RIJKSUNIVERSITEIT GRONINGEN

MASTER THESIS

MALORQA: A Machine Learning approach to Objective Reflection Quality Assessment

---

Author:
Rick DE JONGE

Supervisors:
J. Kosinka and G.J. Hettinga

March 25, 2021

RIJKSUNIVERSITEIT GRONINGEN

Faculty of Science and Engineering

Master Student Computing Science

Abstract

MALORQA: A Machine Learning approach to Objective Reflection Quality Assessment

by Rick DE JONGE

This thesis gives approaches to creating a machine learning model as an alternative to an analytical approach, which determine the smoothness of a surface visible in an image. It starts with describing what the analytical approach was set out to do, and how a machine learning model could improve on this. It then clarifies some important concepts, like what we mean by smoothness and how this can be visualized, and an existing algorithmic way of categorizing this smoothness. The creation of a dataset to identify this smoothness using machine learning is then explained, including the generation and rendering of the data. A first attempt at the machine learning program is defined, using machine learning on images of objects rendered using the same reflection lines as used in the Objective Rendering Quality Assessment (ORQA), of which the results are insufficient. A second attempt then builds on this by increasing the information in the input using normal mapping, and it is demonstrated that this improves both accuracy and speed of the learning process.

Contents

Abstract **iii**

1 Introduction **1**

2 Smoothness **3**

3 Data Acquisition **5**
- 3.1 Code Base 5
- 3.2 Rendering Techniques 5
  - 3.2.1 Reflection Lines 5
  - 3.2.2 Normal Map 6
- 3.3 Interpolation and Subdivision Techniques 6
  - 3.3.1 Triangular mesh 7
  - 3.3.2 Loop Subdivision 7
  - 3.3.3 Catmull-Clark Subdivision 9
  - 3.3.4 Point-Normal Triangles 10
  - 3.3.5 Phong Tessellation 11

4 Reflection Line Approach **13**
- 4.1 The Machine Learning Program 13
  - 4.1.1 Code Base 13
  - 4.1.2 Parameters 13
  - 4.1.3 Input Data 13
  - 4.1.4 Machine learning Model 14
- 4.2 Results 15

5 Color Map Approach **17**
- 5.1 Input Data 17
  - 5.1.1 Input Meshes 17
  - 5.1.2 Program Settings 17
- 5.2 Results 18

6 Discussion **21**

7 Conclusion **23**

A Model Diagrams **25**

B Input Data Samples, Reflection Lines Approach **29**

C Input Data Samples, Color Map Approach **31**

Bibliography **33**

Chapter 1

Introduction

Creating a smooth surface can be a monumental, but important task. Smooth surfaces can make creations more efficient (e.g. aircraft or machinery parts), they can be more visually pleasing (e.g. art sculptures), or make an accurate reconstruction of a real-world object. Since computer models of these creations are limited in the amount of data that can be obtained or stored, algorithms have been created to fill in the missing information in these surfaces based on the information present. These algorithms do however have slight differences in their interpretations of the surface, and need to be compared to find the complete surface most fitting the application it is used for. These differences can be in speed (for real time calculations) and accuracy (how realistic is this given the information we have). While these algorithms do have theoretical limitations, inaccuracies due to the limits of the algorithm do not necessarily need to be present on the model itself. Creating an algorithm that can accurately determine the smoothness of a surface based on a rendered model can help in making the optimal choice in these cases.

In order to determine the smoothness of a surface based on the rendering of a model, ORQA [6] was created to give an objective score to surfaces. A score is assigned by mapping the derivatives of the polylines extracted from reflection line renderings of the surface onto a linear score. An example of this can be seen in Figure 1.1, as can be found in [6].

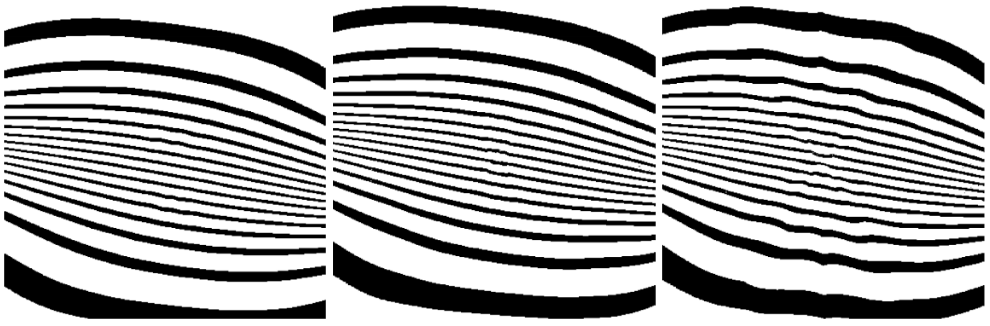
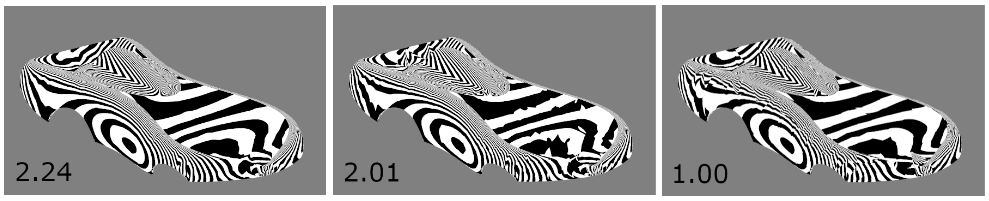

(a) Score 1.00 (b) Score 1.14 (c) Score 5.74

FIGURE 1.1: Samples from the ORQA dataset, the score indicating the smootness of the reflection lines rendered. Images from the original paper. [6]

While ORQA gives promising results, there are still extremes in the dataset the authors created that were classified poorly, e.g. in the cases of certain polyline lengths, which can be seen in Figure 1.2. The aim of this project is to change the method for scoring the surface. Where ORQA used analytical and mathematical methodologies to determine the results, a machine can learn aspects of the data that were not thought of before. A machine learning algorithm could train itself to recognize these now unknown patterns of the data.



(a) Score 2.24 (b) Score 2.01 (c) Score 1.00

FIGURE 1.2: Samples from the ORQA dataset, which were classified incorrectly. The correct order would be an increasing score from left to right. Images from the original paper. [6]

Where the original ORQA algorithm will assign a score to a surface, when training a new machine learning algorithm I will start with a smaller goal, in order to discover the future possibilities. The goal of this MALORQA project is to determine whether a machine learning algorithm for the ORQA project is possible. If this is the case, we can determine whether this machine learning approach can produce better results than the analytical approach.

Concretely, the goal of the implementation is to create a model that is able to classify images into a smoothness category. For this classification, I will use the C-notation, which I briefly explain in the next chapter. The created model will be judged based on the accuracy of the classification, or the percentage of correctly classified images the model was not trained on.

After explaining what is meant with smoothness in Chapter 2, I explain how the input data for training and validation of the models was created in Chapter 3. Chapter 4 and 5 outline two approaches to creating and training the models, one using images with reflection lines, the other using color maps. Following that, the results of the two approaches and further steps for the process are given in the discussion and conclusion, Chapters 6 and 7.

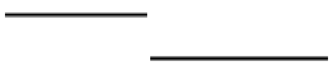
Chapter 2

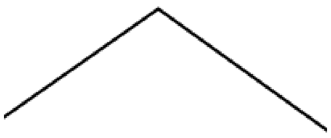
Smoothness

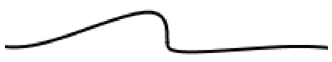
The aim of this project is to identify the smoothness of an object. In this paper, this is done by analysing a surface of the object in question. The smoothness of a surface can be classified using the C-notation. The lowest order in this system is the C^{-1} continuity, meaning a discontinuous surface. A surface with C^n continuity is connected, and its n-th derivative is also continuous. A C^0 continuous surface has a discontinuous first derivative, a C^1 continuity has a continuous first derivative, but a discontinuous second derivative, and so on. The highest class of smoothness is C^∞, of which all derivatives are continuous.

The images generated as the input dataset for the machine learning algorithm are in the categories C^0, C^1 and C^2, depending on the techniques used to generate the meshes. This process is described in Chapter 3.3. The smoothness in these images may not be equal across the whole image. We can distinguish here between local and global smoothness. Local smoothness is the smoothness measured at a single edge, face or vertex. Global smoothness is the smoothness of the whole normal field, determined by the lowest smoothness of each edge, face and vertex in the field.

Examples of the smoothness category of a curve, in two dimensions, can be found in Figure 2.1.


(a) C^{-1}, discontinuous data


(b) C^0, connected data


(c) C^1, continuous first derivative

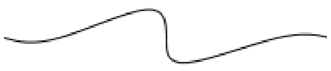

(d) C^2, continuous second derivative

FIGURE 2.1: Smoothness categories visualisation, in two dimensions

In three dimensions, we can visualise this using reflection lines, which are described later, to show the first derivative of the surface. In images like those in Figure 2.2, the derivative is taken in the vertical direction relative to the camera. Figure 2.2a displays a continuous surface, with a discontinuous normal. This can be seen along the red edge, where the reflection lines do not line up. Figure 2.2b, both the position and the angle of the reflection lines on the edge meet, signifying a continuous normal field.

In general, the smoothness category of the normal field is one lower than the smoothness of the surface itself. In some cases however, an object uses fake normals to create a smoother object. In Figure 2.3, the reflection lines do meet at the edge, but the angles are slightly misaligned. In this case the surface and fake normal field smoothness are classified as C^0.

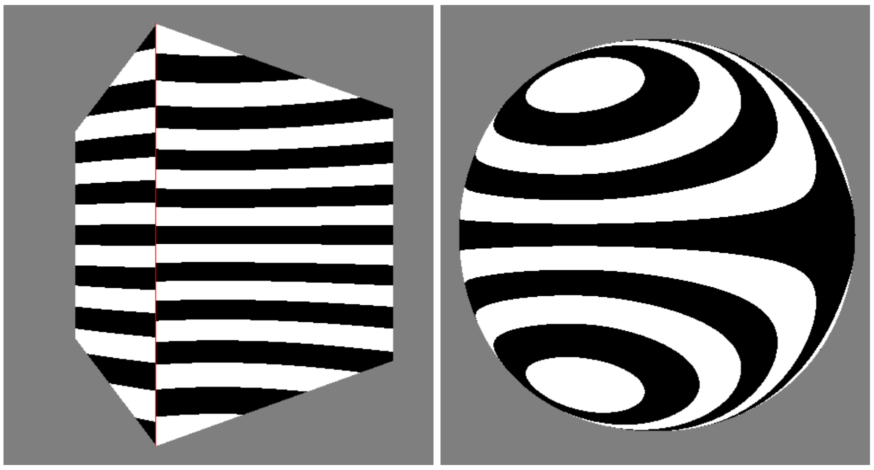

(a) C^0, connected data (b) C^1, continuous first derivative

FIGURE 2.2: Surface smoothness categories visualisation, in three dimensions

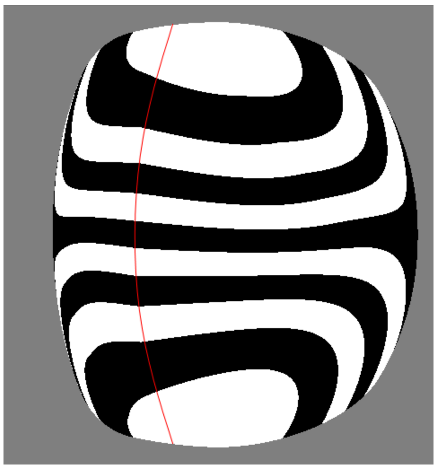


FIGURE 2.3: An object with both surface and fake normal smoothness C^0

Chapter 3

Data Acquisition

To create the datasets for the machine learning process, a rendering program is needed. The data that is fed to the machine learning algorithm is similar to the images created for the ORQA program, but many more images are needed for an accurate result. Concretely, I created a program that produces images based on an input mesh. This input mesh is refined using multiple different interpolation and subdivision techniques. A surface based on these meshes is then rendered, and this rendering is saved as an image file. This chapter starts by explaining the interpolation and subdivision techniques used, the specifics of the program, and the different ways to show the smoothness of the surface rendered.

3.1 Code Base

The generation of the dataset was created in the Qt framework, using the C++ language. The Qt framework can be used to easily create a user interface that helps with generating images quickly. It also adds multiple libraries that allow easier coding of specific and often used features. The C++ language is suited to make quick calculations while Qt allows for the use of higher level functions that allow for quick implementation and to write code that is still readable, and therefore less error-prone than other low-level programming languages.

3.2 Rendering Techniques

Since the mesh is simply a system of vertices and edges, they need to be shown using reflection lines or normal maps, as described below, to allow an algorithm looking at an image to determine what is in the image. There are different approaches to render the subdivided or interpolated mesh to an image. Using the interpolation techniques described below the program has access to the coordinates and normals that correspond to the surface made up by the mesh. Using this information, we can generate a color for each pixel in the image. The program can render the mesh using the following maps from this information to a color.

3.2.1 Reflection Lines

This technique aims to construct black and white bands on the surface to visualise the normal field. For each point on the surface, the normal is given. The angle between this normal and the viewpoint is then calculated. A simple step function then determines whether the color of the surface should be white or black.

This step function assigns half of the values to black, and the other half to white. A flat surface would have unchanging normals, but their angle to the viewpoint will be changing at a constant rate, resulting in a surface with equally spaced bands. The changing normal will result in bands that are thinner on a convex surface and wider on a concave surface. Using this information, we can see irregularities in the normal field of the object, in parts of the surface where the bands are pinched. An example of this can be found in Figure 3.1.

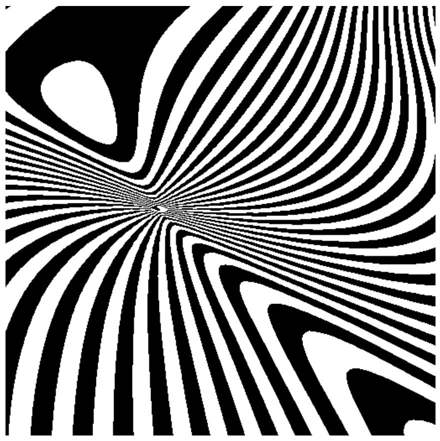


FIGURE 3.1: An area where the reflection lines are close together with a very small width, a pinch indicating an irregularity in the normal field.

3.2.2 Normal Map

Although the reflection lines are a useful tool for a human observer, they do lose some information. Mapping each normal coordinate to a color does not suffer this problem, but it makes it harder for a human to spot irregularities in the normal field. The machine learning algorithms are improved by supplying them with more information, and are excellent at spotting patterns in a normal mapped surface. A normal is also easily mapped to a color. We can scale the angle between the normal and the viewpoint from 0 to 255 and create a grayscale image, or take a normal with x, y and z values from -1 to 1 and map those linearly to red, green and blue values between 0 and 255. Since in this project we are looking for the smoothness of an object, or the change in the normals of the surface of this object, they grayscale normal map gives enough information to continue. Using the normal of the surface makes sure that only information about the surface's curvature is used by the model. Using shading, like reflection lines or Phong shading [7], adds unimportant information, e.g. depth to the camera or angle to the light source, which may be detrimental to the resulting model.

3.3 Interpolation and Subdivision Techniques

To obtain dense meshes with different types of smoothness, I started with simple meshes consisting of only a few vertices, edges and faces. These meshes are then either left simple, e.g. flatshading, subdivided before rendering, e.g. using Loop [5] and Catmull-Clark subdivision [3], or interpolated while rendering, e.g. Point-Normal Triangles [9] and Phong Tessellation. [2] The main difference between the subdivision and interpolation techniques mentioned in this thesis are their place in the pipeline. The subdivision of meshes is done on the CPU, which is slower but can be stored and edited after the fact. The interpolation is done using tessellation, on the GPU, which is faster because of the large amount of parallelisation in the GPU. The downside is that this information can only be used in this rendering stage of the pipeline.

The smoothness of the resulting surfaces belonging to these meshes depends on the technique used. The goal of interpolation and subdivision is to supply the rendering process with information about the position and normal on each point in the surface that is defined by the mesh.

3.3.1 Triangular mesh

The rendering of the mesh is done in two ways, either uniform over each face in the mesh, or calculated per pixel based on the face and its vertices. In the first case, the surface of the object is rendered using the positions present in the mesh. The normals of each point on the surface is the average of the normals of the vertices of the face it is on. This rendering technique is called flat shading. The continuity of the reflection lines on the surface of a triangular mesh rendered using flat shading is C^{-1} at the vertices and edges, where the normal field is discontinuous. The surface itself is connected, and therefore smoothness C^0. This is visible in Figure 3.2, where the reflection lines on the vertical diagonal edges are not continuous.

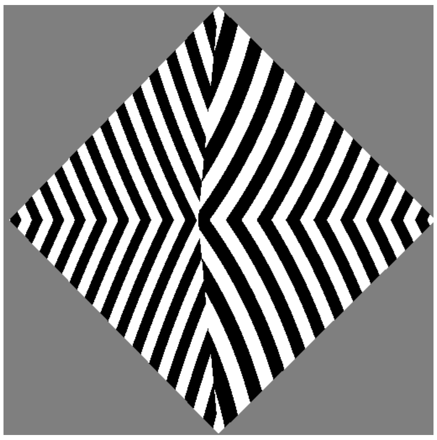


FIGURE 3.2: Rendering an octahedron with flat shading and reflection lines.

3.3.2 Loop Subdivision

Loop subdivision [5] is a technique to subdivide a triangle mesh into smaller triangles, based on quartic box splines [1]. The resulting mesh of a single subdivision step will have four triangles in the place of each triangle in the original mesh, where the position of the vertices of these new triangles are a weighted combination of the previous triangle. This process can be repeated to create a denser mesh, in the limit the result is called a limit surface.

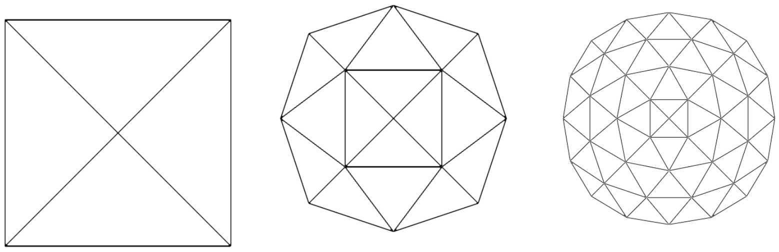
A new vertex is created for each edge in the mesh. The position of this new vertex is a weighted sum of the vertices of the triangles adjacent to the edge. The two vertices at the ends of the edge are weighted three times more then the other vertices of the triangles.

The new position of an existing vertex is also a weighted sum, over all the vertices that share an edge with the current vertex. Loop's original weights β were calculated as

$$\frac{1}{n}\left(\frac{5}{8}-(\frac{3}{8}+\frac{1}{4}\cos\frac{2\pi}{n})^2\right)$$

with n being the valency of the vertex. The weight of the original vertex is $(1-n)$ times the weight of the other vertices. In this project Warren's [10] implementation is used instead, where the weights are $\frac{16}{3}$ when the vertex has valency 3, or $\frac{8n}{3}$ when the valency is higher. These weights were chosen because of the lower computation time.

The results of this subdivision on a simple mesh can be found in Figure 3.3.



(a) No subdivision (b) The mesh subdivided once (c) The mesh subdivided twice

FIGURE 3.3: A simple mesh, subdivided using Loop's algorithm

The smoothness of a limit surface created using Loop's technique is C^2 everywhere, except at extraordinary vertices (EVs), vertices without valency six, where it is C^1. An example of this can be seen in Figure 3.4, where the reflection lines around the front vertex of the octahedron, an EV with valency four, are pinched, but still continuous.

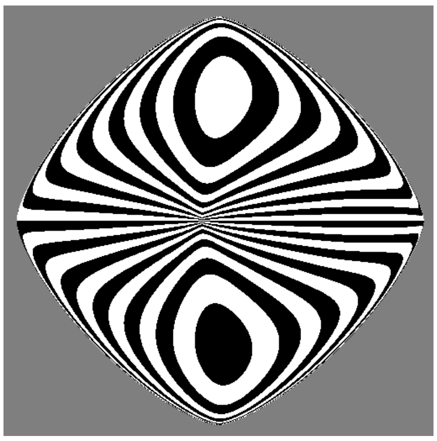


FIGURE 3.4: Rendering an octahedron with reflection lines, after the mesh has been subdivided using Loop subdivision nine times.

3.3.3 Catmull-Clark Subdivision

Catmull-Clark subdivision [3] is a technique to subdivide meshes. The algorithm works best on quadrilateral meshes, where it creates four new faces for each face in the previous mesh. Other shapes, like triangles, are subdivided into quads in their first subdivision. In the limit, this technique produces regular bicubic B-splines surfaces. Each iteration of the Catmull-Clark algorithm goes through the following steps, which are applied to the mesh to create a denser mesh:

1. Create new face points: A face point is a vertex in the subdivided mesh that is placed at the average location of all vertices of a face in the original mesh.
2. Create new edge points: Edge points are vertices in the subdivided surface that are created for every edge in the original mesh at the average location of 4 points: the end points of the original edge and the face points created in the previous step for the 2 faces adjacent to that edge.
3. Create new vertex points: For each vertex in the original surface a vertex is added to the subdivided mesh. The location of this vertex is determined by the average $\frac{Q}{n} + \frac{2R}{n} + \frac{S(n-3)}{n}$, where Q is the average of the face points of adjacent faces, R is the average of the edge points of all outgoing edges, S is the location of the original vertex and n is the valency of that vertex.
4. Create new edges: Edges are created by connecting each face point to all its adjacent edge points and each edge point to its adjacent vertex points.
5. Create new faces: A new face in the subdivided surface is defined as the linear space enclosed by edges created in the previous step.

Applying this process to a regular cube results in a mesh as in Figure 3.5.

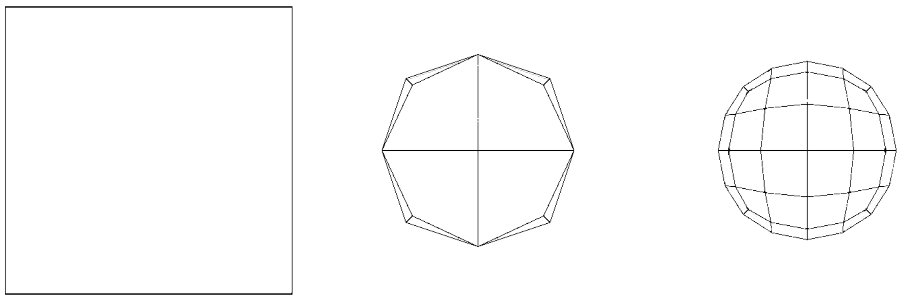

(a) No subdivision (b) The mesh subdivided once (c) The mesh subdivided twice

FIGURE 3.5: A simple mesh, subdivided using the Catmull-Clark algorithm

The smoothness of a Catmull-Clark subdivided limit surface is similar to one using Loop subdivision: C^2 everywhere, except at EVs, vertices without valency four, where it is C^1. Figure 3.6 is an example of an octahedron, rendered using reflection lines.

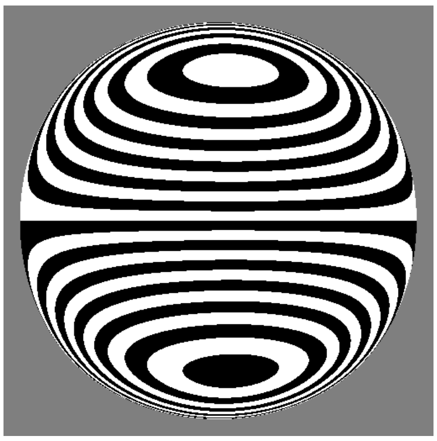


FIGURE 3.6: Rendering an octahedron with reflection lines, after the mesh has been subdivided using Catmull-Clark subdivision seven times.

3.3.4 Point-Normal Triangles

The first interpolation technique used in this project is the Point Normal (PN) triangle method [9], which can be used on triangle meshes. The new coordinates of the points on the surface are interpolated cubically, whereas the normals are interpolated quadratically, calculated independently for each triangle in the mesh.

The smoothness of the surface on a single face interpolated using PN triangles is C^∞, but on edges between faces this is only C^0. An example of this can be seen in Figure 3.7, where the sharp angles of the reflection lines on the edges can be seen as the indication of a connected but suddenly changing normal field.

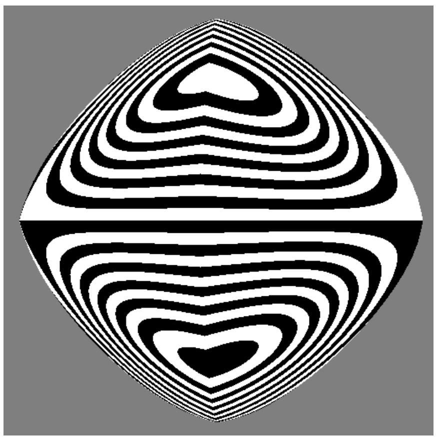


FIGURE 3.7: Rendering an octahedron with reflection lines, after the mesh has been interpolated using PN triangles, using a tessellation level of 50.

3.3.5 Phong Tessellation

Phong Tessellation [2] is based on the same principles that make up Phong shading [7], namely to interpolate the normals of the triangle vertices to calculate the normal of a point. This process consists of three steps:

- Compute the linear tessellation of the given triangle.
- Project the resulting points on planes tangent to the triangle's normals.
- Compute the barycentric interpolation of the projected vertices.
- Interpolate between flat tessellation and Phong tessellation

The final step here is weighted using a variable α, which is suggested to be around $\frac{3}{4}$. In this project, $\frac{4}{5}$ resulted in optimal renderings and will be constant from here on. The above steps result in an estimated position for the selected vertex, after which a weighted average of the old and estimated vertices is taken. Like PN Triangle surfaces, the global smoothness of the surface is C^0, as can be seen in Figure 3.8, with the same reasoning as for PN triangles.

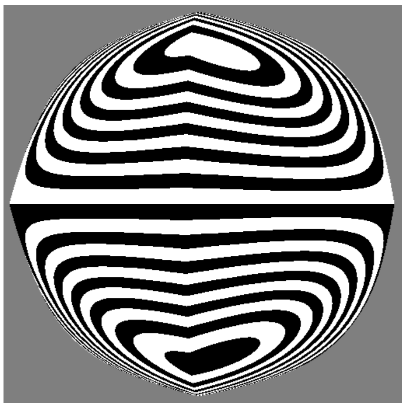


FIGURE 3.8: Rendering an octahedron with reflection lines, after the mesh has been interpolated using Phong tessellation, using a tessellation level of 50.

Chapter 4

Reflection Line Approach

Machine learning experiments were done using images of subdivided and interpolated meshes, rendered using reflection lines, similar to the ORQA project. This attempt aimed to eventually reach output comparable to the ORQA project, given the same input as that program, where an image would be given a score based on the smoothness of the rendered object. This process was divided into multiple simple steps, in order to gradually build to this goal.

4.1 The Machine Learning Program

4.1.1 Code Base

The program was coded in the Python programming language, using the Tensorflow library specifically designed for the creation of machine learning programs. This library is well-known and used in for academic purposes already, and is currently on its second major version. It is well documented too, which allowed me to select the specific implementations which were appropriate for the project.

Python itself is easy to learn, and it is easy to learn the functions of new libraries. Using Pycharm and the build-in debugger and syntax highlighting allows for quick development and a relatively clean codebase.

4.1.2 Parameters

Building on the structure provided by Tensorflow, the main parameters to tune are the machine learning model and the input to the program. The Tensorflow library supplies functions to load large amounts of data and supply this data to the model efficiently, and to save the resulting model parameters. Attempts were made to simplify the process, e.g. by simplifying the image format, but using the provided dataset structure was more efficient without spending too much time modifying code.

4.1.3 Input Data

To start with a simple dataset, simple meshes were used to render the input data. These simple meshes include a cube, icosahedron and octahedron. These meshes were then subdivided or interpolated using the described methods, and rendered using reflection lines. To ensure the program had uniform input, the objects were centered to the frame and close enough to show no background. This was done because the model would learn to identify contours instead of identifying the surface smoothness using patterns in the reflection lines. A sample of this dataset can be found in Appendix B. To artificially create a larger dataset, the images were rotated

Method	Isolines	Subdivision steps	Tessellation level	Global surface smoothness
Triangular mesh	60-100			C^0
Loop	30-60	8-9		C^1†
Catmull-Clark	30-60	6-7		C^1†
Point-normal Triangles	30-60		50	C^0
Phong Tessellation	30-60		50	C^0

TABLE 4.1: Program settings
† The smoothness of the surface is C^2 for any initial mesh without EVs.

and the zoom level was changed slightly in multiple copies of the original image. The exact settings of the generation of the datapoints can be found in Table 4.1.

The full dataset was created using a few different basic meshes, namely an Icosahedron, an octahedron, a cube. Along with these meshes, variations on these were created by perturbing the position of certain vertices to create different angles between the faces of the mesh. These meshes were then subdivided or interpolated, for each of these techniques, a number of images were taken of the mesh, where the mesh was rotated randomly along all three axes. For the largest set of 20000 images, this number was 500 images per technique per mesh.

To train the model, two sets of images are needed, namely a training set and a validation set. The Tensorflow dataset structure that loads the images allows us to directly split the input data into two, with a parameter of the ratio between the two sets. In this experiment, 20% of the data is randomly chosen to be the validation set, and the other 80% as the training set.

4.1.4 Machine learning Model

The main variable of this part of the program is the model itself. This model consists of several layers and the metrics for improvement. While the library has many different kinds of layers available to slot in here, these are the most useful for identifying slopes in a two-dimensional image:

- Rescaling
  This is a processing option that rescales an image with color-values between 0 and 255 to one with values between 0 and 1, since the algorithms are optimized with this scale in mind.

- 2D convolution
  A convolution tries to discern patterns in smaller areas of the image. This filter can look at a small area, say 3×3 pixels, for example to learn extreme changes in areas or look at larger areas, say 11×11 pixels, to learn a particular curvature in the texture.

- 2D MaxPooling
  This filter is used to decrease the size of the data. This uses the values of the pixels in an area and results in a single value for this area. This result is the maximum value of the input for MaxPooling.

- Flatten
  At some point, it does not make sense to represent the data as a 2D object anymore. At this point, the data is flattened, turning it into a simpler 1D tensor.

- Dense
  These are the simplest layers for a multi-layer perceptron, a 1D tensor of nodes. Once patterns have been established in the previous layers, this layer's nodes represent certain combinations of the previous nodes, weighted by a variable for each node. The final layer of the whole model will be a tensor consisting of one node for each category, of which the highest valued node will be returned as the category assigned to the input image.

The metrics of the model we can change are the following:

- The value metric
  This is the metric we want to optimize. In this project, the aim is to be able to accurately predict the smoothness category of a surface. The *accuracy* parameter here means that the model will optimize for the most relative amount of successfully classified images in each batch.

- The loss function
  The loss function used here is the cross-entropy [8] between the true labels and the predicted labels.

- The optimizer method
  The optimization technique can be specified to control how the model will optimize the value metric. The difference between loss functions was not significant during the training in this project, so the Adam optimizer was used, which is a stochastic descent algorithm [4].

The starting model was based on the basic Tensorflow model, which could identify different animals. This model started with three convolution layers, followed by two dense layers, as specified in Figure A.1. From there the model was changed to include more or less convolution or dense layers, or to increase or decrease the sizes of these layers, as can be seen in Appendix A.

4.2 Results

The results of these tests can be found in Figure 4.1 As is clearly visible here, the machine easily learns the properties of the test dataset, and accurately classifies those images. The validation sets however are classified only a little better than a random choice. The results for this dataset are still consistent.

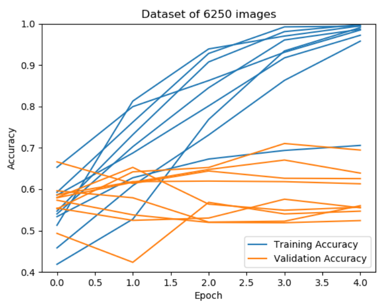


FIGURE 4.1: Training a model on images rendered using reflection lines.

Increasing the size of the dataset could improve the result, as can be seen in Figure 4.2. Here, one attempt to training the model resulted in a validation accuracy of over 80%. This increase in size does however significantly increase the time it takes to complete the learning process. This time consuming nature combined with the widely varying results make this approach not good enough to eventually classify more complex images that would need to be trained on an even larger dataset. Some results of the training lead to models that are as good as a random guess, as can be seen in a single result in Figure 4.2. While there is still some variation in the results, in this variation the model, only a single small dense layer was used. This suggests that too little of either convolution or dense layers, combined with a model that ends up learning the wrong aspects of the images can lead to outliers that are not useful for further refinement.

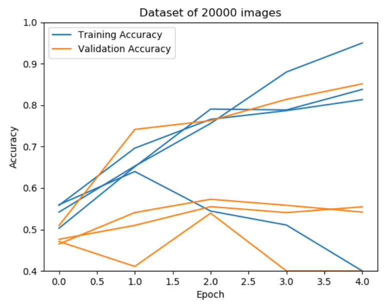


FIGURE 4.2: Increasing the size of the dataset to 20000 images.

Through this test which increases the size of the dataset, we can see that increasing the amount of data could improve the results of the learning process. But since these results are not very consistent, creating other ways to supply the machine with more information to learn from would be the next step in improving the results. Starting with a better controlled dataset could also lead to the model learning the right information, and rely less on the size of the dataset alone.

Chapter 5

Color Map Approach

Since the first attempt at a machine learning model to categorize smoothness of an object did not give useful results, some core ideas about the project needed to be changed. Firstly, the input data immediately tried to be comparable to the input that ORQA was created to analyze. A simpler start would reach a higher validation accuracy more quickly. Secondly, the input was still filtered to allow for a human opinion on the image, while a machine can easily parse more information.

5.1 Input Data

The goal for this second attempt is to see the effects of feeding images into the machine learning program which are more information dense. This is done by using normal mapping instead of rendering the objects using reflection lines. This means that more information can be learned than just the edges of the reflection lines. Modelling the more detailed slopes of the normals of the rendered object can result in a more accurate estimate of the smoothness of this normal field.

5.1.1 Input Meshes

The previous meshes used to render the images of the first attempt were simple meshes, but this approach may still create too many different local smoothness areas in the image. Some of the used subdivision and interpolation techniques have a different smoothness category at different valency vertices, so the images taken will focus on these differences. The images will focus on either a vertex or an edge, with valencies ranging from three to six. This means that only a single vertex or edge of the original mesh (before any subdivision or interpolation is used) is visible in the image.

To generate enough data for a model to train on, the rendered vertex or edge was not centered to the image, and the resulting images were rotated around the center. Different zoom levels were also used to create differences in the dataset. For the dataset with 8000 images centered on vertices, 400 images were taken for each technique and mesh combination. As with the previous approach, 20% of these images were randomly selected as validation data, while 80% were training data.

Examples of this generated dataset can be found in Appendix C.

5.1.2 Program Settings

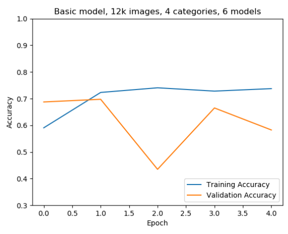
The change in the rendered input meshes results in a global smoothness determined by both the subdivision or interpolation technique and the valency of the edge or vertex displayed. The label smoothness categories used to train the model can be found in Table 5.1.

Mesh vertex valency	3	4	5	6
Triangular mesh	C^0	C^0	C^0	C^0
Loop	C^1	C^1	C^1	C^2
Catmull-Clark	C^1	C^2	C^1	C^1
PN triangles	C^0	C^0	C^0	C^0
Phong Tessellation	C^0	C^0	C^0	C^0

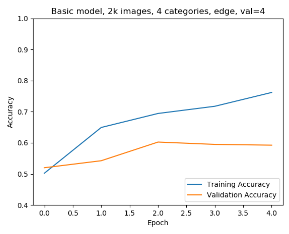
TABLE 5.1: The smoothness of the rendered surface using the technique displayed left, created from a mesh with vertices of valency as displayed above.

5.2 Results

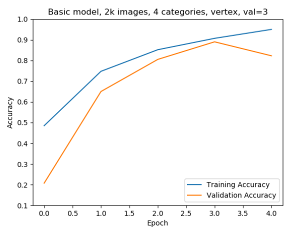
The first tests were to establish the usefulness of training on edges or vertices. The Figures 5.1 show that, the vertices are more useful to the model to classify the smoothness of the shown surface. Training on the edges show a limit in accuracy, 5.1a plateaus almost immediately, while the accuracy in 5.1b is simply much lower than the others. Most notable on the positive side are Figure 5.1c, where the validation accuracy approaches 90%, and 5.1f, in which the validation accuracy is higher than the training accuracy. This resulted in the choice to let the model train on vertices exclusively. For now, this could achieve the best accuracy. In future tests with a larger surfaces, like the surfaces in the ORQA test images, multiple vertices will likely be visible in the image, so the images that cannot be classified using results following from this conclusion will be minimal.



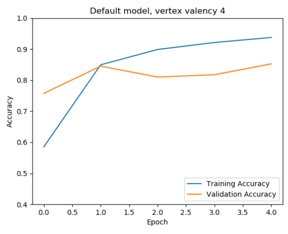
(a) Edge, mesh with vertex valency 3



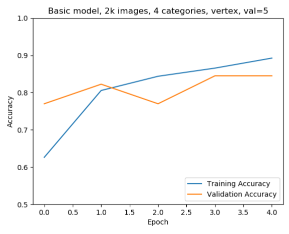
(b) Edge, mesh with vertex valency 4



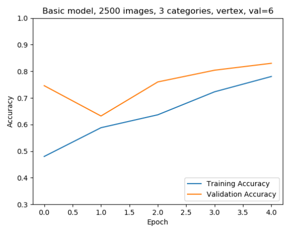
(c) Vertex, valency 3



(d) Vertex, valency 4



(e) Vertex, valency 5

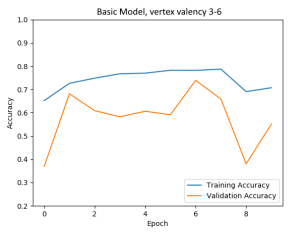


(f) Vertex, valency 6

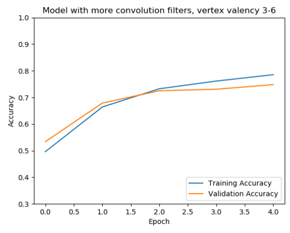
FIGURE 5.1: The basic model (A.1) trained on a single edge or vertex.

After deciding on the content of the dataset, this dataset was used to train all the models as they were in the previous attempt. Some of these results are shown in Figure 5.2. Out of these results, of which the models were chosen to vary in different ways compared to the default model, the model that increased the amount of convolution layers achieved the highest accuracy, as seen in Figure 5.2b.

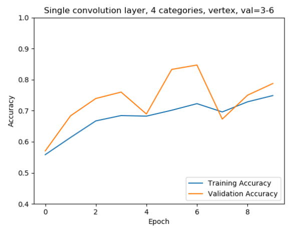
From this most prominent model, variations were made to increase the accuracy further. Some of the attempts can be seen in Figure 5.3a, while best results are shown



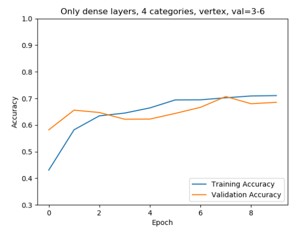
(a) Model A.1.



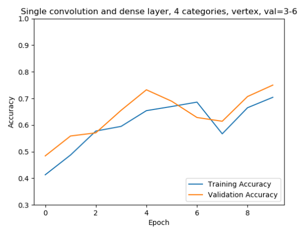
(b) Model A.2c, with more convolution filters.



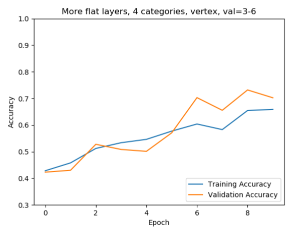
(c) Model A.2b, modified with a single large dense layer.



(d) Model A.3b, with only dense layers.



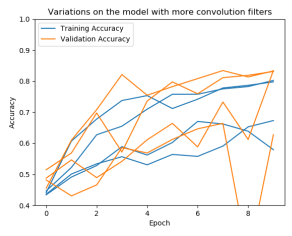
(e) Model A.3a, with only a single convolution and dense layer.



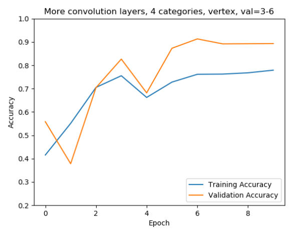
(f) Model A.3c, with less convolution layers and a larger dense layer.

FIGURE 5.2: Different models trained on a dataset focused on vertices of different valencies.

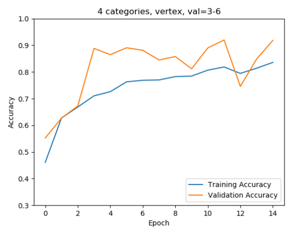
in Figures 5.3b and 5.3c, where the validation accuracy reaches and stays around 90%. In these figures, the validation accuracy is higher than the training accuracy. This means that the model learned the characteristics of the categories, instead of the details of the images in the training set. While this is unusual in machine learning, this does mean that the model is not overfitting, in which a model learns details of the training data which are not representative of the data as a whole. Both validation accuracies do plateau in fairly quickly in the learning cycle, which means that increasing the time the model learns the data will likely result in overfitting if the learning continues.



(a) Multiple variations on Model A.2c.



(b) Model A.4a.



(c) Model A.4b.

FIGURE 5.3: Slight adaptations on Model A.2c, highlighting the two best performing adaptations.

Chapter 6

Discussion

The best results of the previous experiments classify the smoothness present in an image 90% of the time. This means that so far this method is not as reliable as ORQA, which is more nuanced in it's conclusion, giving a score instead of a defined category. More importantly, when using ORQA, there are some known patterns in the images that can cause less accurate results. With the machine learning approach, it is yet unknown which patterns in the input give inaccurate results. Getting to the optimal solution is hard in itself. So far, some machine learning models have had large differences in the accuracy when training on the same architecture multiple times, so training the machine learning model likely has to be executed more than once to reach a good result.

While overfitting is present in the models, it is hard to determine partly through training whether this state has been reached. Multiple machine learning models have fluctuated heavily in their evaluation of the validation data, sometimes over 20% between epochs. However, in the best results the validation accuracy for each training epoch seems to peek or steadily come to a plateau.

Despite these problems, this technique still has clear pathways that can be explored in order to increase this accuracy, compared to ORQA which is mostly working as intended. The main improvement is the creation of a dataset which is much larger, and includes more realistic meshes, which were left out here due to time constraints. A model training on a larger and more varied dataset, while taking much longer to train, will be more stable and reach a better validation accuracy. While various model architectures were investigated, there are numerous small variations left, and more than are vastly different.

The experiments have also not included other Tensorflow model layers other than convolution, maxpooling and dense layers. Other layers like cropping or averaging layers may be useful in selecting the right areas for the model to train on or to disregard unimportant details present in the image. Other tools from and beyond the Tensorflow library could also improve the results presented here.

Chapter 7

Conclusion

Creating a machine learning model is a long and detailed process. The dataset needs to be generated, the machine created and the variables tuned. In this project, the way towards competing with the original ORQA project using machine learning is still incomplete. Step by step however, progress is made. Starting with creating a dataset that allows a model to train itself to recognize the areas around a vertex to indicate its smoothness. While more complicated data needs more time to train, this project has proven that such a machine learning model can achieve an acceptable accuracy on simple renderings. The specific architecture used for this is still to be tuned further, but this should be done weighing the time it takes to tune versus the accuracy it needs to achieve.

All in all, the goal of creating a machine learning model that is able to distinguish between different smoothness categories has been achieved, if only on simple rendered surfaces. This project gives a foundation to increase the accuracy and complexity of such a machine learning model, and concludes that a machine learning approach to this analytical problem of determining smoothness is possible, but for now the alternative is still better.

Appendix A

Model Diagrams

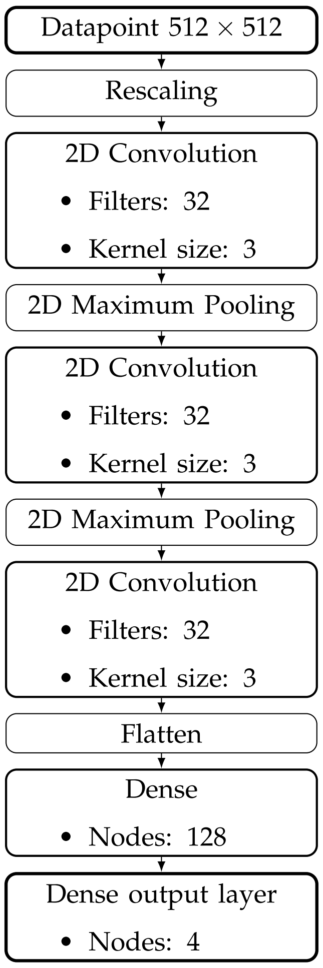
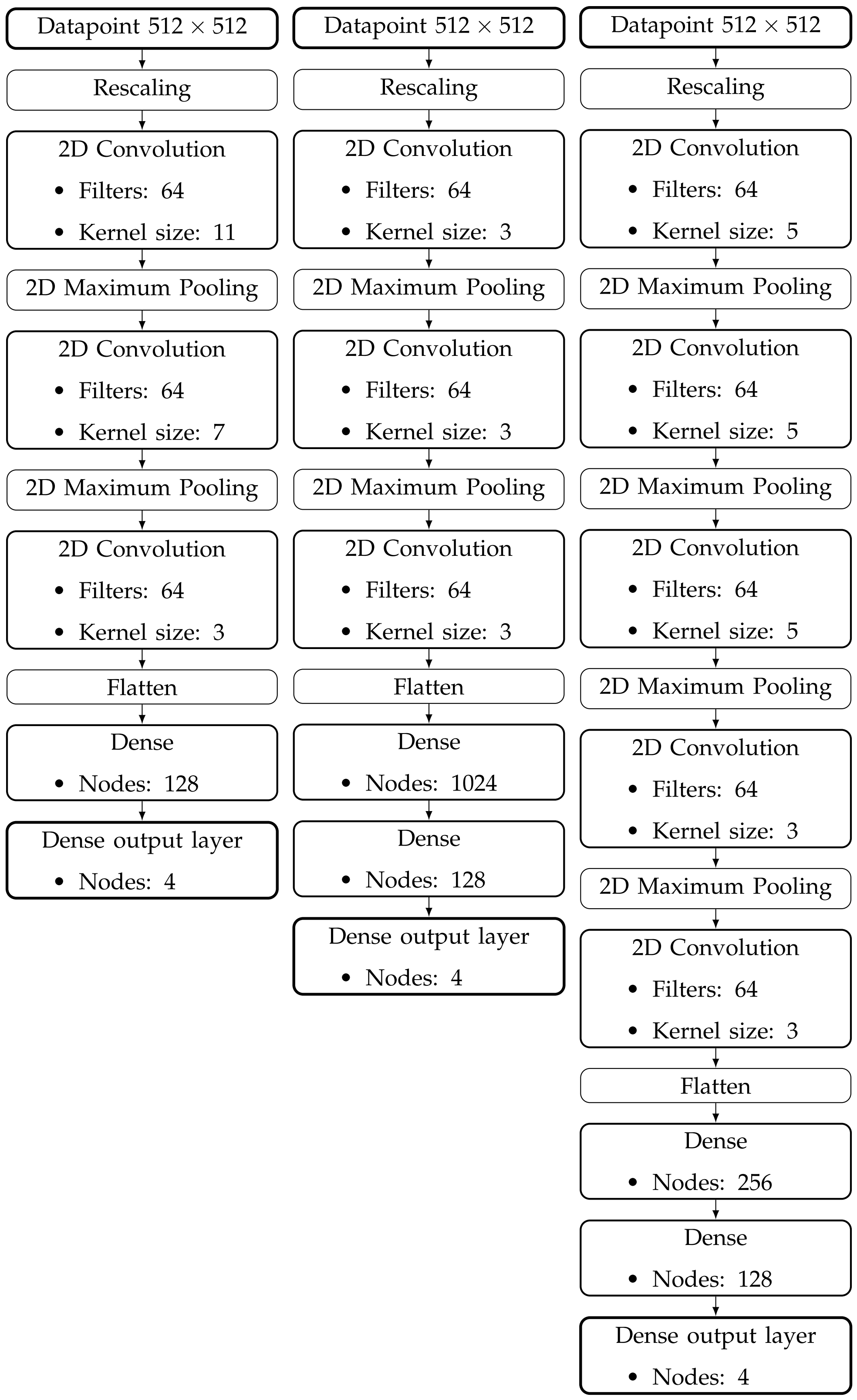


FIGURE A.1: The starting model of three convolution layers and two dense layers.

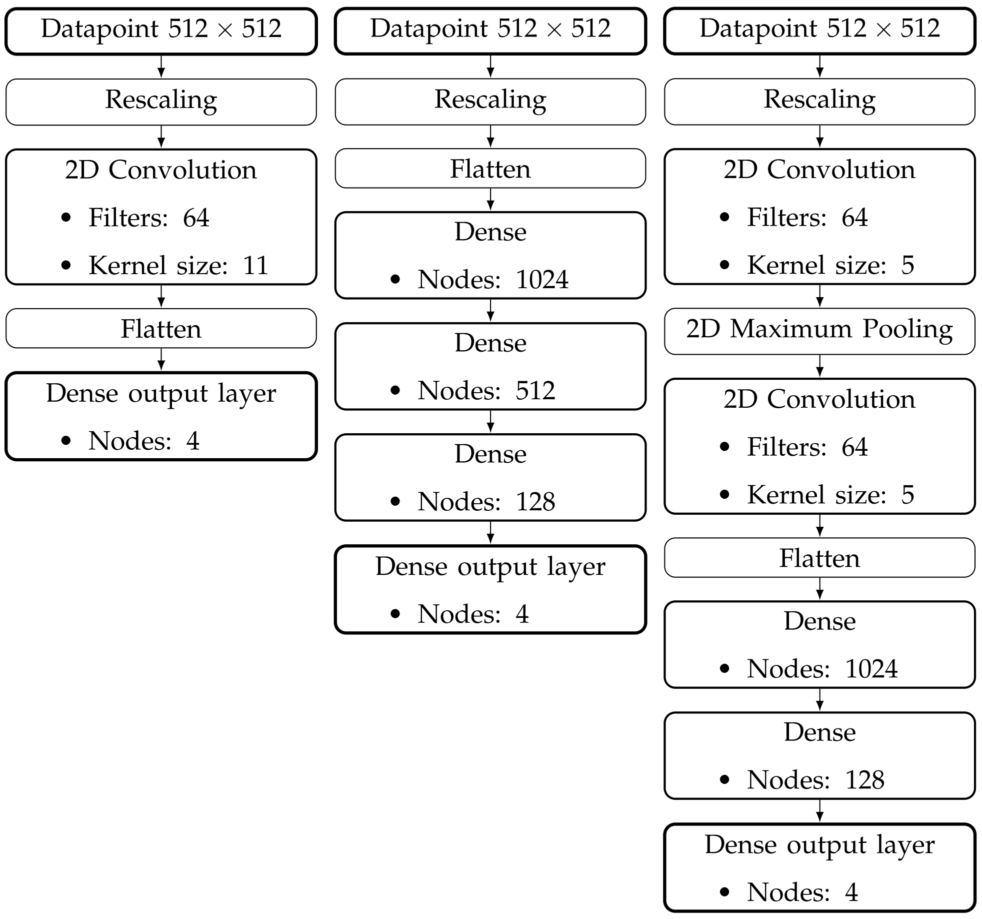


(a) A model that increases the filter size of the convolution layers, compared to the starting model of Figure A.1.

(b) A model that increases the amount of nodes in the dense layers, compared to the starting model of Figure A.1.

(c) A model with more convolution and dense layers, compared to the starting model of Figure A.1.

FIGURE A.2: Variations on the model shown in Figure A.1.

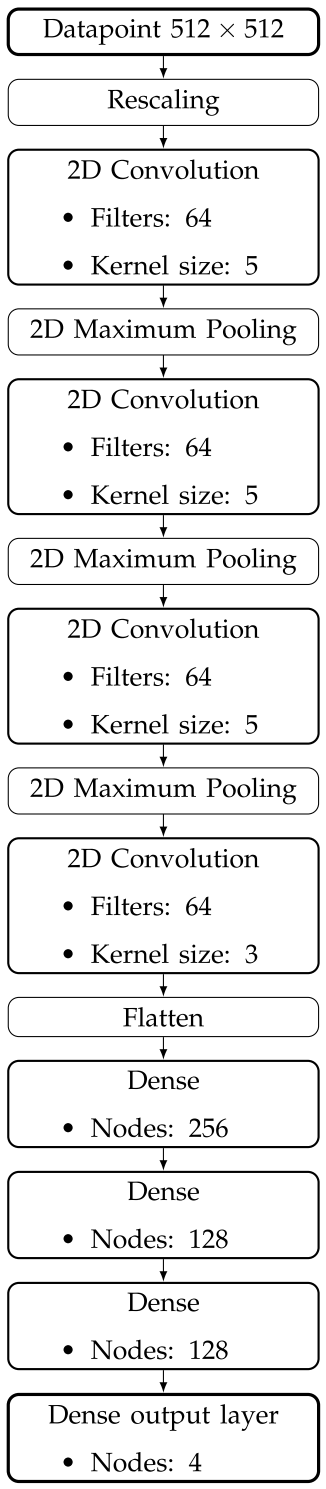


(a) A simple model with a single convolution and dense layer.

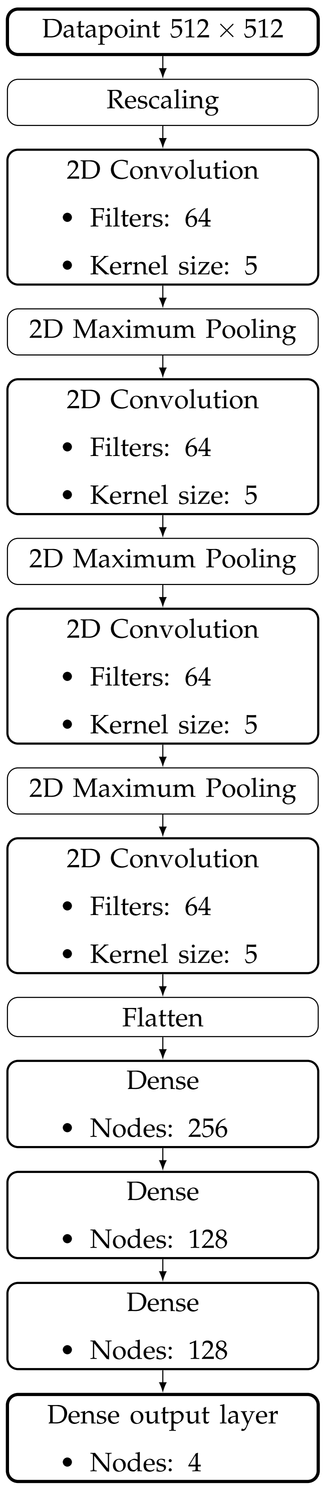
(b) A model with only dense layers of various sizes.

(c) A model with two convolution and dense layers, which are larger compared to the starting model of Figure A.1.

FIGURE A.3: Variations on the model shown in Figure A.1.



(a) A model with 5 convolution layers and 3 dense layers.



(b) A model with only dense layers of various sizes.

FIGURE A.4: Variations on the model shown in Figure A.2c.

Appendix B

Input Data Samples, Reflection Lines Approach

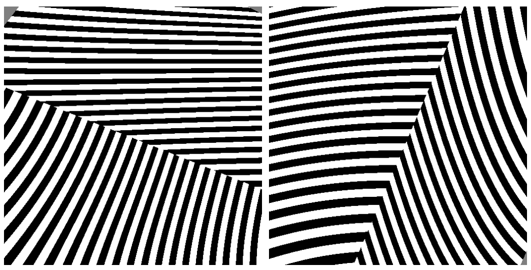


FIGURE B.1: Samples of the input data, from rendering a cube. Rendered using flat shading. These images were labeled as smoothness C^0.

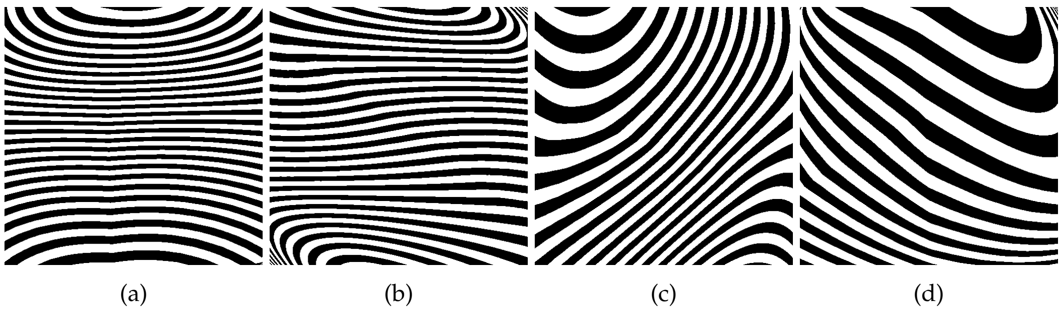


FIGURE B.2: Samples of the input data, from rendering a cube. Rendered using PN triangles (a, b) and Phong tessellation (c, d). These images were labeled as smoothness C^0.

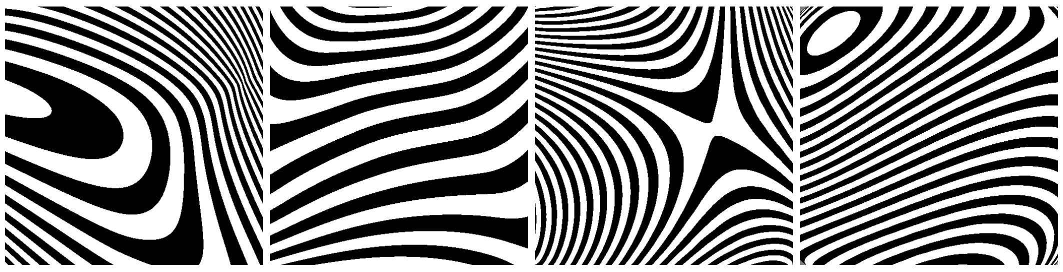


FIGURE B.3: Samples of the input data, from rendering a cube. Rendered using Loop (a, b) and Catmull-Clark subdivision (c, d), labeled as smoothness C^1.

Appendix C

Input Data Samples, Color Map Approach

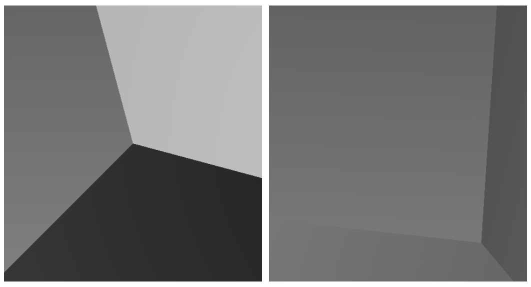


FIGURE C.1: Samples of the input data, around a vertex with valency 3. Rendered using flat shading, labeled as smoothness C^0.

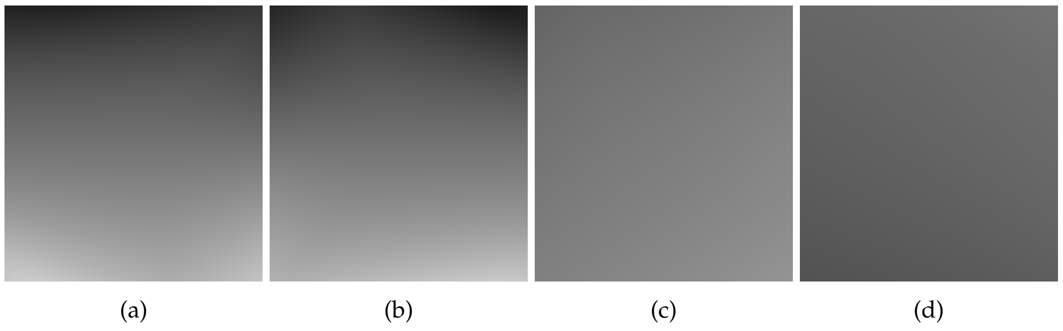


FIGURE C.2: Samples of the input data, around a vertex with valency 3. Rendered using PN triangles (a, b) and Phong tessellation (c, d), labeled as smoothness C^0.

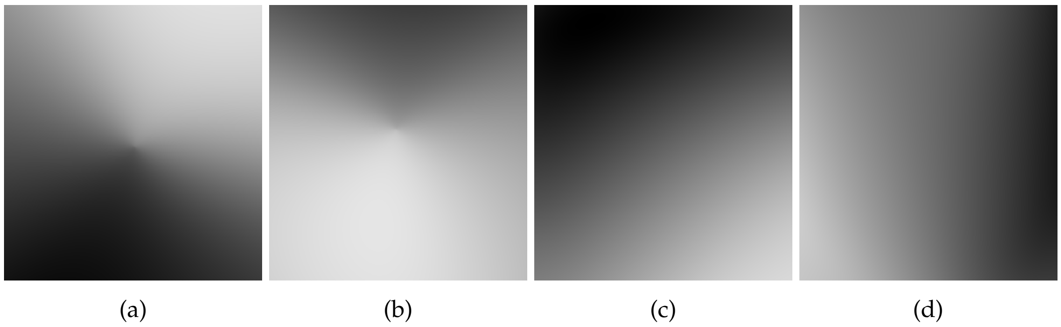


FIGURE C.3: Samples of the input data, around a vertex with valency 3. Rendered using Loop (a, b), and Catmull-Clark subdivision (c, d), labeled as smoothness category C^1.

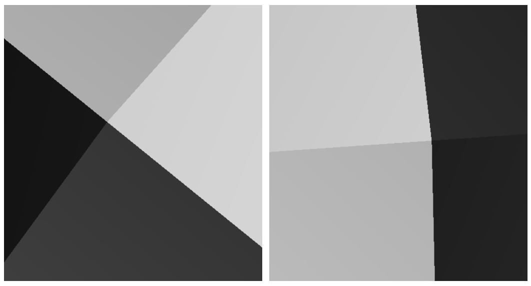


FIGURE C.4: Samples of the input data, around a vertex with valency 4. Rendered using flat shading, labeled as smoothness C^0.

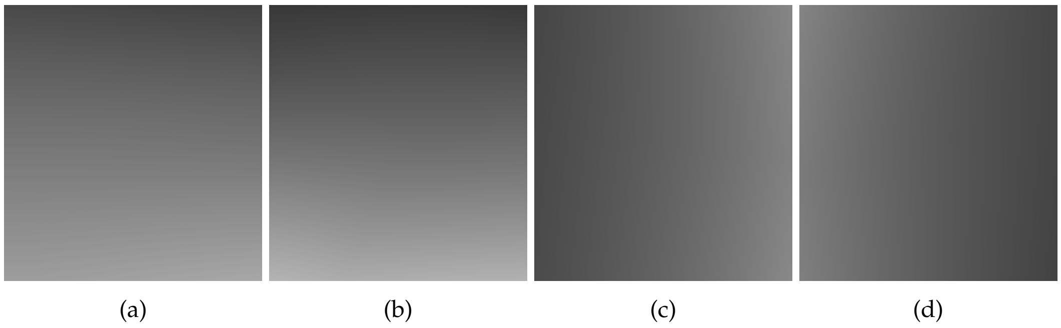


FIGURE C.5: Samples of the input data, around a vertex with valency 4. Rendered using PN triangles (a, b) and Phong tessellation (c, d), labeled as smoothness C^0.

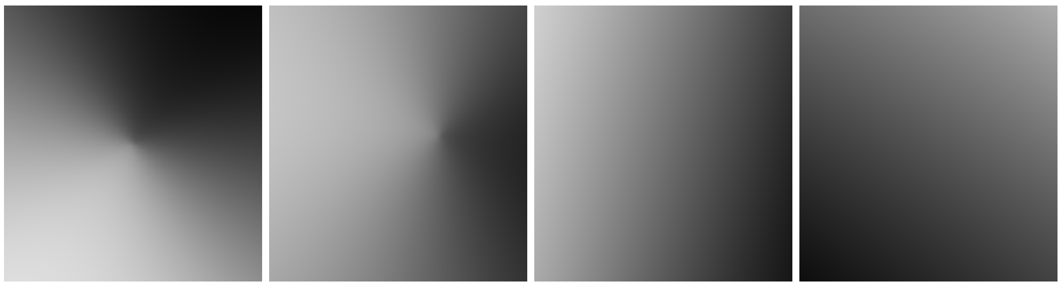


FIGURE C.6: Samples of the input data, around a vertex with valency 4. Rendered using Loop subdivision, labeled as smoothness C^1, and Catmull-Clark subdivision, labeled as smoothness C^2.

Bibliography

[1] C. de Boor. *A Practical Guide to Splines*. Vol. 27. Cambridge, Mass.: Springer New York, 1978. ISBN: 978-3-540-90356-7.

[2] T. Boubekeur and M. Alexa. *Phong tessellation*. Vol. 27. 2008, p. 141.

[3] E. Catmull and J. Clark. *Recursively generated B-spline surfaces on arbitrary topological meshes*. Vol. 10. 1978, pp. 350–355. DOI: 10.1016/0010-4485(78)90110-0.

[4] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015. 2014. URL: http://arxiv.org/abs/1412.6980.

[5] C.T. Loop. *Smooth subdivision surfaces based on triangles, Master Thesis*. Department of Mathematics, University of Utah, 1987.

[6] *ORQA: Objective Rendering Quality Assessment*. 2020.

[7] B.T. Phong. *Illumination for computer generated pictures*. Vol. 18. 1975, pp. 311–317.

[8] R.Y. Rubinstein and D.P. Kroese. *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation, and Machine Learning*. Springer-Verlag, New York, 2004. ISBN: 978-0-387-21240-1.

[9] Alex Vlachos et al. *Curved PN Triangles*. New York, NY, USA, 2001, 159–16. ISBN: 1-58113-292-1. DOI: 10.1145/364338.364387.

[10] J. Warren and H. Weimer. *Subdivision Methods for Geometric Design*. Morgan Kaufmann Publishers, 2002.