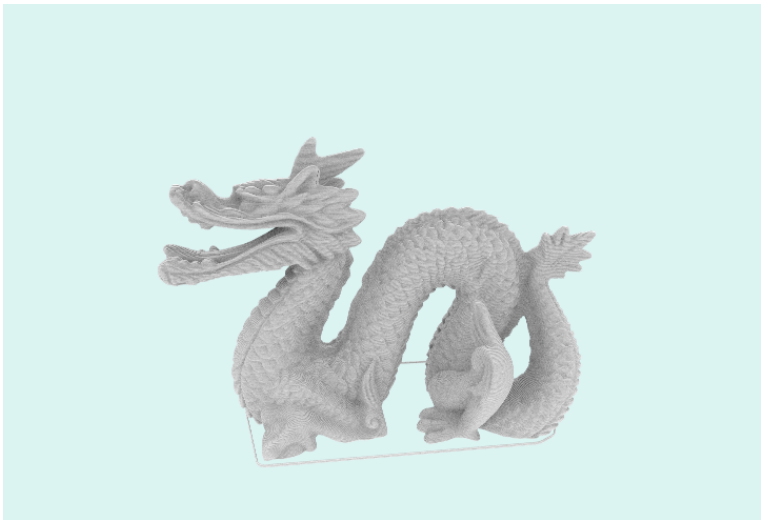

Figure 9.3. Rendered Stanford dragon model using AO shader. Volume of the model is 140.8 cm^3.

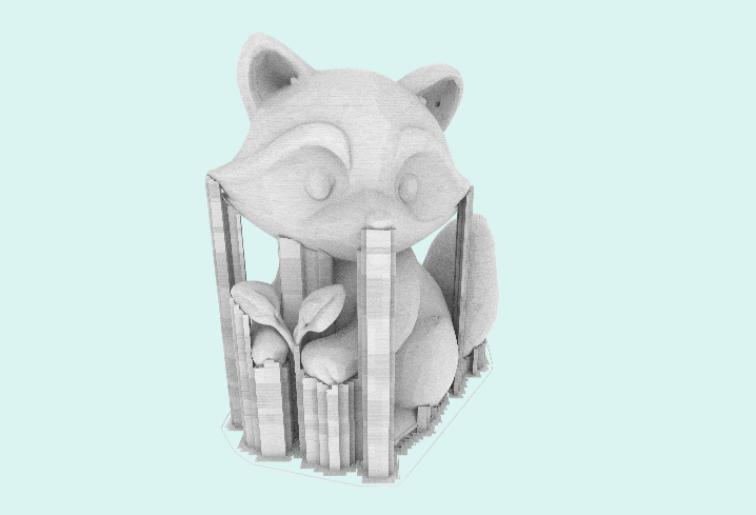

Figure 9.4. Rendered Raccoon model using AO shader. Volume of the model is 1508.8 cm^3.

9.1 Division of long segments

To increase performance we found the optimal ratio of maximum allowed length of line segments describing extrusion primitive (if line segment is too long it is split to equal parts with maximum length possible) by testing performance using the ambient occlusion shared.

Each figure contains four plots, on the top left plot displays relation of time the renderer took to generate an image ignoring the time it took to prepare optimization data structures (building BVH tree etc.) on Y axis compared to maximal allowed ratio of length to width of the extrusion primitive on the x axis. X axis is the same in all plots. On the top right is a plot displaying relation of times renderers needed to prepare the optimization data structures. Bottom left plot shows how many times the numbers of primitives increased after segmenting of long extrusion primitives. Lastly bottom right plot shows total time of both renderer preparation and the rendering of an image.

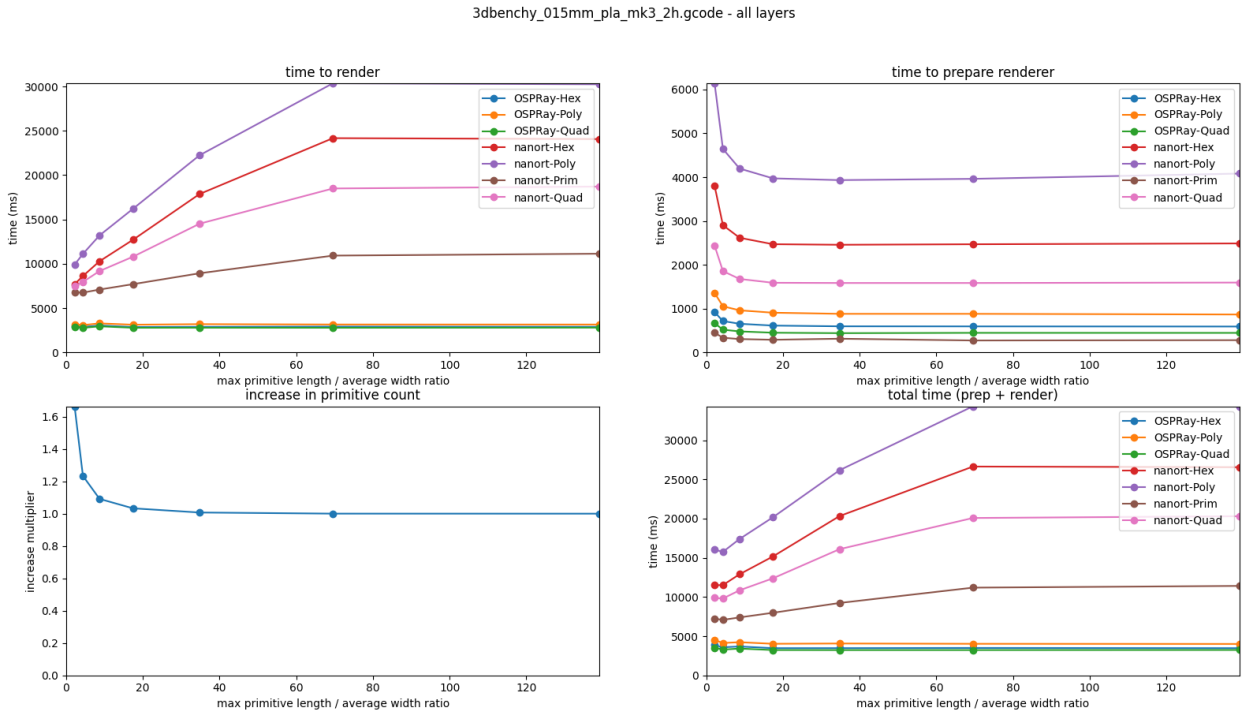

Figure 9.5. Benchy all layers.

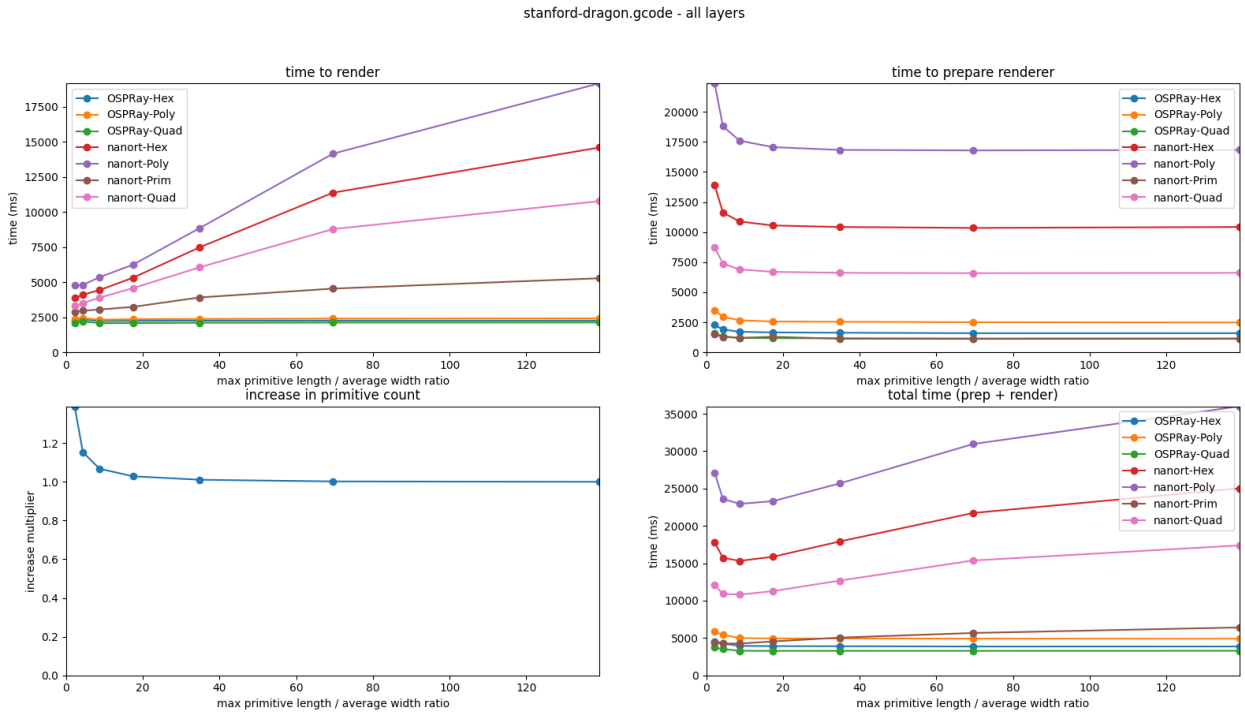

Figure 9.6. Stanford Dragon all layers.

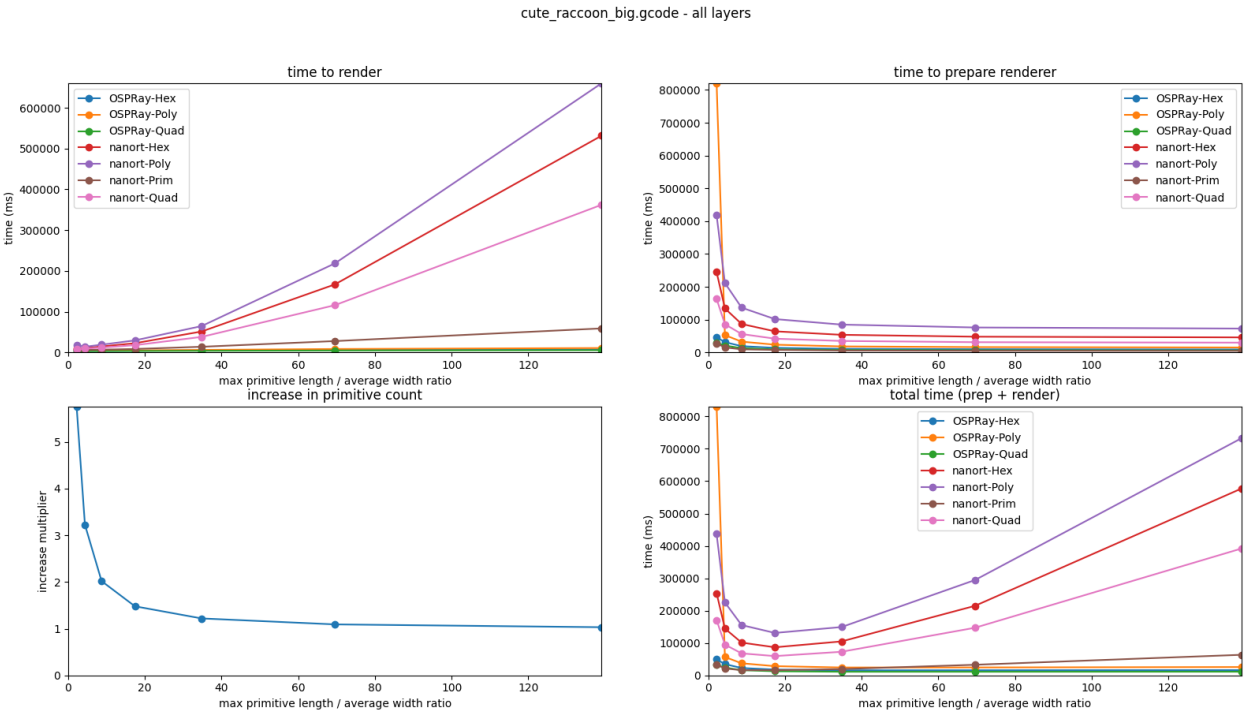

Figure 9.7. Racoon Big all layers.

In the next three graphs using the same three models is shown relation of time needed to generate geometry using tessellation or creating extrusion primitives and the ratio of maximal allowed length to average width.

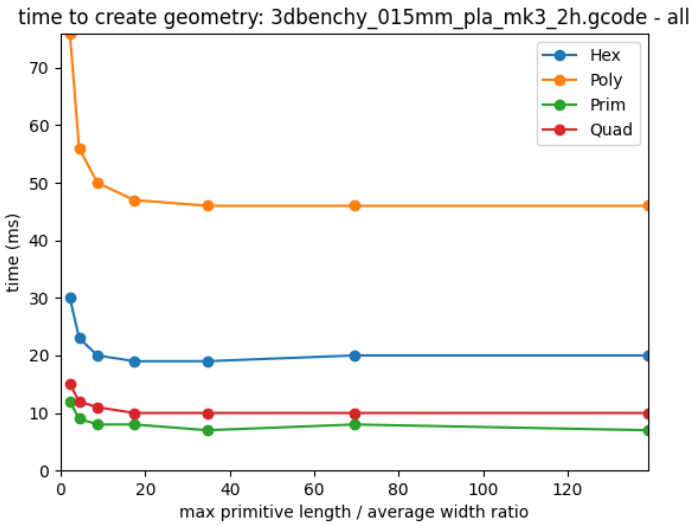

Figure 9.8. Benchy time to prepare geometry.

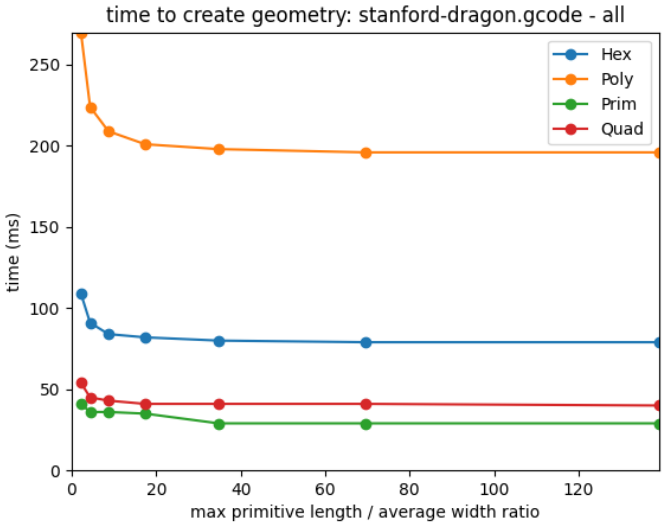

Figure 9.9. Stanford Dragon time prepare geometry.

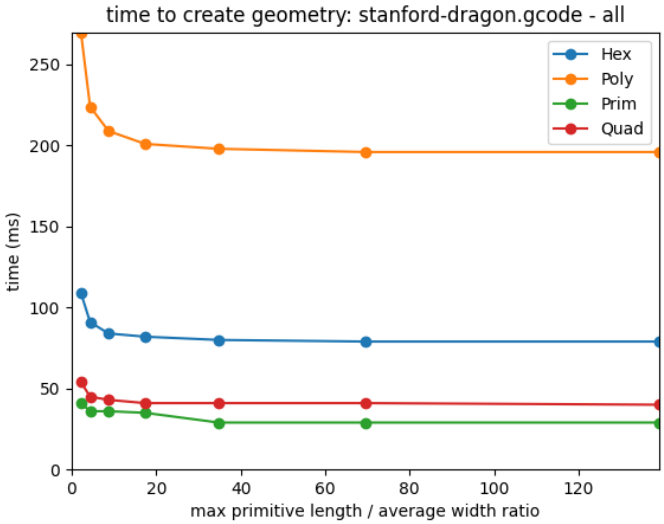

Figure 9.10. Racoon Big time prepare geometry.

The next three graphs 9.11 to 9.13 using the same 3 models show total time to prepare geometry, prepare renderer and to render the image.

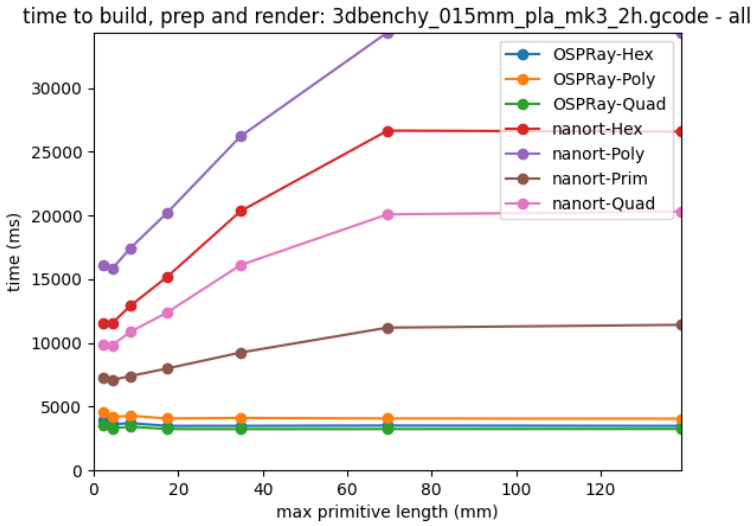

Figure 9.11. Benchy total time to prepare geometry, build and render.

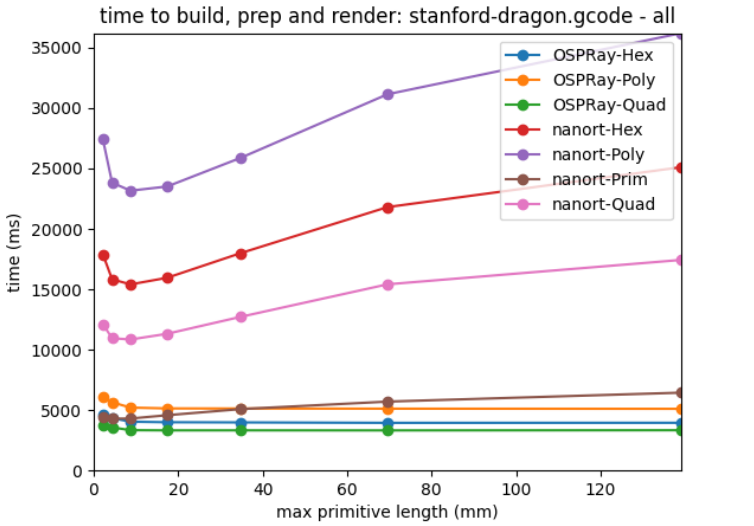

Figure 9.12. Stanford Dragon total time to prepare geometry, build and render.

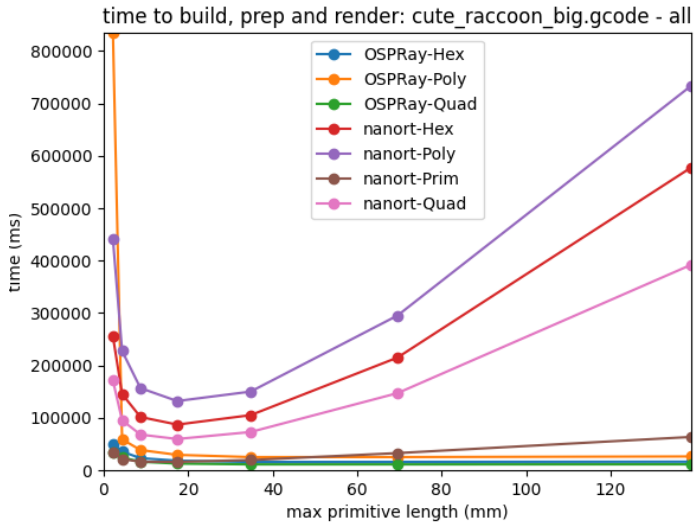

Figure 9.13. Racoon Big total time to prepare geometry, build and render.

Last graph 9.14 is zoomed in to better see results of fastest renderers from figure 9.13

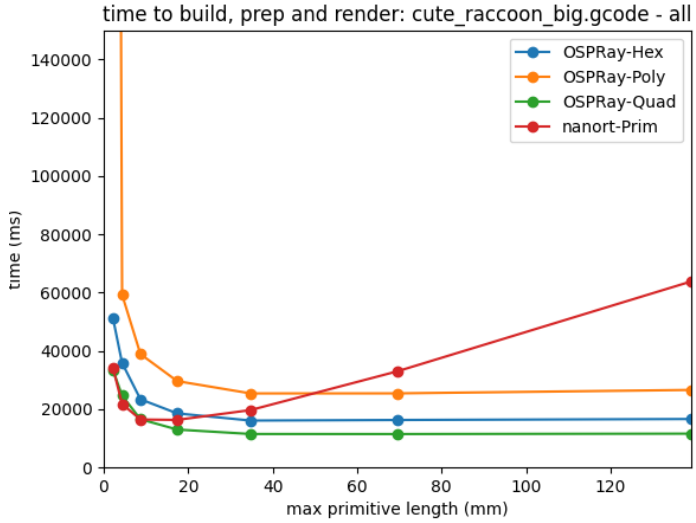

Figure 9.14. Racoon Big total time to prepare geometry, build and rende. Graph is zoomed in on faster results.

We can see that across sizes, decreasing the maximal allowed ratio of length to width of the primitives decreases time needed to render the image. However there is a point that decreasing the ratio starts worsening our results. The optimal ratio is between 8 to 16 depending on the model. All upcoming images were rendered using maximum allowed ratio of 16 using NanoRT renderer working with the extrusion primitive with 32 samples per pixel and each ray calculates color using measured BRDF and environment mapping. In each figure 9.15 to 9.26 is described by the name of the model and by volume of the model.

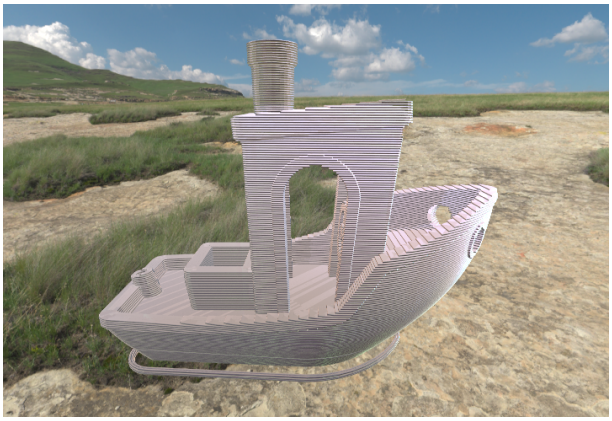

Figure 9.15. Benchy model [40]. 15.6 cm^3.

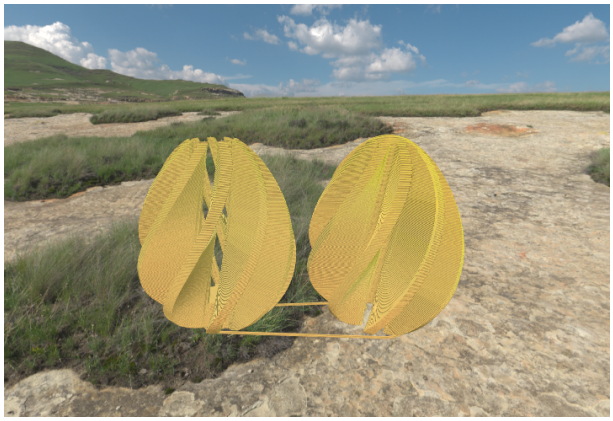

Figure 9.16. Spiral egg model [43]. 46.7 cm^3.

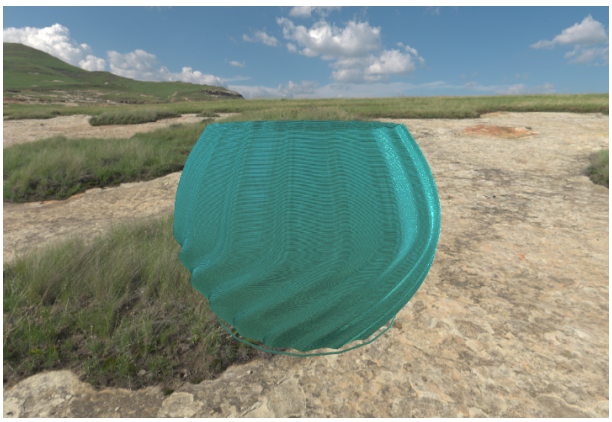

Figure 9.17. Swirl pot model [44]. 60.8 cm^3.

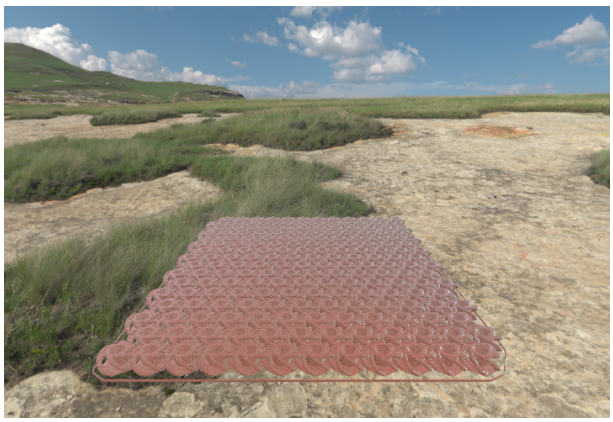

Figure 9.18. Nasa fabric model [45]. 86.9 cm^3.

Figure 9.19. Stanford dragon model [41]. 140.8 cm^3.

Figure 9.20. Bunny model [46]. 172.4 cm^3.

Figure 9.21. Rocket model [47]. 496.9 cm^3.

Figure 9.22. Cat model [48]. 215.2 cm^3.

Figure 9.23. Baby dragon model [49]. 1214.1 cm^3.

Figure 9.24. Maoi face model [50]. 1253.5 cm^3.

Figure 9.25. Owl model [51]. 1362.4 cm^3.

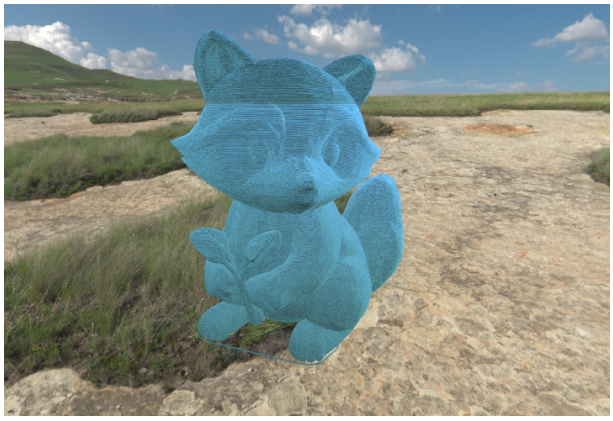

Figure 9.26. Cute raccoon model [42]. 1508.7 cm^3.

Model	G-code size	BVH build	Render	primitives
Benchy	2.7 KB	0.259 s	96.937 s	65084
Spiral egg	11.5 KB	1.004 s	72.410 s	280773
Swirl pot	24.0 KB	1.508 s	77.185 s	552636
Nasa fabric	64.7 KB	2.786 s	72.074 s	1077530
Stanford dragon	21.7 KB	1.550 s	71.312 s	559084
Bunny	12.6 KB	1.128 s	48.389 s	376960
Cat	10.6 KB	1.082 s	35.059 s	342491
Rocket	42.2 KB	3.394 s	37.686 s	1276321
Baby dragon	43.2 KB	4.192 s	66.560 s	1530100
Maoi face	23.2 KB	3.724 s	58.306 s	1167474
Owl	43.0 KB	4.461 s	66.091 s	1635285
Cute raccoon	39.2 KB	4.478 s	76.169 s	1626697

9.2 Photo comparison

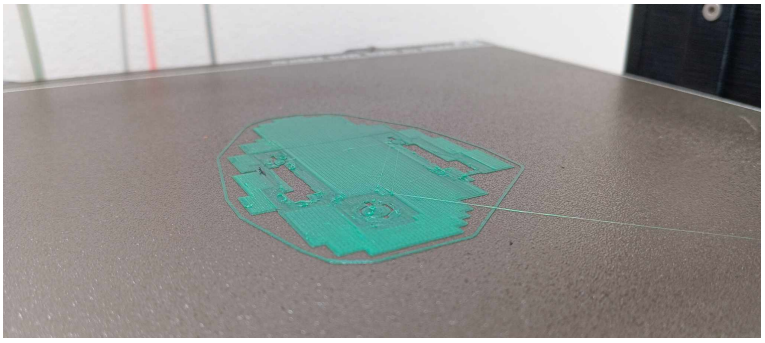

Figure 9.27. Printed out bunny first layer.

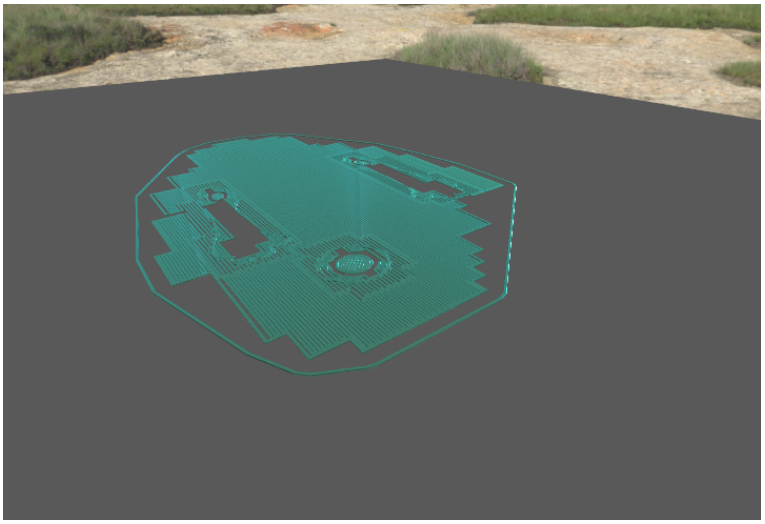

Figure 9.28. Rendered out bunny first layer.

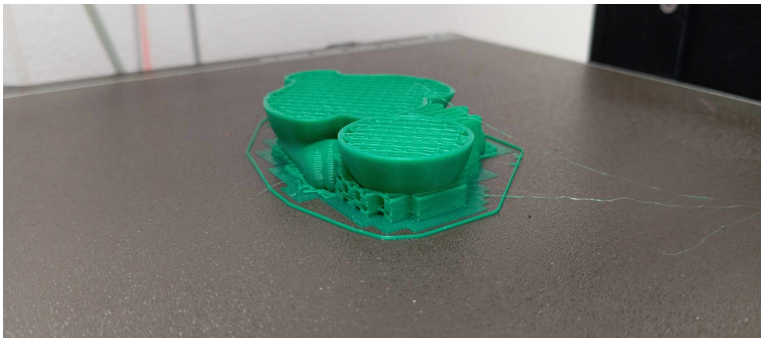

Figure 9.29. Printed out bunny 25

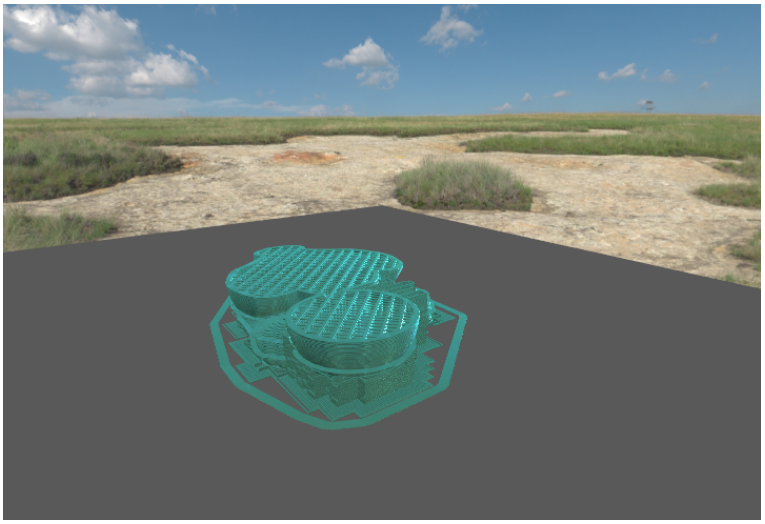

Figure 9.30. Rendered out bunny 25

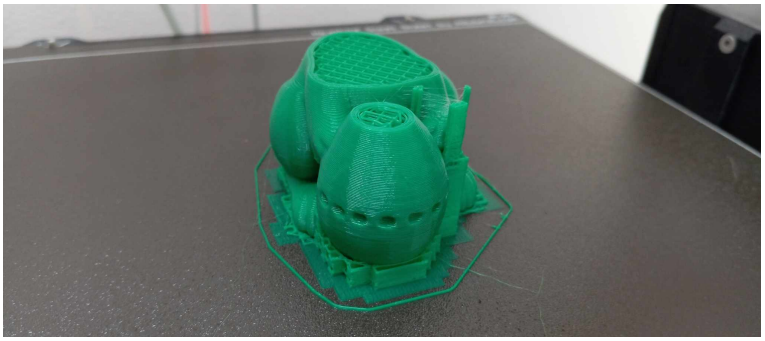

Figure 9.31. Printed out bunny 50

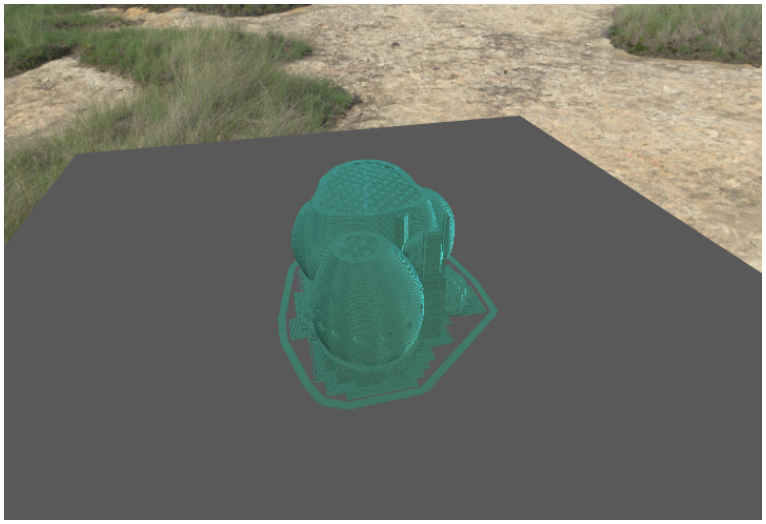

Figure 9.32. Rendered out bunny 50

Figure 9.33. Printed out bunny 100

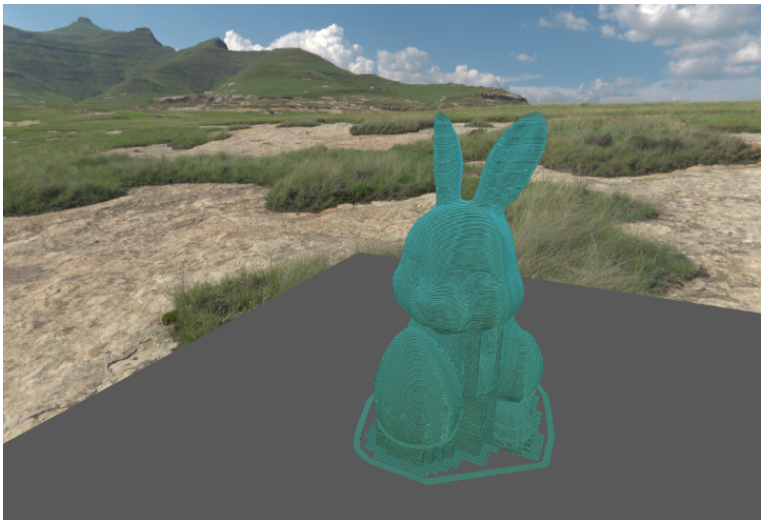

Figure 9.34. Rendered out bunny 100

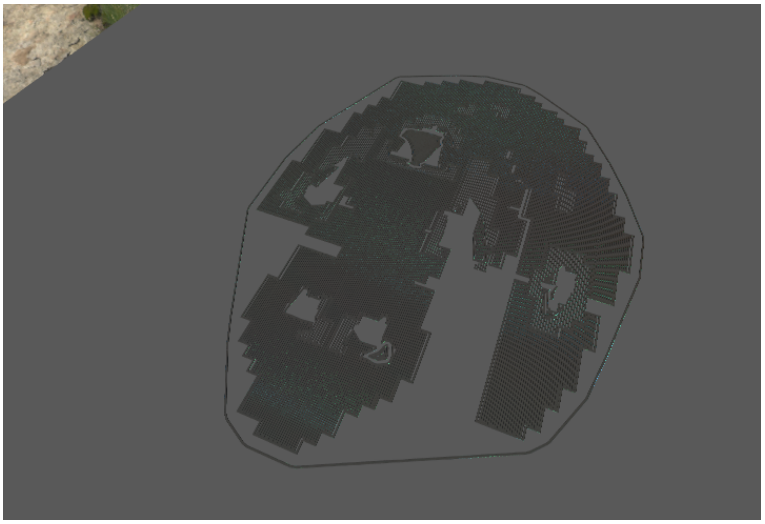

Figure 9.35. Rendered out cat first layer.

Figure 9.36. Printed out cat first layer.

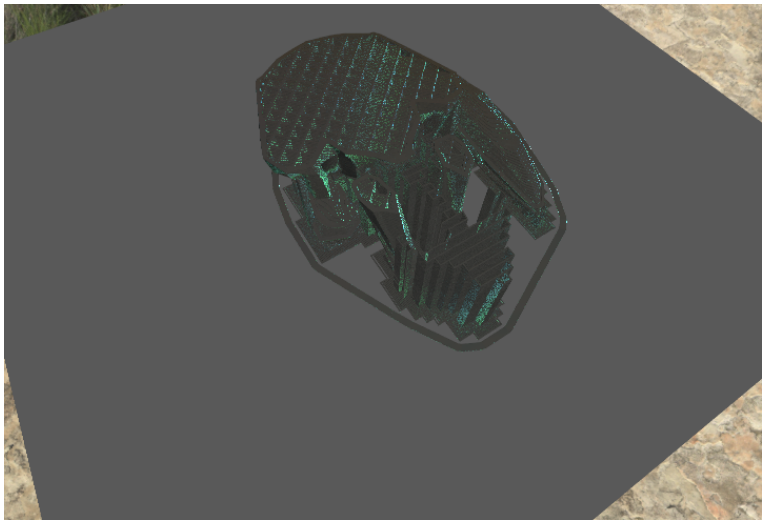

Figure 9.37. Rendered out cat 25%.

Figure 9.38. Printed out cat 25%.

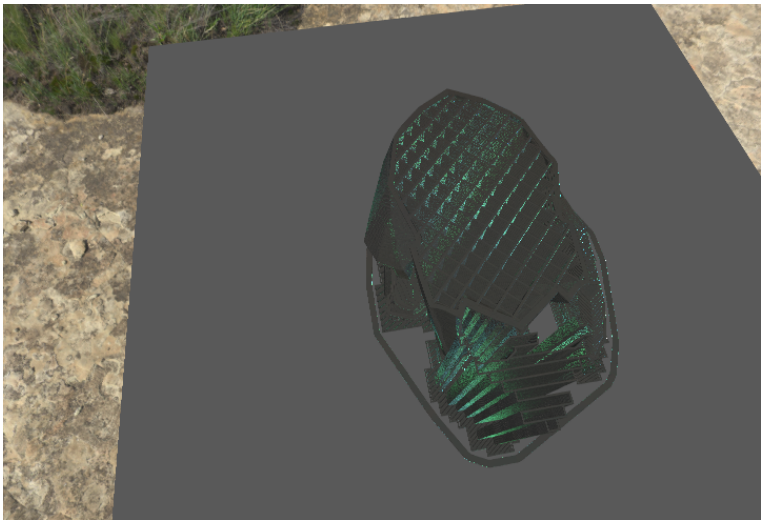

Figure 9.39. Rendered out cat 50%.

Figure 9.40. Printed out cat 50%.

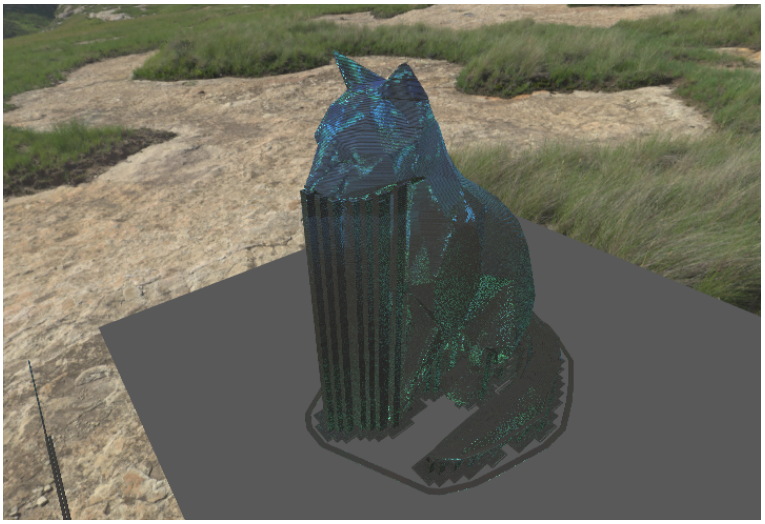

Figure 9.41. Rendered out cat.

Figure 9.42. Printed out cat.

Chapter 10
Conclusions

Goal of this thesis was to generate geometrical representation of a 3D printed part from G-code program for a 3D printer and then to render previews of the resulting printed object using ray tracing with measured BRDF data of materials used by 3D printers.

We devised a way to transfer G-code to a geometrical representation of 3D printed object and compared renderers we implemented using ray tracing libraries Intel OSPRay and NanoRT that worked with the generated geometry. The NanoRT renderer was also expanded to test intersection with extrusion primitive instead of triangular mesh and to use measured BRDF data with light sampled from environment map.

In comparison renderer using Intel OSPRay library was faster when rendering triangular meshes but NanoRT renderer using extrusion primitive instead of triangular mesh achieved comparable performance.

We renderer 12 models of different sizes using our NanoRT renderer using extrusion primitive and BRDF with environment mapping, the size of the model or size of the G-code used to generate the geometry did not significantly affect the resulting time needed to render an image.

We also compared photographs of 2 printed out models with rendered previews. Rendered geometry resembles printed object quite well during the printing process but to achieve higher level of photo-realism future work is needed.

References

[1] Protolabs Network. *3D printing*. https://www.hubs.com/guides/3d-printing/. Accessed on 11.19.2024.

[2] Lucas Carolo. *What Is FDM 3D Printing? - Simply Explained*. 2024. https://all3dp.com/2/fused-deposition-modeling-fdm-3d-printing-simply-explained/.

[3] Yiqiao Wang, Wolf-Dieter Müller, Adam Rumjahn, and Andreas Schwitalla. Parameters Influencing the Outcome of Additive Manufacturing of Tiny Medical Devices Based on PEEK. *Materials*. 2020, 13 (2). DOI 10.3390/ma13020466.

[4] Giorgi Kakabadze. ANISOTROPIC BEHAVIOUR ANALYSIS OF 3D PRINTED STRUCTURES. *ReseachGate*. 2020. DOI 10.13140/RG.2.2.13651.40484.

[5] Protolabs Network. *What is FDM (fused deposition modeling) 3D printing?* https://www.hubs.com/knowledge-base/what-is-fdm-3d-printing/. Accessed on 11.19.2024.

[6] PrusaResearch. *3D tiskárna Original Prusa MK4S*. https://www.prusa3d.com/cs/produkt/3d-tiskarna-original-prusa-mk4s-5/. Accessed on 11.19.2024.

[7] stratasys . *Fortus 450mc Industrial FDM 3D Printer*. https://www.stratasys.com/en/3d-printers/printer-catalog/fdm-printers/fortus-450mc/.

[8] Jackson O'Connell. *3D Printing Infill: The Basics for Perfect Results*. 2024. https://all3dp.com/2/infill-3d-printing-what-it-means-and-how-to-use-it/.

[9] Protolabs Network. *What are supports in 3D printing? When and why do you need them?* https://www.hubs.com/knowledge-base/supports-3d-printing-technology-overview/. Accessed on 11.19.2024.

[10] Rachel Kelly. *Multi-material 3D Printer: Types & Printing Guide*. 2023. https://all3dp.com/2/multi-material-3d-printing-an-overview/.

[11] Thomas Sanladerer. *How bad is wet filament really?* 2021. https://toms3d.org/2021/11/23/how-bad-is-wet-filament-really/.

[12] Carolyn Schwaar. *The Complete Guide to 3D Printing Materials*. 2024. https://all3dp.com/1/3d-printing-materials-guide-3d-printer-material/.

[13] Goodwin UniversityGoodwin University. *What is CNC machining?* 2024. https://www.goodwin.edu/enews/what-is-cnc/.

[14] Ronan Ye. *G-code for CNC Machine: Commands & Uses*. 2024. https://www.3erp.com/blog/g-code-for-cnc/.

[15] Jack Minardi. *Mecode*. 2024. https://github.com/jminardi/mecode.

[16] RepRap. *Mecode*.
https://reprap.org/wiki/Mecode. Accessed on 11.20.2024.

[17] RepRap. *G-code*.
https://reprap.org/wiki/G-code. Accessed on 11.20.2024.

[18] Simplify3D. *3D Printing G-Code Tutorial*. 2019.
https://www.simplify3d.com/resources/articles/3d-printing-gcode-tutorial/.

[19] Lucas Carolo. *What is a 3D slicer? – simply explained*. 2024.
https://all3dp.com/2/what-is-a-3d-slicer-simply-explained/.

[20] *STL (STereoLithography) File Format Family*. 2019.
https://www.loc.gov/preservation/digital/formats/fdd/fdd000504.shtml.

[21] *The OBJ File Format*.
https://www.scratchapixel.com/lessons/3d-basic-rendering/obj-file-format/obj-file-format.html. Accessed on 11.20.2024.

[22] Joint Development Foundation. *.3MF Specification*. 2024.
https://3mf.io/spec/.

[23] PrusaResearch. *PrusaSlicer*. 2024.
https://github.com/prusa3d/PrusaSlicer/tree/63f8fda61d7222ca2e93aad8e47d32025e701ac5.

[24] Felix W Baumann, Martin Schuermann, Ulrich Odefey, and Markus Pfeil. From GCode to STL: Reconstruct Models from 3D Printing as a Service. *IOP Conference Series: Materials Science and Engineering*. 2017, 280 (1), 012033. DOI 10.1088/1757-899X/280/1/012033.

[25] Pretesh John, Venkateswara Rao Komma, and Skylab Paulas Bhore. Development of MATLAB code for tool path data extraction from the G code of the fused filament fabrication (FFF) parts. *Engineering Research Express*. 2023, 5 (2), 025018. DOI 10.1088/2631-8695/accc6f.

[26] Wojciech Sobieski, and Wojciech Kiński. Geometry extraction from GCODE files destined for 3D printers. *Technical Sciences*. 2020. DOI 10.31648/ts.5644.

[27] Thinkyhead. *Linear move*. 2025.
https://marlinfw.org/docs/gcode/G000-G001.

[28] *An introduction to ray tracing*. Oxford, England: Morgan Kaufmann, 1989.

[29] Matt Pharr, and Greg Humphreys. *Introduction*. Physically Based Rendering. Elsevier, 2010 . 1–52.

[30] *Intel OSPRay*.
https://www.ospray.org/. Accessed on 12.22.2024.

[31] *Intel OSPRay Documentation*.
https://www.ospray.org/documentation.html.

[32] Lighttransport. *GitHub - lighttransport/nanort: NanoRT, single header only modern ray tracing kernel*.
https://github.com/lighttransport/nanort/tree/release.

[33] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering: From Theory To Implementation*. 2023 .
https://pbr-book.org/3ed-2018/Primitives_and_Intersection_Acceleration/Bounding_Volume_Hierarchies.

[34] Steven R. Schill, John R. Jensen, George T. Raber, and Dwayne E. Porter. Temporal modeling of bidirectional Reflection Distribution Function (BRDF) in coastal vegetation. *GIScience & Remote Sensing*. 2004, 41 (2), 116–135. DOI 10.2747/1548-1603.41.2.116.

[35] Renfu Lu. *Light scattering technology for food property, quality and Safety Assessment*. CRC Press, 2017.

[36] Synopsys. *Mini-Diff v2 Users guide*. 2019.

[37] Cezner, Lukáš. *Měření povrchu BRDF s využitím LIGHTEC Minidiff V2 pro využití v programováni her*. 2023.

[38] Dimitrios Savva, and Jarod Guest. *golden gate hills*. 2025 . `https://polyhaven.com/a/golden_gate_hills`.

[39] Arvi VR. *Ambient occlusion: An extensive guide on its algorithms and use in VR*. 2025 . `https://vr.arvilab.com/blog/ambient-occlusion`.

[40] Prusa Research. *3D BENCHY by Prusa Research | Download free STL model | Printables.com*. 2025 . `https://www.printables.com/model/3161-3d-benchy`.

[41] xchgre . *Stanford dragon by xchgre | Download free STL model | Printables.com*. 2025 . `https://www.printables.com/model/696344-stanford-dragon/files`.

[42] Senk. *Cute Racoon by Senk | Download free STL model | Printables.com*. 2025 . `https://www.printables.com/model/1257280-cute-racoon/files`.

[43] Jakub Lattenberg. *Spiral Easter Egg by Jakub Lattenberg | Download free STL model | Printables.com*. 2025. `https://www.printables.com/model/1232334-spiral-easter-egg/files`.

[44] Oleksandr. *Swirl Pot (plant pots) by Oleksandr | Download free STL model | Printables.com*. 2025. `https://www.printables.com/model/1251142-swirl-pot-plant-pots/files`.

[45] Carter Hurdman. *nasa fabric by Carter Hurdman | Download free STL model | Printables.com*. 2023. `https://www.printables.com/model/516398-nasa-fabric/files`.

[46] Senk. *Easter Bunny by Senk | Download free STL model | Printables.com*. 2025. `https://www.printables.com/model/1255374-easter-bunny/files`.

[47] InnesPort. *Colored Prusa Rocket Engine for XL by InnesPort | Download free STL model | Printables.com*. 2024. `https://www.printables.com/model/1004320-colored-prusa-rocket-engine-for-xl/files`.

[48] Senk. *LowPoly Cat by Senk | Download free STL model | Printables.com*. 2025. `https://www.printables.com/model/1239993-lowpoly-cat/files`.

[49] Senk. *Baby Dragon by Senk | Download free STL model | Printables.com*. 2025. `https://www.printables.com/model/1253881-baby-dragon/files`.

[50] Senk. *Moai Face by Senk | Download free STL model | Printables.com*. 2025. `https://www.printables.com/model/1276658-moai-face/files`.

[51] Senk. *Owl by Senk | Download free STL model | Printables.com*. 2025. `https://www.printables.com/model/1252976-owl/files`.

Appendix A
Used files

Used G-code files
baby_dragon_big.gcode
benchy.gcode
big_owl.gcode
cute_raccoon_big.gcode
geometric_feline.gcode
maoiFace.gcode
nasa_fabric.gcode
rocket_engine_Prusa.gcode
spiralEgg.gcode
stanford-dragon.gcode
swirlVase.gcode

Generated BRDF textures
PETG_black.exr
PLA_blue.exr
PLA_green.exr
PLA_red.exr
PLA_white.exr
PLA_yellow.exr
SILK_pink.exr
PETG_orange.exr

Environment maps
golden_gate_hills_0075k.exr
golden_gate_hills_0125k.exr
studio_small_08_4k.exr

Appendix B
source folder structure

```
Brdfs - generated BRDF textures
EnviromentMaps - used environment maps
Gcode - gcode used to render 12 previews
   printed - gcode of the printed objects
Header - .h files of code created in this work
Source - .cpp files of code created in this work
```