UCL Université catholique de Louvain

École polytechnique de Louvain (EPL)

Watermarking of models destined to 3D printing by deformation of surfaces

Dissertation presented by
Antoine PRIEËLS

for obtaining the Master's degree in
Computer Science and Engineering

Supervisor(s)
Benoît MACQ

Reader(s)
Sébastien LUGAN, Patrice RONDÃO ALFACE

Academic year 2017-2018

Acknowledgement

Foremost, I would like to thank my supervisor, Benoît Macq, for his support and motivation throughout the year. He always took the time to see me and give me some advice. He also helped me find the right tools and persons that could help me during the whole process of research.

Then, I must express my very profound gratitude to Patrice Rondão Alface for his precious help and his interesting ideas. He spent a lot of time to help me during different meetings, call conferences or even through emails exchanges. He shared with me his code and experience. This thesis would not be as it is without him.

I am grateful to Gaëtan Cassiers for taking the time to meet me to help me continue his work and give me advice on how to get into the subject and get started with it.

Finally, I would like to thank my friends, family and Morgane Dieleman for their feedback and continuous encouragement throughout my years of study and through the process of researching and writing this thesis.

Abstract

The use of digital models has not stopped growing in the last few years due to the expansion of 3D printing. Following this growing demand of digital models, the industry has expressed a strong interest in the ability to watermark the models they create. The term "watermarking" defines the process of inserting a mark into an object that should be sold or kept private. Its primary goals are to prove the ownership or identify the source of an unauthorized distribution.

This works reviews three different watermarking schemes and compares them in terms of robustness, imperceptibility, capacity and time. The first algorithm presented is based on the principle of volume moments in order to determine which patches to modify and to what extent. The two next algorithms use a spectral decomposition to select and modify the frequencies of the model that would cause the barely visible distortions. They differ mostly in the way that they divide the model into sub-meshes. The first one creates classical patches while the second one divides the model into layers.

Contents

1 Introduction

The use of digital models has not stopped growing in the last few years due to the expansion of 3D printing. Every year, more and more people buy 3D printers to be able to reproduce a large range of objects in an industrial context or even at home. Following this growing demand of digital models, the industry has shown an strong interest in the ability to watermark the models they want to sell or to share.

Watermarking consists of adding a mark to an image, a film or even a physical object. This mark should be impossible to remove without deteriorating the object itself but should be easily read. The goal of such a watermark is to insert hidden data into an object destined to be sold or kept private to be able to know the source of an unauthorized distribution or to prove the ownership and intellectual property of an asset.

When applied to three-dimensional models, watermarking can take multiple forms. Meshes can be represented using two types of information, a geometry which is a set of points that represents the shape of the model, and a connectivity which are links that connect the points to one another to form the surface of the model. Both those data types can be used to watermark a 3D model. Watermarking the connectivity of a mesh is performed by changing the length or the direction of the links that form the mesh. The geometry of a mesh is watermarked by slightly moving the position of the points, therefore, deforming the surfaces of the mesh.

A lot of different techniques to watermark digital models have been introduced in the last few years. In our case, we are focusing on models destined to 3D printing, this means that we should be able to retrieve a watermark by scanning a 3D printed model. This print-scan process might result in a model whose connectivity is completely different from the original model and whose geometry has been slightly modified for the needs of the printing process. Watermarking the connectivity of the object is thus to be banned and all the information should be hidden in the geometry of the model, keeping in mind that some inaccuracies might be introduced before the retrieval of the inserted signature.

The work presented here follows what has been done last summer by Gaëtan Cassiers [2] during his internship at the Massachusetts Institute of Technology's Center for Bits and Atom with the help of Neil Gershenfelds. We will take as starting point their implementation of the watermarking algorithm based on volume moments proposed by Wang *et al.* [10].

1.1 Objectives

The main objective of this thesis is pretty straightforward. We want to implement an algorithm that will insert and extract a watermark in a 3D model and make it as efficient as possible regarding the following criteria: robustness, imperceptibility, capacity and time.

The first direction that we took was to finish the implementation of the watermarking scheme based on volume moments. This part intended to fix some of the existing bugs, adapt the code to be used on any model, decrease the error rate of the watermark retrieval and make the algorithm usable by the industry. A second part of the work consisted in evaluating the performances and the possibilities of practical applications.

It then came to our mind that another scheme with the same objectives and similar applications was previously developed by Cayre *et al.* [3] with the help of researchers at Université Catholique de Louvain. It then appeared that a comparison of the two approaches would be really interesting and that the two algorithms could benefit from the ideas and solutions proposed

by one another. We then reoriented our work towards a new implementation of this solution in order to have sufficient data to compare those two algorithms.

1.2 Outline

This paper presents the progress that has been made during the last few months. As we want to focus on the comparison of three watermarking algorithms, we will try to be as objective as possible.

Section 2 will first go through all the background knowledge that is needed to understand what this work is about. In Section 2.1, we will explain how three-dimensional models are represented. Then, we will cover watermarking in general in Section 2.2 and watermarking applied more specifically to 3D models in Section 2.3. This part will try to focus more on applications of watermarking algorithms by explaining the possible use cases of such algorithms and the ways we have to evaluate their performances. To finish off with the background knowledge, we will go to a more practical subject that is the constraints and parameters of additive manufacturing in Section 2.5.

After all of those theoretical explanations, we will start with the more interesting subjects which are the algorithms themselves. The three next sections will each focus on a specific watermarking technique. Section 3 will cover the technique based on volume moments, Section 4 will present another technique that was developed with the help of researchers from Université Catholique de Louvain and which is based on the decomposition of the model into its spectral coefficients, and then, Section 5 will try to present an alternative to the spectral decomposition algorithm that better suits the needs and constraints of additive manufacturing.

Each of those sections will follow a simple structure. First, we will try to explain as directly as possible the principles behind those techniques, then we will evaluate them based on the criteria developed in the Section 2.4 and finally we will try to interpret those results.

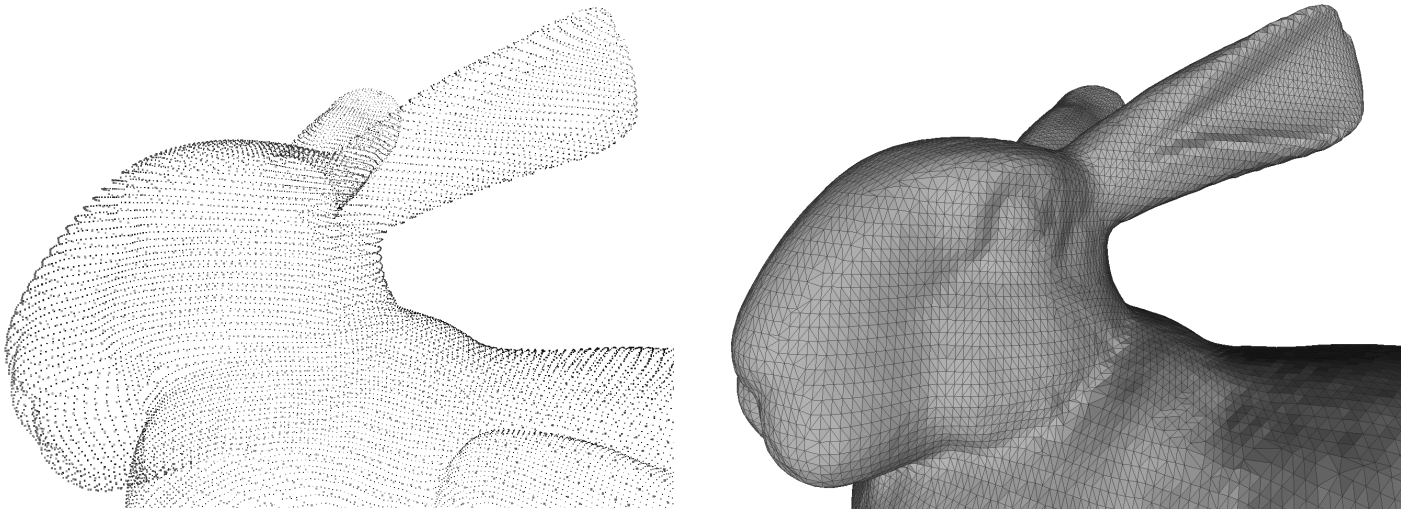
Next, in Section 6, we will consider those three algorithms following the same criteria as previously in order to compare them and determine which algorithm best suits our needs. A small section (Section 7) will be dedicated to the software implementation of those algorithms and how they were developed. Right before concluding, we will cover the possible leads that could follow this work in Section 8 in case anyone will be inspired and would like to take part in this journey.

2 Background Knowledge on Digital Models Representation, Processing and Printing

2.1 Mesh Representation

Numerically, there exist two ways to represent 3D shapes. The first one uses a combination of polyhedrons to represent the volume occupied by the shape itself while the second one describes only the "shell" formed by the shape using polygons that fit its borders. Most of the time, in 3D printing, we do not really care about the inside of models, so we use the shell representation. Using simple polygons such as triangles or quadrilaterals makes the rendering and processing of meshes trivial, so they are the most commonly used ones. This kind of meshes is composed of two types of information: a geometry and a connectivity.

The geometry is a set V of N vertices described by their three-dimensional coordinates that will give its shape to the model. Then, the connectivity groups the vertices together in order to form a list of faces that will link the nodes to each other using edges. Such a representation allows a trivial rendering of the model as represented in Figure 2.1.



(a) Geometry of a mesh

(b) Connectivity of a mesh

Figure 2.1: Mesh representation of a digital model

Additional information can be given to meshes as attributes. This information includes colors, textures, normals, or, more generally, any kind of information that can be used to render or process a mesh.

2.2 Watermarking

As explained by Cox [6], information hiding is a general term that describes the process of hiding messages or data into another container. In opposition to steganography which tries to make the very existence of the message itself a secret, watermarking only focuses on making the message imperceptible.

The most common use of a watermark is to assert the identity and authenticity of the owner. Contrary to simpler techniques that try to associate some additional information with a work such as inserting an explicit text or modifying the title of a file, watermarking has three advantages:

- The inserted data is imperceptible. The object can still be used after the introduction of a watermark the same way that the non-watermarked object could be used.
- The watermark cannot be separated from the object that it protects. No matter how hard an attacker tries to remove a watermark, it should be impossible, or at least nearly impossible, to remove the watermark without degrading the object itself.
- The watermark undergoes the same transformation as the work. This means that, from the watermark, the owner of the object could determine if its creation has been modified, and sometimes, even how it has been modified.

Watermarking is a term that concerns any type of object, even physical. The emergence of digital content such as images, audio or even 3D models has created a new field of watermarking designated as digital watermarking. The principles behind digital watermarking are exactly the same as previously except that its purpose is to be applied to non-physical data.

2.3 Digital Watermarking of 3D models

Adding a watermark to a 3D model is a quite difficult problem. It is performed by modifying either the geometry or the connectivity of a mesh in such a way that someone will then be able to make sense out of the introduced modifications. Modifications on the geometry can be introduced by slightly shifting the different vertices while making sure that the deformations are almost imperceptible. On the other hand, modifications on the connectivity can, for example, insert new vertices in the middle of a face in order to create smaller faces without impacting the global shape of the mesh.

2.3.1 Use Cases

Cox [6] proposed a few interesting applications of digital watermarking. Some of them also apply to watermarking of 3D models and are usually somehow linked to the ownership of the object. They include :

- **Owner Identification:** a watermark can be used in a similar way than a textual copyright notice to find the legitimate owner of a 3D model in case its product was misused.
- **Proof of ownership:** similarly to the previous use case, a watermark can also be used to prove that something belongs to you.
- **Transaction tracking:** this consists in including one or more transactions that have taken place in the history of the copy of a work. For example, the watermark might record the recipient in each legal sale or distribution of the work. It is also called fingerprinting as each copy can be identified uniquely.
- **Content authentication:** watermarks can also be used to make sure that some object has not been modified. A digital signature, which is essentially an encrypted summary of the message, is inserted as a watermark in a model. This way, when a customer receives a new model, by recomputing this signature, he can make sure that the model itself was not altered during the sending.

2.4 Watermarking Algorithm Evaluation

2.4.1 Blindness

There exist three kinds of watermarking algorithms: blind, semi-blind and non-blind. The blind ones do not need the original model to extract the data from a watermarked one while non-blind algorithms need to compare the two of them to retrieve the embedded data.

- **Non-blind watermarking:** a non-blind watermarking scheme needs the original model in order to be able to detect or retrieve the inserted data.
- **Semi-blind watermarking:** the goal of semi-blind schemes is to detect whether a certain watermark can be read from a watermarked model. It thus cannot retrieve a watermark if we have no idea what value it has.
- **Blind watermarking:** a blind watermark is able to retrieve the whole watermark without needing the base model. Since anyone can have access to the watermarked model, it can be useful to encrypt the data before inserting it into the model when using a blind watermark.

2.4.2 Robustness

The possible inaccuracies introduced by the process of printing and scanning should not have any impact on the readability of the watermark. Also, the watermark should be difficult to remove, even knowing the algorithm used and the data that was inserted. There exist multiple ways to remove a watermark from a model. The most common one consists in adding noise to the watermarked model then smoothing the resulting mesh.

Robustness against remeshing It could totally be conceivable to design algorithms that do not deform the global shape of the model at all, making the watermark completely imperceptible. These schemes would use techniques based on re-meshing that alter the connectivity of a mesh without modifying its geometry. The problem of such algorithms relies upon the fact that a slight modification of the connectivity might make the whole watermark unreadable. In our case, we want the watermark to resist the process of printing and scanning. Such mechanisms cannot ensure that the connectivity will be preserved. Total imperceptibility is therefore impossible in our case because the watermark needs to be embedded in the geometry and not in the connectivity of the mesh.

We then want to find an algorithm that inserts a watermark by slightly moving the nodes of the mesh. The amplitude or the direction of the shifting will indicate the value of the embedded data bit. The difficulty is to modify it enough so that the watermark can be retrieved but not too much so that the modifications of the model are not perceptible.

Robustness against printing and scanning More than re-meshing, the process of printing and scanning might introduce some imprecisions in the coordinates of the vertices or even some deformations due to the resolution limitations of the different steps. The bit error rate should stay somehow limited to ensure to good retrieval of the watermark.

Robustness against geometrical attacks Finally, we want our algorithm to be as resistant as possible against possible attackers. The most common attacks against 3D watermarking algorithms are of two types. The first of these common attacks is to add some random noise to the whole model. Another option is to smooth the model in order to homogenize the introduced deformations. These two procedures are intended to randomly modify the values introduced by the watermark.

2.4.3 Imperceptibility

Modifications applied to the model by the embedded bits should remain as discrete as possible. The problem of the imperceptibility is even more constraining in applications destined to the industry. The smallest variations in the shape of a model might make the object completely unusable if it cannot fit in its destination or if the variations make the model structurally weaker. Furthermore, parts destined to be used in machines are usually quite smooth and flat so no

specific area will be particularly suited to be modified.

We will use three different methods to evaluate the imperceptibility. They are all based on a similar principle which is the computation of the distance between a point in the original model and in the watermarked one. In the following paragraphs, the notation $\|p - p'\|$ will be used to represent the euclidean distance between the two points p and p'

Root-Mean-Square Error This metric simply uses the distance between the same vertex in the two different models.

$$\sqrt{\sum_{v_i \in V} \|v_i - v_i'\|^2}$$

where v_i is a vertex in the original model and v_i' is the same vertex in the watermarked model.

Hausdorff Distance The Hausdorff distance [1] is a metric that can be applied to 3D models which computes the minimal distance between a vertex in a first model to any other vertex in a second model. In other words, we define the distance between a vertex in the original model and the set of vertices of the watermarked model as

$$d(v_i, V') = \min_{v_i' \in V'} \|v_i - v_i'\|$$

The Hausdorff distance is then obtained as

$$d(V, V') = \max_{v_i \in V} d(v_i, V')$$

It is relevant to note that this metric is not symmetrical. We can define the symmetrical Hausdorff distance as

$$d_s(V, V') = \max[d(V, V'),\ d(V', V)]$$

Local Smoothness Finally, the local smoothness of a mesh is a good indicator of imperceptibility of a watermark. As Cayre *et al.* [3] stated, the eye better accommodates on a smooth object rather than on a noisy mesh. We will then compute the distance between of vertex to the center of its neighbors. We define the geometrical local Laplacian that represents the local smoothness as

$$GL(v_i) = v_i - \frac{\sum_{j \in \{i^*\}} \|v_i - v_j\|^{-1} v_j}{\sum_{j \in \{i^*\}} \|v_i - v_j\|^{-1}}$$

The visual distance between two meshes is therefore based on the actual distance between their vertices but also on the distance between their geometrical Laplacians:

$$D_{vis}(V, V') = \frac{1}{2|V|} \left(\sum_{i=0}^{N-1} \|v_i - v_i'\| + \sum_{i=0}^{N-1} \|GL(v_i) - GL(v_i')\| \right)$$

2.4.4 Capacity

The bits capacity is a determining factor for the choice of a watermarking algorithm. More capacity means more data to be inserted but more capacity can also mean more redundancy. Increasing the redundancy in a model will lead to increased robustness but also reduced imperceptibility.

2.4.5 Time

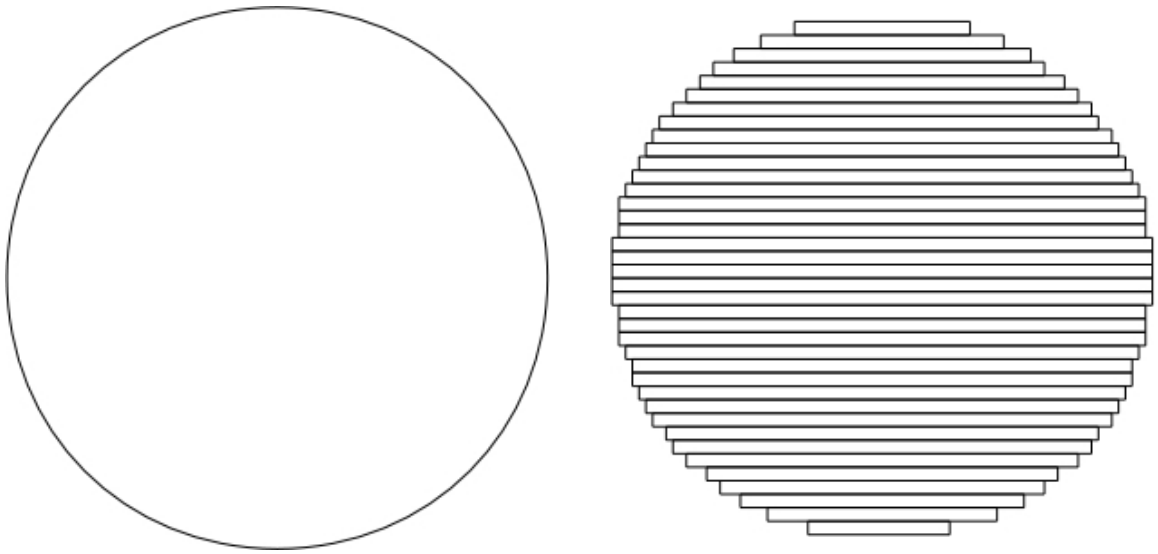
Finally, time complexity could also help to choose a watermarking algorithm. Linear algorithms would be preferred over exponential ones.

2.5 Additive Manufacturing

There exist a lot of different 3D printing techniques, most of them regroup under the more general appellation "Additive Manufacturing". This term is used to define all printing techniques that build 3D objects by adding layers-upon-layers of materials. Even though their principle is the same, their applications are quite different.

Kruth *et al.* [8] stated that we can first differentiate them based on the type of bulk material they use. The material itself can go from polymers to metal, to composites or even to ceramic. The state of the material can also vary. Most common applications such as Fused Deposition Modeling (FDM) or Stereolithography (SLA) need a liquid input while others are based on solid[1], powdery[2] or gaseous[3] inputs.

A common constraint of all additive manufacturing techniques is the slicing problem. Since 3D printers operate layer by layer, the model of the object to be printed should first be divided into layers. This process can introduce some inaccuracies since a curve following the z-axis must be approximated using vertical faces. Figure 2.2 presents a simplified representation of this process and clearly shows that we lose accuracy. A parameter that can help reduce this inaccuracy due to the slicing is to modify the resolution along the z-axis, which is related to the layer thickness.



(a) Model to be sliced

(b) Sliced model

Figure 2.2: Simplified representation of the slicing process

[1]Laminated Object Manufacturing (LOM)
[2]Three-Dimensional Printing (3DP), Selective Laser Sintering (SLS)
[3]Selective Laser Chemical Vapor Deposition

3 Watermarking Based on Volume Moments

Wang *et al.* [10] introduced a new technique based on geometrical volume moments. Their first assumption was that a good watermark needs to be intrinsically linked to the 3D shape that is behind the mesh. In other words, areas of the mesh that have the roughest surfaces should be the ones to be deformed. Deforming a flat shape would make the watermark pretty obvious.

3.1 Principles

The principle of this algorithm is to insert one bit of the watermark per patch by deforming it. The introduced deformation should modify the volume moment of the patch by a value that will depend on the bit to be inserted. The difficulty is to make sure that the patches obtained during the retrieval process are the same that were used in the embedding step. The computation of the volume moments of the watermarked patches, defined in Section 3.1.1, then allows us to determine the bit that was inserted.

Figures 3.1 and 3.2 show the different steps that make the processes of embedding and retrieval of a watermark.

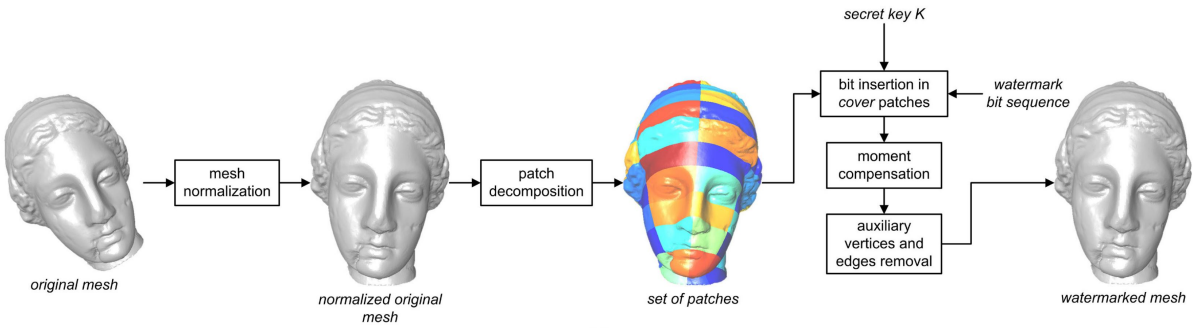


Figure 3.1: Watermark based on volume moments embedding process

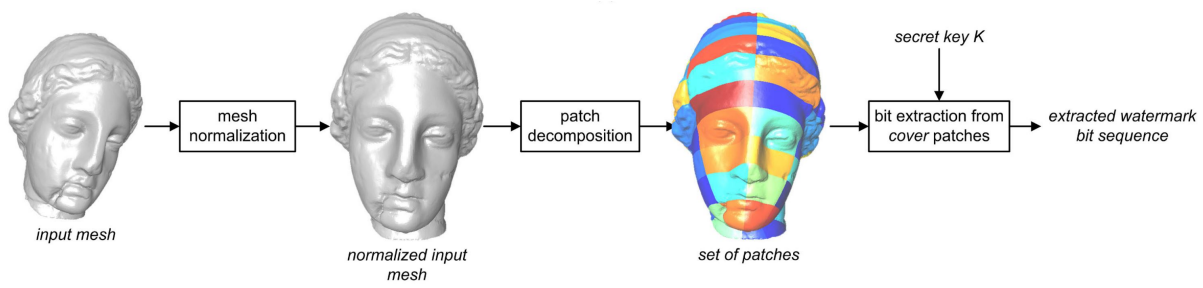


Figure 3.2: Watermark based on volume moments retrieval process

3.1.1 Volume Moments

The parts of the mesh that have the highest variations in their volume moments can support more significant deformations while keeping the watermark unnoticed. Volume moments are defined as :

$$m_{pqr} = \int x^p y^q z^r \ \rho(x, y, z) \ dx \ dy \ dz \tag{1}$$

where p, q, r are the orders and $\rho(x, y, z)$ is the volume indicator function which is equal to 1 if (x, y, z) is inside the closed surface or 0 otherwise.

3.1.2 Mesh Normalization

The goal of mesh normalization is to make sure that we get the same patches when inserting or reading a watermark. First, the center and the principal axes of the mesh are computed. This can be done easily using volume moments. The center of the mesh and its principal axes are respectively given by C and the eigenvectors of M.

$$C = (x_c, y_c, z_c) = (\frac{m_{100}}{m_{000}}, \frac{m_{010}}{m_{000}}, \frac{m_{001}}{m_{000}}) \tag{2}$$

$$M = \begin{bmatrix} m_{200} & m_{110} & m_{101} \\ m_{110} & m_{020} & m_{011} \\ m_{101} & m_{011} & m_{002} \end{bmatrix} \tag{3}$$

The normalization is achieved in three steps :

1. **Translation :** the mesh is translated so that its center coincides with the center of the targeted Cartesian coordinate system.
2. **Rotation :** a rotation is applied to all the vertices so that the three principal axes are aligned to the coordinate system.
3. **Scaling :** the mesh is scaled so that it is bounded within a unit sphere

There are two cases that might lead to different normalizations of the base mesh and the watermarked mesh. The first one is the case of an attack. Important local deformations might change the result of the computation of the principal axes. This is the case for all blind watermarking algorithms and no one seems to have found a solution. The other problematic case happens when watermarking spheres or other symmetrical objects because they present a lot of equivalent axes with different directions that would result in different patches decompositions.

3.1.3 Patch Decomposition

Before inserting the bits of the watermark into the mesh, we need to divide it into patches that will contain data bits. This patch decomposition should give the same patches during the embedding than during the retrieval of the watermark bits. Therefore, the patch decomposition cannot depend on the shapes of the mesh.

The solution that was found is to divide the set of vertices depending on their cylindrical coordinates. The h and θ domains are partitioned into I_h and I_θ intervals with steps h_{step} and θ_{step}. Faces that cover multiple patches need to be split into smaller faces in order to avoid changing the volume moment of the neighbor patches when moving some vertices.

Note that decreasing the steps h_{step} and θ_{step} would create more patches and thus increase the number of bits that can be inserted into the model but would also lead to higher-amplitude deformations.

3.1.4 Patch Classification

Patches that have the highest moment variations are the ones that can be deformed while keeping the modification less perceptible. They then need to be divided into classes, depending on their possible use:

- **Discarded patches:** this group contains patches that have low zero-order moment amplitudes or small h or θ ranges and that will not support strong modifications. These patches will not be used.
- **Compensation patches:** the twelve patches that have the largest moment amplitudes allow important deformations and will be used as compensation patches.
- **Cover patches:** the N remaining patches will be used to insert $N-1$ watermark bits. They are noted as $\mathcal{P}^w_{n,\ n=0,1,\ldots,N-1}$

3.1.5 Moment Quantization

The watermark bit is inserted by quantizing the zero-order moment of the patch. Instead of simply replacing the initial value with the quantized one, the initial zero-order moment is 'pushed' towards the quantized one by a value u defined as:

$$u_{m_{000}^{(\mathcal{P}_n^w)}} = z.\Delta^{(\mathcal{P}_n^w)} + a.\frac{\Delta^{(\mathcal{P}_n^w)}}{2} + t^{(\mathcal{P}_n^w)}\Delta^{(\mathcal{P}_n^w)} \tag{4}$$

where:

- z is a parameter that allows to control the insertion intensity
- $\Delta^{(\mathcal{P}_n^w)}$ is the quantization step, that is approximately proportional to the moment amplitude and is detailed in Equation 5

$$\Delta^{(\mathcal{P}_n^w)} = \begin{cases} \Delta_{pre}.\left| m_{000}^{(\hat{\mathcal{P}}_{n-1}^w)} / \left\lceil \frac{m_{000}^{(\hat{\mathcal{P}}_{n-1}^w)}}{m_{000}^{(\mathcal{P}_n^w)}} \right\rceil \right|, & \text{if } \left| \frac{m_{000}^{(\hat{\mathcal{P}}_{n-1}^w)}}{m_{000}^{(\mathcal{P}_n^w)}} \right| > 1 \\ \Delta_{pre}.\left| m_{000}^{(\hat{\mathcal{P}}_{n-1}^w)} / \left\lfloor \frac{m_{000}^{(\mathcal{P}_n^w)}}{m_{000}^{(\hat{\mathcal{P}}_{n-1}^w)}} \right\rfloor \right|, & \text{otherwise} \end{cases} \tag{5}$$

$$\Delta_{pre} = \begin{cases} 0.04, & \text{if } \left| m_{000}^{(\mathcal{P}_n^w)} \right| > 0.01 \\ 0.07, & \text{if } \left| m_{000}^{(\mathcal{P}_n^w)} \right| \leq 0.01 \end{cases} \tag{6}$$

- $a \in \{0,1\}$ is the value of the bit to be inserted
- $t^{(\mathcal{P}_n^w)}\Delta^{(\mathcal{P}_n^w)}$ is an additive pseudo-random dither signal, $t^{(\mathcal{P}_n^w)}$ being a sequence of random variables uniformly distributed between $[-\frac{1}{2}, \frac{1}{2}]$ generated using a secret key.

The targeted volume moment value is given by the following expression:

$$m_{000}^{(\hat{\mathcal{P}}_n^w)} = m_{000}^{(\mathcal{P}_n^w)} + \alpha^{(\mathcal{P}_n^w)}(u_{m_{000}^{(\mathcal{P}_n^w)}} - m_{000}^{(\mathcal{P}_n^w)}) \tag{7}$$

with $\alpha^{(\mathcal{P}_n^w)} \in [0,1]$, a compensation factor that should be greater or equal to 0.5 to ensure the correctness of the watermark extraction.

3.1.6 Patch Deformation

Each patch needs to reach its target zero-order moment value. The transformation of the zero-order moment is not reversible so the patch needs to be deformed progressively until the targeted value is reached.

The process first normalizes the coordinates in order to create a smooth deformation. Vertices close to the patch border will move less than vertices in the middle of the patch as represented in Figure 3.3. Updated coordinates are obtained with:

$$h'_k = 1 - \left| \frac{2(h_k - h_{min}^{(\mathcal{P}_n^w)})}{h_{max}^{(\mathcal{P}_n^w)} - h_{min}^{(\mathcal{P}_n^w)}} - 1 \right| \in [0,1] \tag{8}$$

$$\theta'_k = 1 - \left| \frac{2(\theta_k - \theta_{min}^{(\mathcal{P}_n^w)})}{\theta_{max}^{(\mathcal{P}_n^w)} - \theta_{min}^{(\mathcal{P}_n^w)}} - 1 \right| \in [0,1] \tag{9}$$

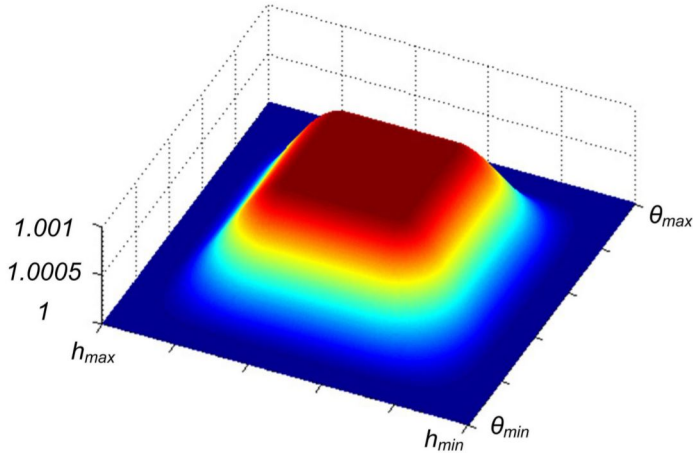


Figure 3.3: Deformation of a single patch

The values of h'_k and θ'_k will then be translated into weights ($wt_{h'_k}$ and $wt_{\theta'_k}$) that will be used to compute the deformation factor s_{v_k} of every single vertex v_k, depending on the global deformation factor s:

$$wt_{h'_k} = \begin{cases} 0, & \text{if } 0 \leq h'_k < 0.1 \\ \frac{1}{2}\sqrt{|s-1|}[\sin(\frac{5\pi}{3}(h'_k - \frac{2}{5})) + 1], & \text{if } 0.1 \leq h'_k < 0.7 \\ \sqrt{|s-1|}, & \text{if } 0.7 \leq h'_k \leq 1.0 \end{cases} \tag{10}$$

$$wt_{\theta'_k} = \begin{cases} 0, & \text{if } 0 \leq \theta'_k < 0.1 \\ \frac{1}{2}\sqrt{|s-1|}[\sin(\frac{5\pi}{3}(\theta'_k - \frac{2}{5})) + 1], & \text{if } 0.1 \leq \theta'_k < 0.7 \\ \sqrt{|s-1|}, & \text{if } 0.7 \leq \theta'_k \leq 1.0 \end{cases} \tag{11}$$

$$s_{v_k} = \begin{cases} 1 + wt_{h'_k}.wt_{\theta'_k} & \text{if } s > 1 \\ 1 - wt_{h'_k}.wt_{\theta'_k} & \text{if } s < 1 \end{cases} \tag{12}$$

Finally, the coordinates of the moved vertex are given by the multiplication of the base coordinates by the deformation factor s_{v_k} or $(2 - s_{v_k})$ if the faces connected to the vertex have a negative volume moment.

3.1.7 Moment Compensation

The goal of moment compensation is to retrieve the same center and principal axes than before the deformation so that the patch decomposition gives the same result and that the watermark can, therefore, be extracted. The algorithm from Section 3.1.6 is applied to compensation patches to compensate the modifications of volume moments introduced in the previous step.

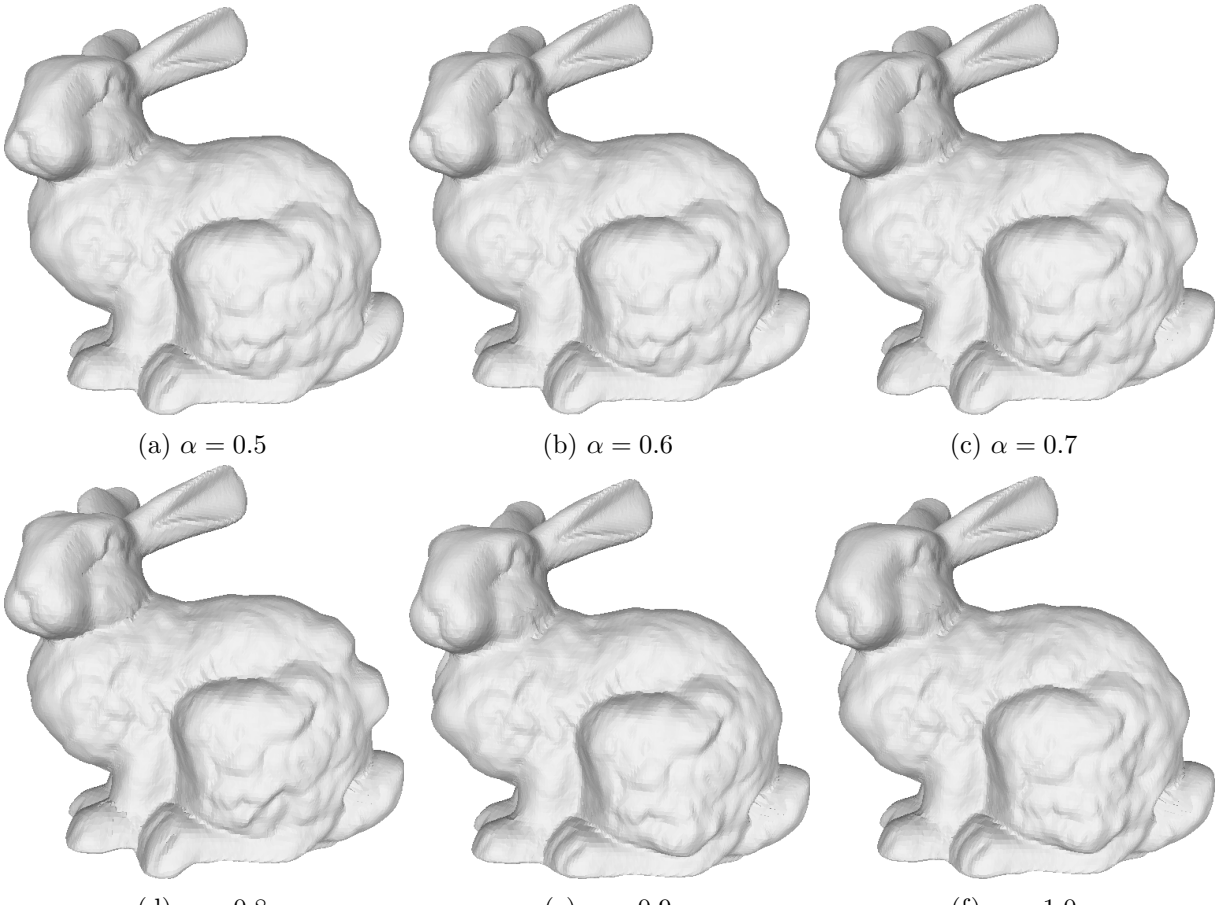
3.2 Evaluation

As stated in Section 2.4, the evaluation of a watermarking algorithm is based on the following criteria : **imperceptibility**, **capacity** and **robustness**. All our tests will be performed on the same base model of the Stanford Bunny, which is composed of 34834 vertices and 69664 faces.

3.2.1 Imperceptibility

Let's first take a look at the imperceptibility of the watermark. The first step we performed is to check visually the influence of the α parameter, which corresponds to the strength of the watermark, onto the model. The value of α should be included between 0.5 and 1.0 to make sure that the watermark is retrieved correctly.

As shown in Figure 3.4, by modifying the α parameter, the displacement applied to the different patches seems to vary, as expected. Surprisingly, and contrary to what one might think, the watermarked model with the highest α (Figure 3.4f) is not the one that presents the most visible modifications. This might be due to the fact that bigger modifications do not necessarily introduce bigger perceptible changes. If multiple patches located next to each other have important deformations, it will look quite smooth and it might not appear as abrupt as a small deformation on an isolated patch.



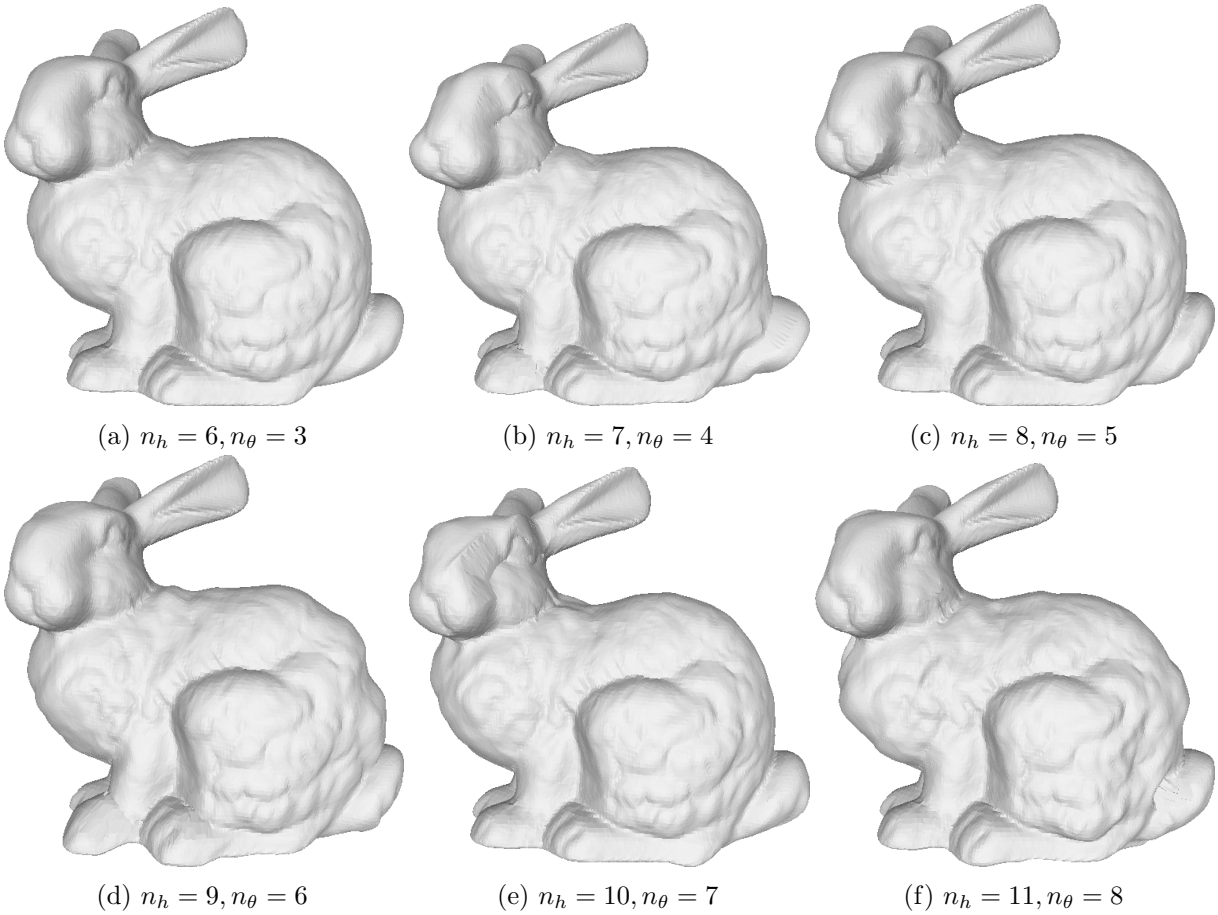
(a) $\alpha = 0.5$ (b) $\alpha = 0.6$ (c) $\alpha = 0.7$

(d) $\alpha = 0.8$ (e) $\alpha = 0.9$ (f) $\alpha = 1.0$

Figure 3.4: Varying the strength of the volume moments-based algorithm with a fixed number of patches ($n_h = 11, n_\theta = 8$)

Next, we will try to see if the number of patches has a visible impact on the model. As we can see from Figure 3.5, the main difference concerns the locations of the deformed patches. Since the patches to be deformed are selected based on their initial volume moment value, smaller patches might create different results than bigger ones.

It is also important to note that the model with the most patches seems to be the most deformed one, mostly because it also holds the greatest number of bits of data. If we inserted the same number of bits in all these models, the watermark on the 3.5f model would be the less perceptible for one main reason. The same number of patches would be deformed in all the models, but since its patches are smaller, the introduced changes would be much more localized and therefore less perceptible.



(a) $n_h = 6, n_\theta = 3$ (b) $n_h = 7, n_\theta = 4$ (c) $n_h = 8, n_\theta = 5$

(d) $n_h = 9, n_\theta = 6$ (e) $n_h = 10, n_\theta = 7$ (f) $n_h = 11, n_\theta = 8$

Figure 3.5: Varying the number of patches for a fixed strength of $\alpha = 1.0$

Let us analyze those variations in a more rigorous way, based on three objective metrics, the Root-mean-square error (Figure 3.6), the Hausdorff distance (Figure 3.7) and finally the smoothness of the mesh (Figure 3.8). As expected, the three metrics increase almost linearly when the strength increases too. As for now, the values expressed by those three metrics cannot really be interpreted, so we will have to wait until we have those values for our others watermarking schemes.

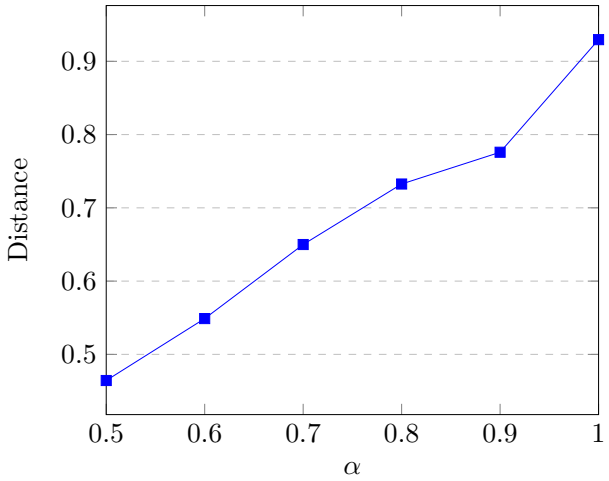


Figure 3.6: Evaluation of the Root-Mean-Square Error for the volume moments-based algorithm

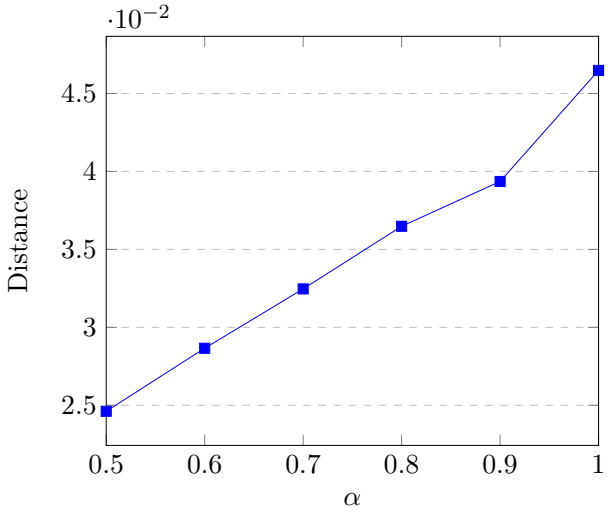


Figure 3.7: Evaluation of the Hausdorff distance for the volume moments-based algorithm

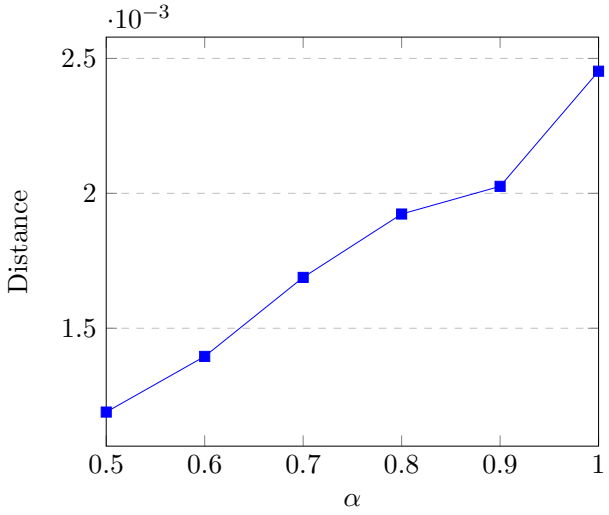


Figure 3.8: Evaluation of the Local Smoothness metric for the volume moments-based algorithms

3.2.2 Capacity

Next, let us take a look at the capacity of this first scheme. The principle of this technique is to insert a single bit per patch classified as "cover patch". The strength of the watermark has no influence on the capacity of the scheme. However, Figure 3.9 shows that the number of cover patches increases when the number of patches increases too. As we can see, the number of bits is quite limited and inferior to the usual 64 bits watermark that we want to insert. Unfortunately, as we keep increasing the number of patches, some cover patches need tremendous deformations to retrieve the original orientation of the model, which becomes unusable (Figure 3.10).

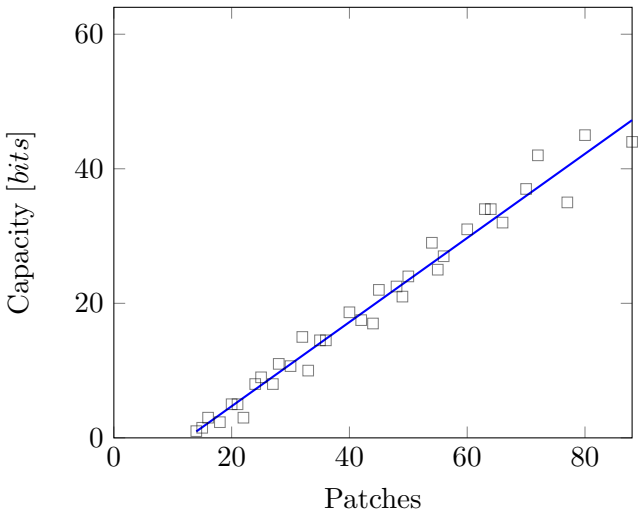


Figure 3.9: Capacity of the watermark depending on the number of patches

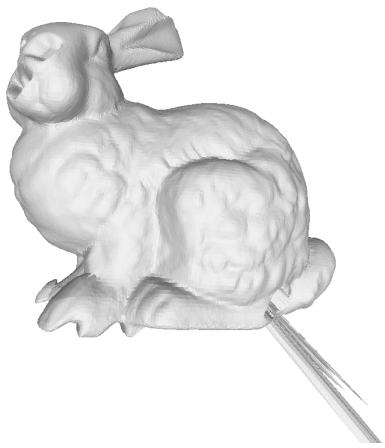


Figure 3.10: When the number of patches increases, some deformations become important

3.2.3 Robustness

We will consider the robustness of this first algorithm. First, we will ensure that leaving the watermarked model as is, we obtain a limited bit error rate. Here, the bit error rate is defined regarding the number of actually inserted bits and not the complete 64 bits signature that we tried to insert. Figure 3.11 presents the mean bit error rates of the models with a different number of patches. All of those values stay relatively close to 10%, which is quite important but restrained enough to possibly be corrected with an error correction code.

The complete results presented in Appendix A.1 show that while some patches configurations allow a bit error rate equal to zero, some others completely lose the data. Moreover, this bit error rate increases when the strength increases too which is the opposite of what was expected, a greater strength should imply a reduced error rate!

This makes us think that there is a problem in the implementation we used to perform our measurements. Since the bit error rate increases with the strength, the origin of the problem has to somehow be linked to the amplitude of the deformations, either in the cover patches or, more probably, in the compensation patches. If the compensation patches do not reorient the principal axes of the model properly, the bits we read might be completely reordered.

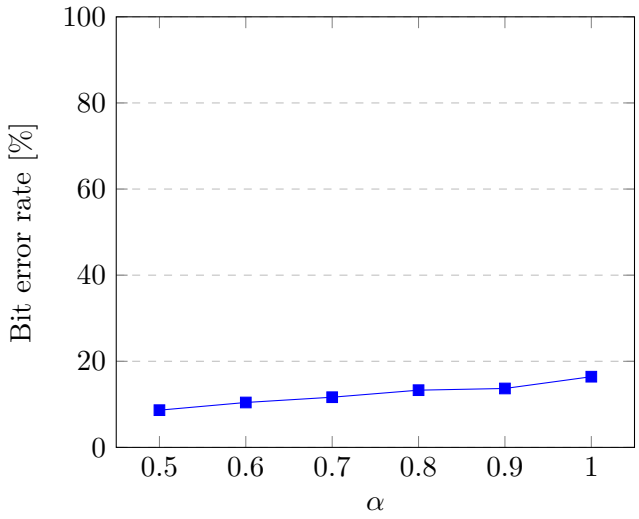


Figure 3.11: Bit error rate of the watermarking algorithm based on volume moments

Finally, we measured the bit error rate of one of our model patched into 88 patches ($n_h = 11$, $n_\theta = 8$) under two kinds of attacks: a smoothing attack and a random noise addition.

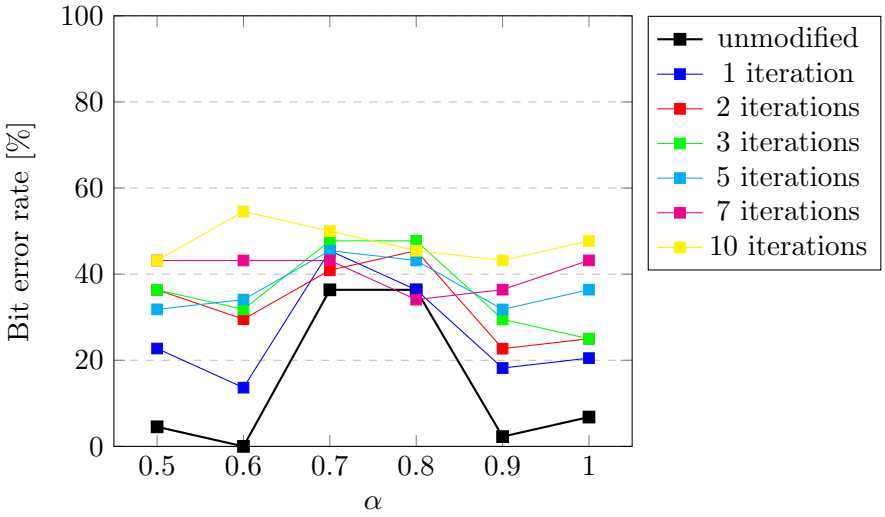


Figure 3.12: Robustness of the volume moments-based algorithm against a smoothing attack

The smoothing attack is performed by iterations, around 15% of the bits are lost after the first iteration and some more are lost during the next iterations. After seven iterations, the

watermark is almost completely lost. The strength of the watermark does not seem to have any impact on the robustness of this watermarking scheme.

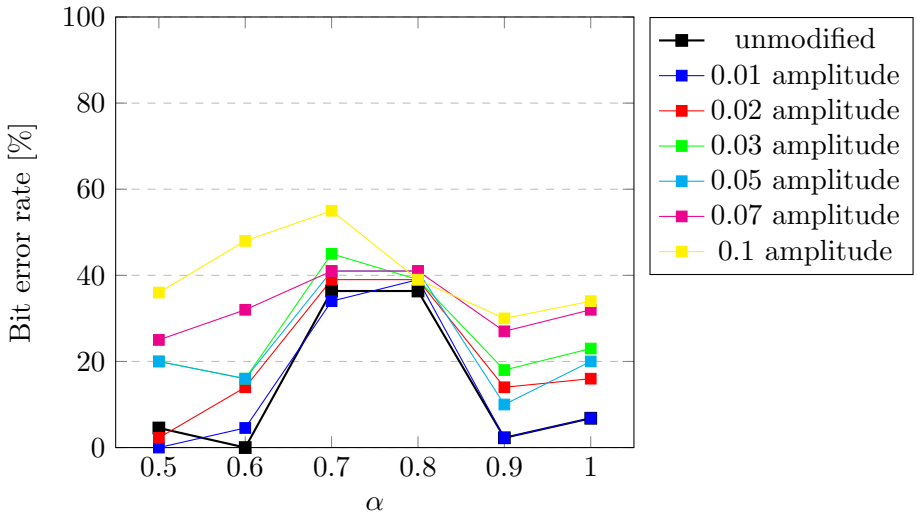


Figure 3.13: Robustness of the volume moments-based algorithm against random noise addition

The random noise addition has almost no effect on the bit error rate with modifications of an amplitude smaller than 0.3 times the coordinates of the vertices. Again, the strength of the watermark has no influence on its robustness to noise. The fact that the bit error rates of the models with $\alpha = 0.7$ and $\alpha = 0.8$ are so important is only due to the significant bit error rate of the unattacked model.

4 Watermarking Based on Spectral Decomposition

The application of spectral decomposition proposed by Cayre *et al.* [3] computes the pseudo-frequencies corresponding to the Cartesian coordinates of the nodes. The main advantage of working in the spectral space is that the smoothness of the changes is directly related to the modified frequencies. This means that modifications on the lower frequencies of the spectra will result in important, localized visual deformations contrary to higher frequencies that will result in smooth global changes.

4.1 Principles

4.1.1 Decomposition

Instead of representing the geometry of meshes as N vectors of dimension 3, spectral decomposition considers 3 vectors of size $N : (X, Y, Z)$ corresponding to the N nodes of the mesh. The goal of spectral decomposition is to find a matrix B that transforms the Cartesian coordinates into their corresponding spectra.

The *star* $\{i*\}$ of a vertex $v_i \in V$ corresponds to the set of neighbors connected to this particular vertex and is defined as

$$\forall v_j \in V : j \in \{i*\} \Leftrightarrow (v_i, v_j) \in E \tag{13}$$

where E is the set of all vertices in the model.

The degree d_i of a vertex can then be computed as

$$d_i = |\{i*\}| \tag{14}$$

Finally, the Laplacian Matrix L of a mesh is defined as

$$L_{ij} = \begin{cases} 1 & \text{if } i = j \\ -d_i^{-1} & \text{if } j \in \{i*\} \text{ and } d_i \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

The pseudo-frequencies of the geometry defined over the mesh have an orthogonal basis given by the eigenvectors $(e_0, e_1, ..., e_{N-1})$ of the Laplacian matrix L. The eigenvectors of L are given by the columns of B in the following projection system

$$\begin{pmatrix} e_0 & 0 & \dots & \dots & 0 \\ 0 & \ddots & \ddots & & \vdots \\ \vdots & \ddots & e_i & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ 0 & \dots & \dots & 0 & e_{N-1} \end{pmatrix} = B^{-1}LB \tag{16}$$

Finally, the correspondence between the Cartesian coordinates and the spectra is represented on Figure 4.1.

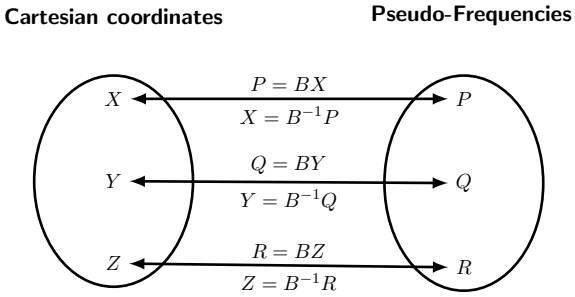


Figure 4.1: Correspondence between the Cartesian coordinates and the spectra.

4.1.2 Partitioning

The spectral decomposition raises a new problem. The computation of the eigenvalues is done in $\mathcal{O}(N^3)$ and results in poor performances. A solution is to partition the mesh into several smaller patches and to operate the previous computation on every single one of them. This partitioning is performed with the help of a tool called METIS [7].

4.1.3 Watermark Embedding and Retrieval

Watermarking needs two elements: the watermark itself and a secret key. The secret key is used to scramble the watermark bits so that the first bits do not have more importance than the rest of the watermark. The scrambled watermark is then repeated as much as possible in the bits of the mesh.

Inserting data bits at lower frequencies implies more perceptible changes on the resulting shape. We then define a strength, which corresponds to the percentage of frequencies to be modified. We use this percentage to compute the percentile of frequencies to modify. The frequencies to modify are then chosen regarding to their values. A strength of 1% will select only the 1% of frequencies that are the lowest. The greater the value of the strength, the more perceptible changes will be created but the more repetition is introduced in the mesh. Thus, this strength has a direct impact on the intensity of the watermark.

We then insert bits one by one into the coordinates of the vertices.

$$\begin{aligned} &\forall i : 0 < i < N : \\ &(P_i, Q_i, R_i) \longrightarrow (C_{min}, C_{inter}, C_{max}) \text{ with } \begin{cases} C_{min} = min(P_i, Q_i, R_i) \\ C_{max} = max(P_i, Q_i, R_i) \\ C_{inter} = mid(P_i, Q_i, R_i) \end{cases} \end{aligned} \tag{17}$$

We define a factor k that will correspond to the magnitude of the introduced modification and $\Delta = C_{max} - C_{min}$. This gives us two intervals:

$$W_0 = [C_{min}; C_{min} + \frac{\Delta}{2}] \tag{18}$$

$$W_1 =]C_{min} + \frac{\Delta}{2}; C_{max}] \tag{19}$$

The embedding process of this watermarking technique is to modify C_{inter} to be in the interval whose number corresponds to the bit to be inserted. We find four possible cases listed in Table 1. A visual representation of the process is shown in Figure 4.2.

	The bit to insert is 1	The bit to insert is 0
$C_{inter} \in W_0$	$C_{inter} = C_{min} + \frac{\Delta}{2} + \frac{C_{max} - C_{inter}}{k}$	No modification
$C_{inter} \in W_1$	No modification	$C_{inter} = C_{min} + \frac{\Delta}{2} - \frac{C_{max} - C_{inter}}{k}$

Table 1: Modifications to be applied to C_{inter} to insert a watermark bit

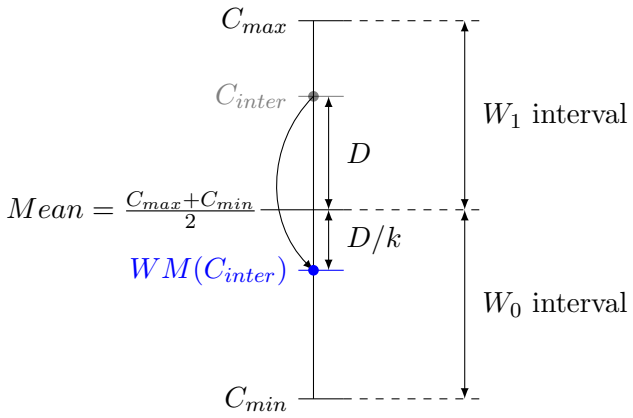


Figure 4.2: Inserting a 0 when C_{inter} is in W_1

The retrieval of the bits is straightforward. If C_{inter} is in W_0, we read a 0 and if it is in W_1, we read a 1. We can repeat this process on all the nodes to minimize the reading error. The obtained series of values can then be unscrambled using the private key to retrieve the watermark.

4.2 Evaluation

In order to evaluate this second algorithm, we will use techniques similar to what we have done previously with the algorithms based on volume moments. We will first evaluate the visual impact on the model of the introduced modifications.

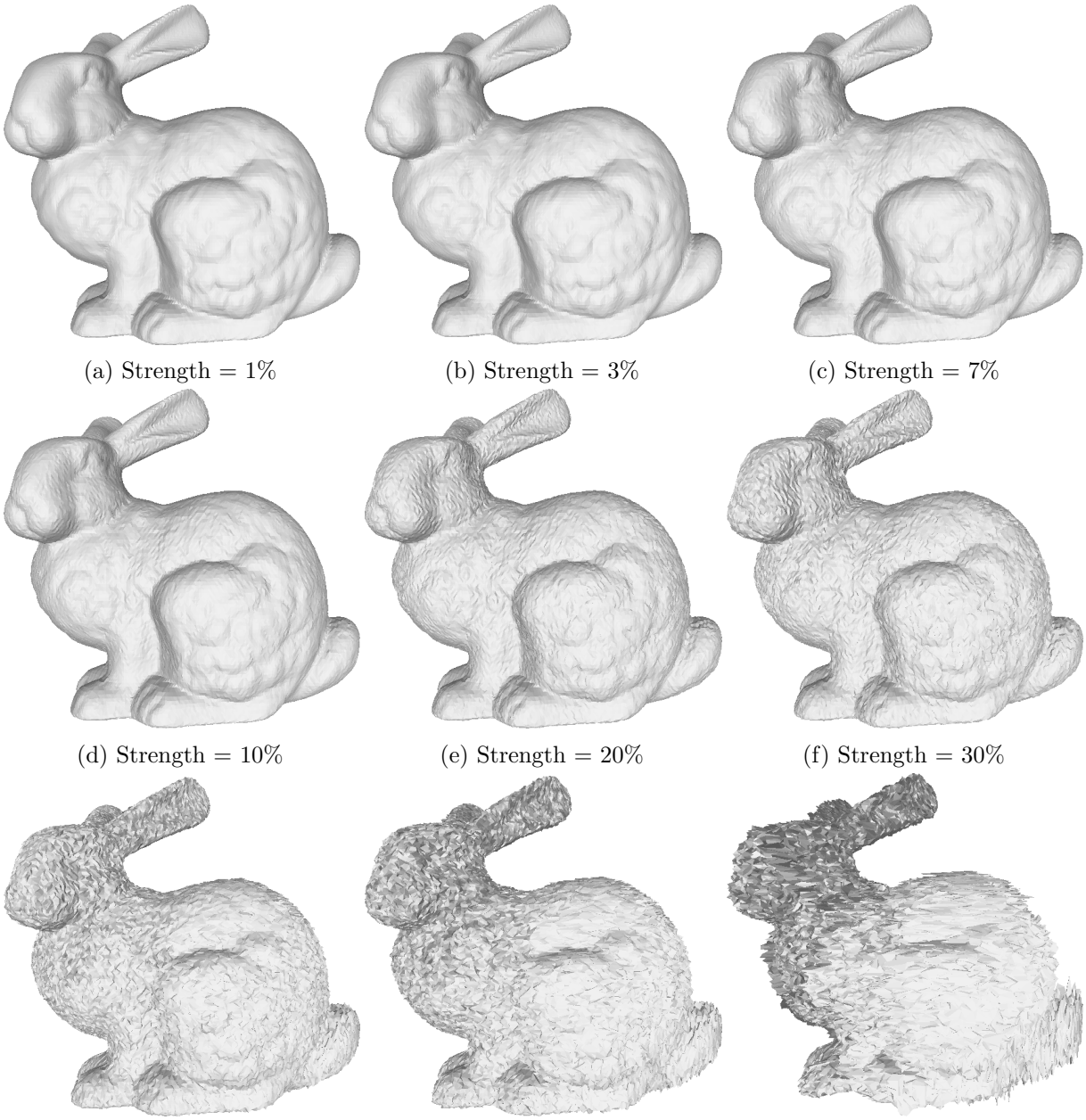
4.2.1 Imperceptibility

Let us evaluate the perceptibly of the watermark which depends on the number of patches and on the percentage of modified frequencies, the strength parameter. First, let's take a fixed strength and vary the number of patches. As we can observe from Figure 4.3, there is no perceptible difference between a model divided into 30 patches and one divided into 100 patches. The time of the insertion, however, decreases by nearly 40% when using 100 patches. Even if the modified frequencies will not be exactly the same in both cases, the amount of modifications stays the same, and they are applied to the smallest frequencies. This will make our evaluation simpler because we will not have to vary the number of patches to evaluate our imperceptibility metrics.

(a) 30 Patches (b) 100 Patches

Figure 4.3: Varying the number of patches with a fixed strength of 10%

The strength represents the percentage of frequencies that should be modified. The lowest frequencies should be the one that has the smallest perceptible impact on the model. A strength of 1% should, therefore, modify only the 1% of the lowest frequencies and make the watermark almost imperceptible. By increasing the strength to 2%, we are increasing the redundancy of the watermark, but we are also modifying higher frequencies and thus adding more changes to the one that was introduced with a strength of 1%. A strength of 100% modifies all the existing frequencies. We can clearly see from Figure 4.4 that our expectations are completely verified.



(a) Strength = 1% (b) Strength = 3% (c) Strength = 7%

(d) Strength = 10% (e) Strength = 20% (f) Strength = 30%

(g) Strength = 50% (h) Strength = 70% (i) Strength = 100%

Figure 4.4: Varying the strength of the watermark in a model divided into 60 patches

In a more formal way, the three metrics we used previously will help us compare those algorithms impartially. Figures 4.5, 4.6 and 4.7 show respectively the evaluation of the Root-Mean-Square Error, the Hausdorff distance and the Local Smoothness. All of them follow the exact same curve. Moreover, we can see that the distances increase exponentially, which confirms that the modifications on lower frequencies lead to less perceptible deformations.

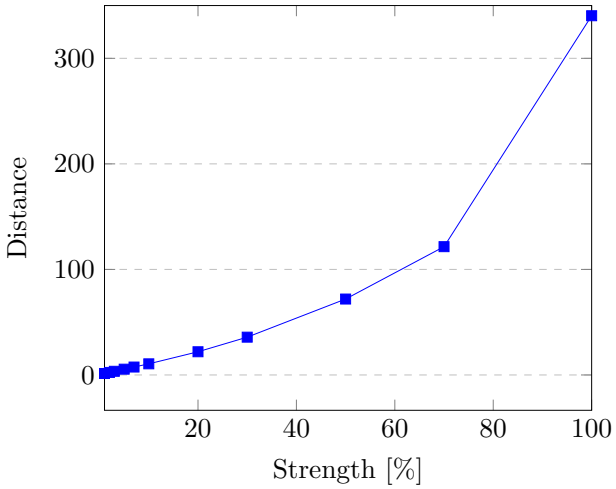


Figure 4.5: Evaluation of the Root-Mean-Square Error for the algorithm based on spectral decomposition

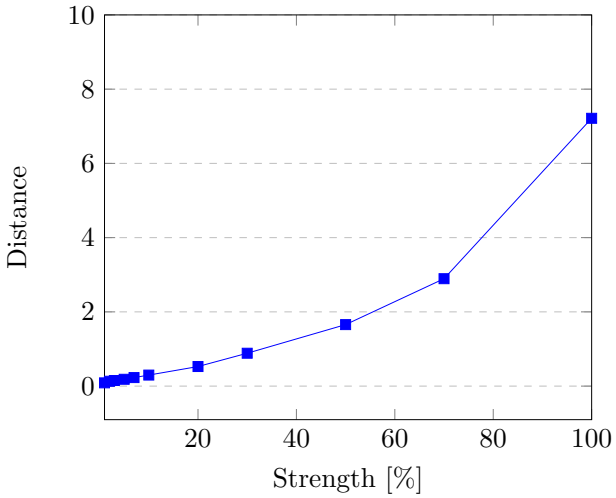


Figure 4.6: Evaluation of the Hausdorff distance for the algorithm based on spectral decomposition

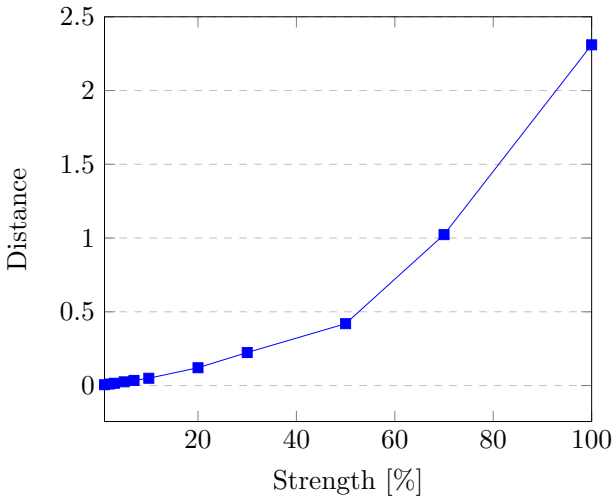


Figure 4.7: Evaluation of the Local Smoothness metric for the algorithm based on spectral decomposition

4.2.2 Capacity

During the insertion of the watermark bits, we assign one bit to many frequencies. All frequencies are paired with a bit but not all frequencies will host this corresponding bit because not all frequencies are modified. This means that we can only count how many times a bit has been inserted in a patch, but we cannot guarantee that all the bits have been inserted in all patches. Figure 4.8 shows the number of bits that have been inserted. As expected, all possible patches decompositions have similar capacities. The decomposition in patches does not have any impact on the capacity of the watermarking scheme.

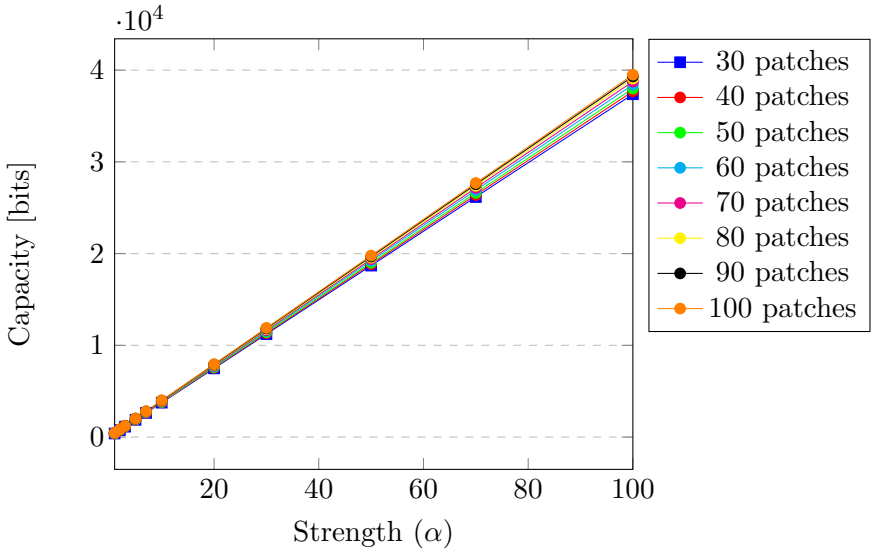


Figure 4.8: Capacity of the watermark depending on the number of patches and the strength

4.2.3 Robustness

As we stated in Section 4.2.2, we cannot ensure that all bits have been inserted in all patches. The bit error rate of the unattacked watermarked bit is even more representative than previously. The number of patches has virtually no impact on the robustness of the watermark since the

total number of modified frequencies is the same in all cases.

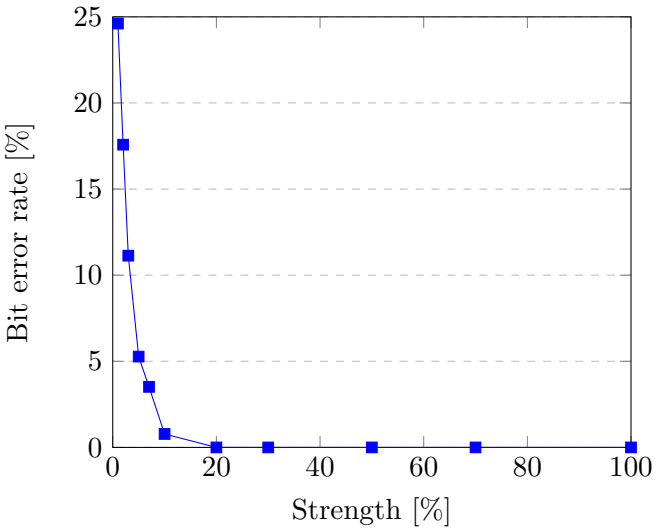


Figure 4.9: Bit error rate of the watermarking algorithm based on spectral decomposition

As we can see from Figure 4.9, even with a strength as small as 1%, the bit error rate is smaller than 25% and gets really close to zero with a strength of 10%. It is important to note that contrary to the previous algorithm scheme, the bit error rate is here based on the full 64-bits signature that we wanted to insert since we cannot be sure of which bits have been inserted.

The behavior of this algorithm regarding the smoothing attack (Figure 4.10) is quite different from the one regarding the random noise addition (Figure 4.11). The first one completely obliterates the watermark after only two to three iterations while the latter has a really small impact on the retrieved bits, even after a significant noise addition of 0.1 times the coordinates of the vertices.

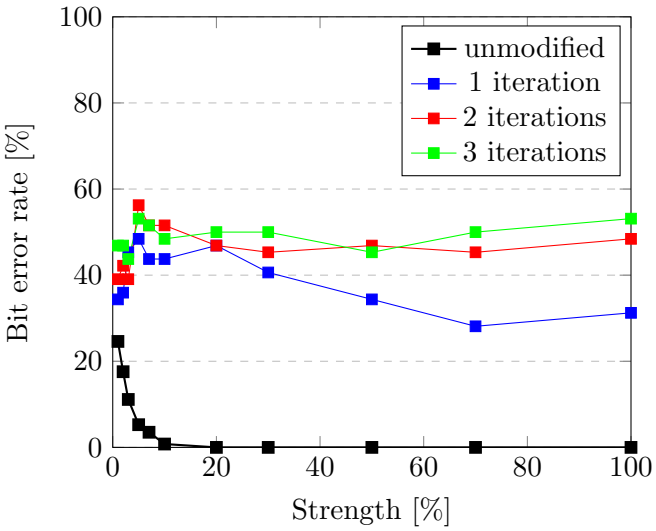


Figure 4.10: Robustness of the algorithm based on spectral decomposition against a smoothing attack

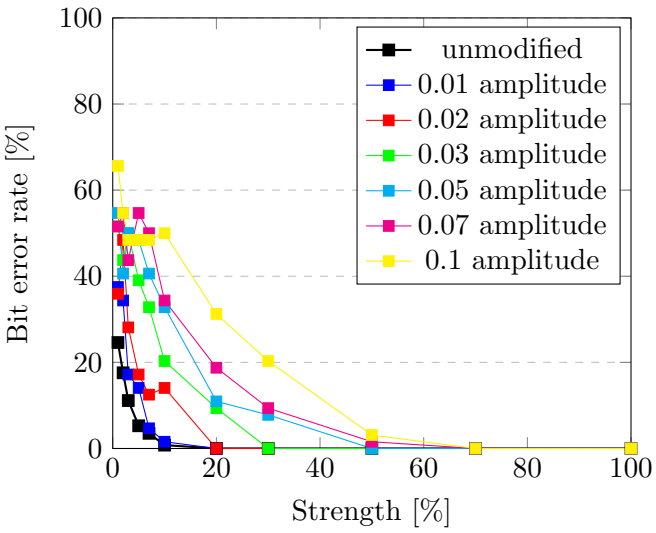


Figure 4.11: Robustness of the algorithm based on spectral decomposition against random noise addition

5 Adapting Spectral Decompositon to Additive Manufacturing

5.1 Principles

The spectral decomposition technique introduced in Section 4 cannot be applied as such to printed models. The main problem comes from the fact that the patch decomposition is performed using a tool called METIS. When printing and scanning a model, the model is completely re-meshed and therefore, the patches returned by METIS, which depend on the structure of the mesh, will be completely different from the ones that were used during the insertion of the watermark, resulting in a reordering of the retrieved bits, making the watermark completely unreadable.

The solution to this problem is to perform a patch decomposition that won't depend on the connectivity but rather on the geometry of the mesh. To do so, we tried to implement a technique that creates patches from layers of an object. This way, if the model is oriented in the same way during the embedding and the retrieval of the bits, the patches will be exactly the same and the bits will be correctly read.

5.1.1 Orientation

The goal is to make sure that we get the exact same patches during the insertion and extraction processes in order to retrieve the same bits that were inserted. Since we want to create layers from a model, we only need one direction to match between the two procedures, which is the direction that is perpendicular to the created cuts. The orientation problem is thus composed of two parts:

Orienting the mesh before the embedding The orientation of the mesh before the printing needs to be performed in such a way that the z-axis of the model will match the printing direction. The printing direction can be selected based on multiple parameters that are discussed by Kulkarni *et al.* [9]. Those parameters include the area of the base, the area in contact with supports, the total height of the object or even the volume of supports used. To as permissive as possible, the choice of the printing direction is not performed automatically and will be given as argument to the algorithm.

We want to create a rotation matrix to get from the direction given as argument to the $[0, 0, 1]$ direction, corresponding to the z-axis. Similarly, this can also be performed as a 2D-rotation to align the given axis with the plane formed by the x and z-axes, then a second 2D-rotation to align the obtained vector to the z-axis.

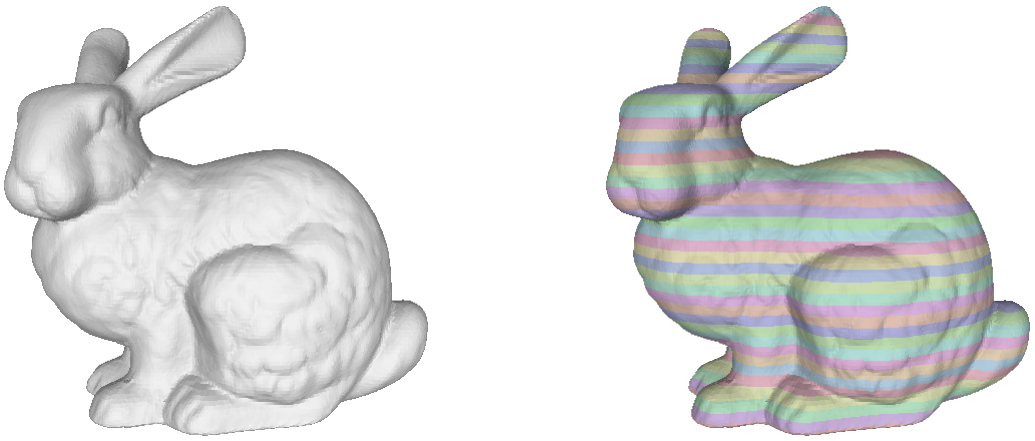
Determining the way that the model was printed after it has been scanned The orientation of the scanned model needs to match the one from the original model in order to retrieve the same patches. We then have to retrieve only the printing axis. As described in Section 2.5, additive manufacturing introduces a *staircase effect* to printed models. We'll use this effect to determine in which direction the object was printed. There are two possible orientations for the normals of the faces of the resulting object:

- **Aligned with the printing direction:** the horizontal part of the created slices will have a normal that matches the direction of printing.
- **Perpendicular to the printing direction:** the rest of the faces composing the object will be pointing in different directions. However, they will all belong to parallel planes whose normals will match the printing direction.

In order to find the printing direction, we therefore need to sort those different normals. To do so, a possible option would be to compare the different normals to one another by forming pairs. If the two vectors of a pair are perpendicular or parallel, we cannot deduce anything from this pair. On the other hand, if the angle they form is different from $\alpha \times \frac{\pi}{2}$ with $\alpha \in \mathbb{N}$, this means that the two normals both belong to vertical faces to the set of vectors perpendicular to the printing axis, and they both can be removed from the original set of vectors. All the normals that have not been filtered out at the end of the process should all be aligned in the same direction, the printing direction.

5.1.2 Layers Decomposition

Once the mesh is oriented in the right direction, we can start splitting it into layers as represented in Figure 5.1. Since the printing direction is now aligned to the z axis, the partitioning will only depend on the z values. The total range of z values allows us to determine the interval that we'll use to cut our piece. The full process is described in Algorithm 5.1.



(a) Original model (b) Patches created

Figure 5.1: Visual representation of the goal of layers decomposition

When a cut goes through a face, the latter should be divided into three smaller faces. This subdivision is represented on Figure 5.2. It consists in computing the intersection between the plane $z = cut\_value$ and the two straight lines going from one of the vertices to the two vertices on the other side of the cut plane. To make the computation of the eigenvalues needed for the spectral decomposition more efficient, we need to limit the number of vertices. To do so, instead of copying all vertices from the base mesh to its partitions, we copy only the used ones. This means that the created faces must use the updated indexes of the vertices.

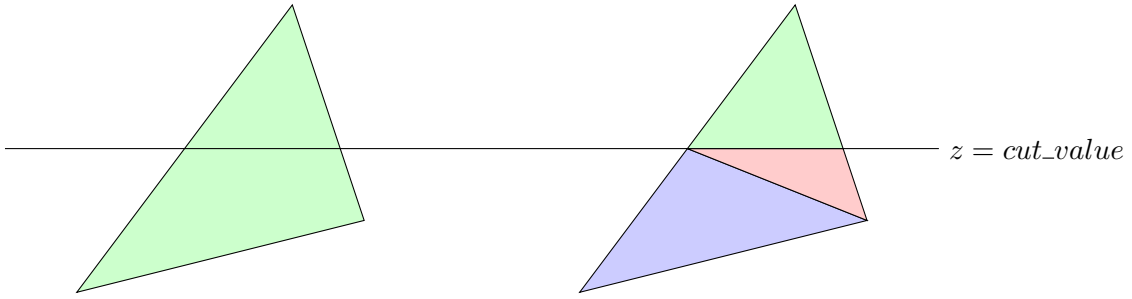


Figure 5.2: Process of cutting a face based on its z value

Algorithm 5.1: Layers Decomposition

Data: The mesh to partition & N, the number of partitions wanted
Result: A list of N submeshes

$interval = \frac{max(z)-min(z)}{N}$
$cut\_value = min(z) + interval$
while $cut\_value \leq max(z)$ **do**
    for $face \in remaining\_faces$ **do**
        if *All vertices are located under than the cut_value* **then**
            Add the face to the current patch and remove it from the remaining_faces
        else if *Two of three vertices are located under the cut_value* **then**
            Divide the triangle into three smaller triangles
            Two of them belong to the current patch, the third one is added to the remaining_faces
        else if *One of three vertices is located under the cut_value* **then**
            Divide the triangle in three smaller triangles
            One of them belongs to the current patch, the two other ones are added to the remaining_faces
        end
    end
    $cut\_value = cut\_value + interval$
end

5.1.3 Disconnected Parts Splitting

For a spectral decomposition to be performed, all vertices must belong to a single part. Partitioning meshes based on a single direction can result into meshes that are composed of different parts, as represented on Figure 5.3

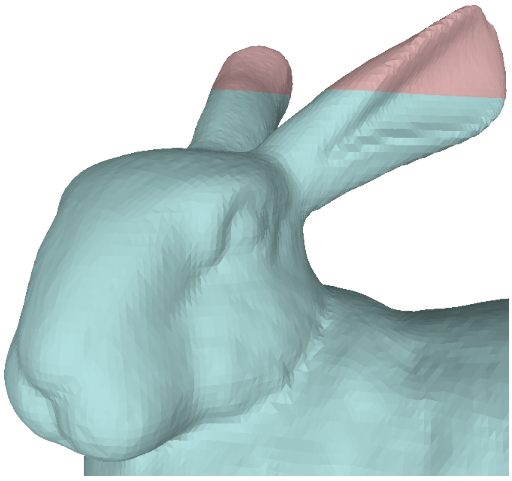


Figure 5.3: The patch formed by the two parts of ears of the bunny should be split into two different sub-meshes

The disconnected patches in a single mesh must be divided into multiples sub-meshes. This is done by considering the connectivity of a mesh as a graph and performing an algorithm similar to a Breadth First Search or a Depth First Search. The vertices are the nodes of the graph and each face leads to three edges in the constructed graph. By choosing randomly a first vertex from the mesh, we can travel to all the nodes that are connected to this first one, marking them along the way. All the marked vertices form the first patch and the process can, therefore, be repeated onto the remaining vertices until there are no vertices left.

5.1.4 Spectral Decomposition

The process of the spectral decomposition is very similar to what has been done previously. We still faced a difficulty: modifications applied to the R spectra, which would lead to a modification of the z coordinates, might introduce important deformations that would make a vertex move from one layer to another. We found two possible solutions to avoid that:

- Do not consider frequencies when the R value is C_{inter} during the embedding and retrieval processes. This means that we remove one-third of the frequencies that could host a watermark bit, and we then have less redundancy.
- If we cannot modify the value of C_{inter} relatively to the mean of C_{max} and C_{min} as represented in Figure 4.2, the other option is to rather modify C_{max} and C_{min} in order to move their mean so that C_{inter} is located in the correct interval as shown of Figure 5.4. Note that, here, we modify two values instead of one, making the changes more perceptible.

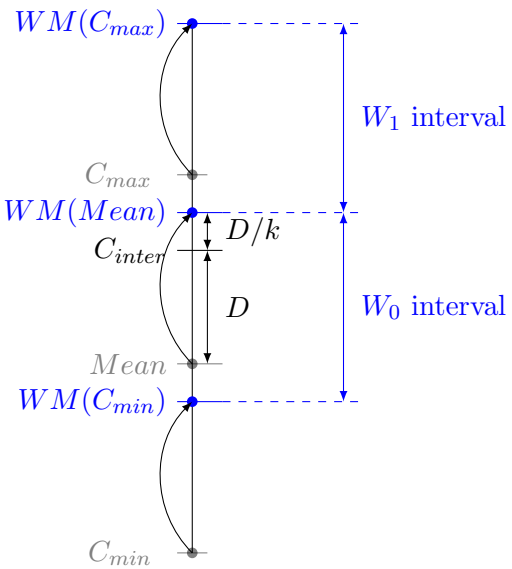


Figure 5.4: Inserting a 0 when C_{inter} in W_1 and we do not want to modify C_{inter}

Here, we implemented the first option because we figured that the models we would use would have an important number of vertices so the redundancy would not be a problem. However, limiting the imperceptibility is a nice element to consider.

5.1.5 Avoiding Discontinuities

When splitting the original mesh into layers we created multiple vertices that have the exact same coordinates but belong to different patches. This can lead to discontinuities because the vertices from either side of the cut will move independently from each other. To avoid this, we

decided to completely remove the vertices and the faces on the border from the embedding and retrieval process. This implies that those faces need to be removed from the computation of the Laplacian and the vertices must be excluded from the bits insertion and extraction. Since we have simple layers, vertices on the border of patches can be retrieved easily because their z coordinate is equal to the cut value.

Of course, the removed vertices and faces need to be reinserted when creating the final watermarked patch. To do so, we created a mapping between the indexes of the vertices in the original patch and in the watermarked one. Values that do not have a mapping are considered being located on the border of a patch and should therefore not be modified.

5.1.6 Avoiding perceptible modifications

After all of this, some patches were still heavily modified as shown in Figure 5.5. We tried to implement different techniques to overcome this problem.

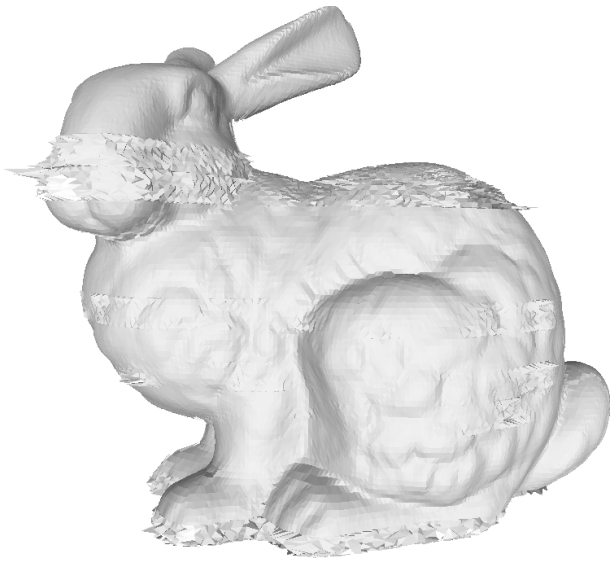


Figure 5.5: Watermarked bunny without solutions detailed in 5.1.6

Divide the range in more than two intervals Applying a greater modification to one of the values of the spectra might introduce significant modifications to the whole set of patches. A simple solution would be to the range of C_{min} to C_{max} into more than the two intervals corresponding to the bit values 0 or 1. Using four intervals would divide the average modification by two, also decreasing the perceptibility of the embedded bit.

The embedding process is similar to what has been done previously, the only difference being that the values around which C_{inter} is inverted are not the mean of C_{min} and C_{max} but rather the mean of either C_{min} or C_{max} and the previously computed mean as represented on Figure 5.6. Of course, we could use more than four intervals but this could also introduce more errors when retrieving the embedded bits due to imprecision errors.

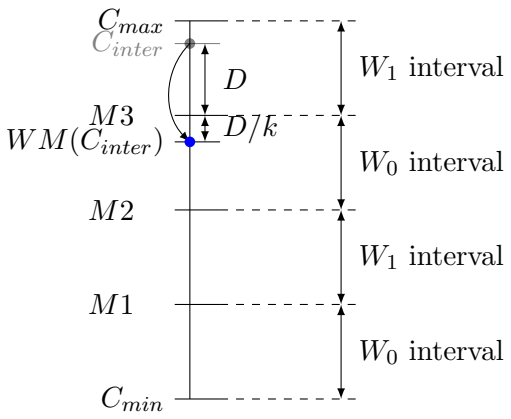


Figure 5.6: Inserting a 0 when C_{inter} is in W_1 using four intervals

Discard patches with important modifications Some patches that have sharp edges might introduce important modifications no matter how many frequencies we modify. We refer to those patches as "unwatermarkable". To detect which patches are unwatermarkable the only option is to process those patches as any other, compute the introduced distortions using algorithms such as those presented in 2.4.3 and restore the different spectrum to their previous values if the metric is too high. A problem appears for the retrieval part of the algorithm. It is impossible to detect which patches have been discarded during the embedding of the data. Therefore, the data retrieved from some patches could be completely inaccurate. However, the number of discarded patches should be quite small relative to the number of total patches so the redundancy introduced in other patches should be important enough to correct the faulty values.

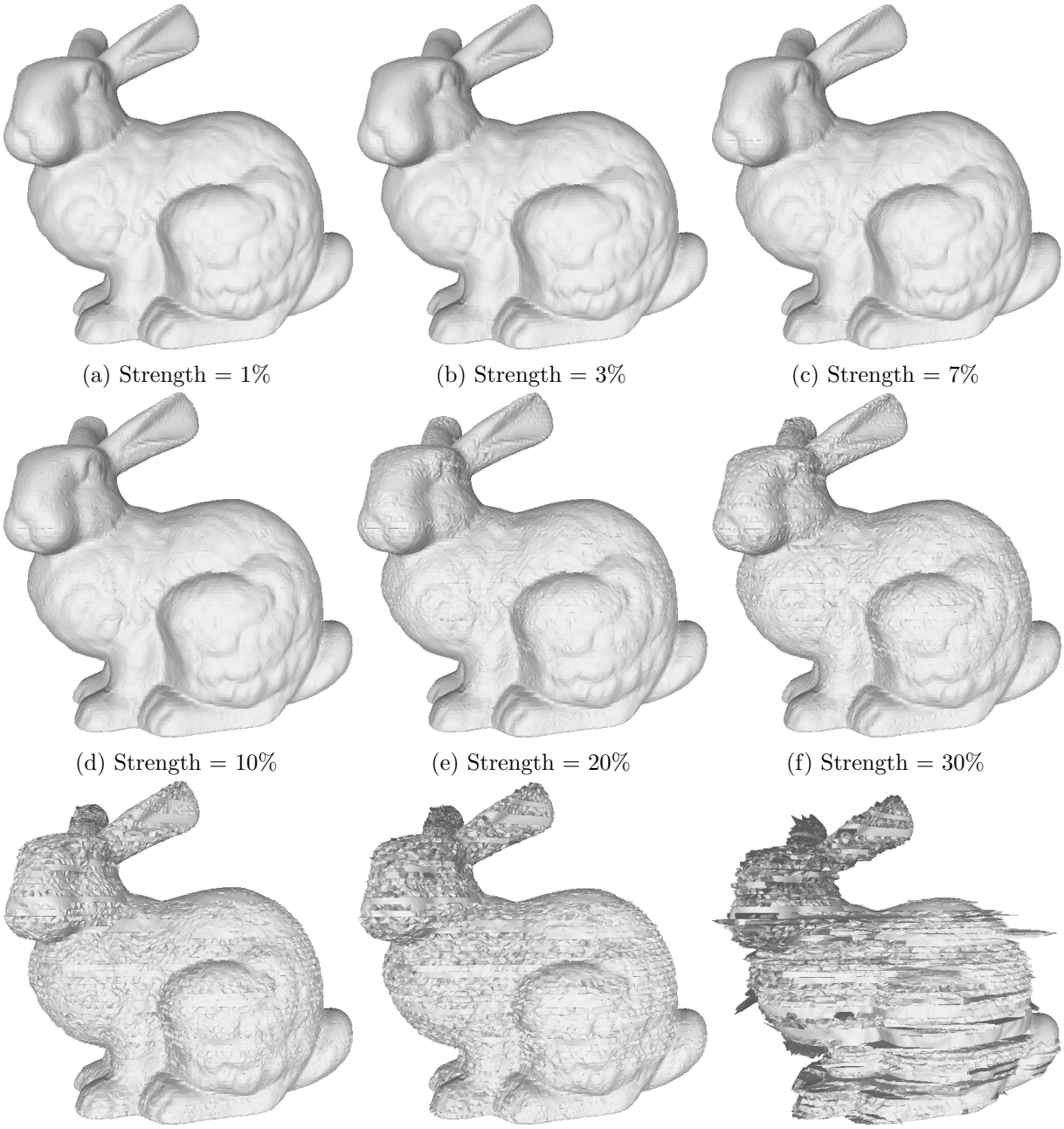
Choose frequencies to modify It appears that ordering the eigenvalues was not sufficient to ensure that the corresponding spectrum values were also sorted in decreasing order. This might be due to the shape of the created patches. Therefore, instead of discarding the N first frequencies, we need to select them based on their values. By keeping them ordered regarding their corresponding eigenvalues, we can make sure that the bits will be retrieved in the same order as they were inserted. A small problem comes from the fact that, by modifying the values of the spectra, some values that were considered during the embedding process might become larger than the ones that were discarded and therefore, won't be selected during the retrieval process, resulting in some errors. Again, redundancy was important enough to ensure that the bit error rate was small enough.

5.2 Evaluation

As for the two previous algorithms, we will evaluate the performances of this third watermarking scheme. The results should be pretty much similar to the algorithm based on spectral decomposition with regular patches.

5.2.1 Imperceptibility

Visually, the introduced transformations follow the same pattern regarding the strength of the watermark. Two notable difference are present in Figure 5.7. The first one is the orientation of the deformations: the coordinates of the vertices are only moved in the same yz-plane because we don't want a vertex to go from one layer to another. The second difference is the unmodified layers when the strength gets above 50%, some layers undergo too many distortions and their modifications are completely discarded.



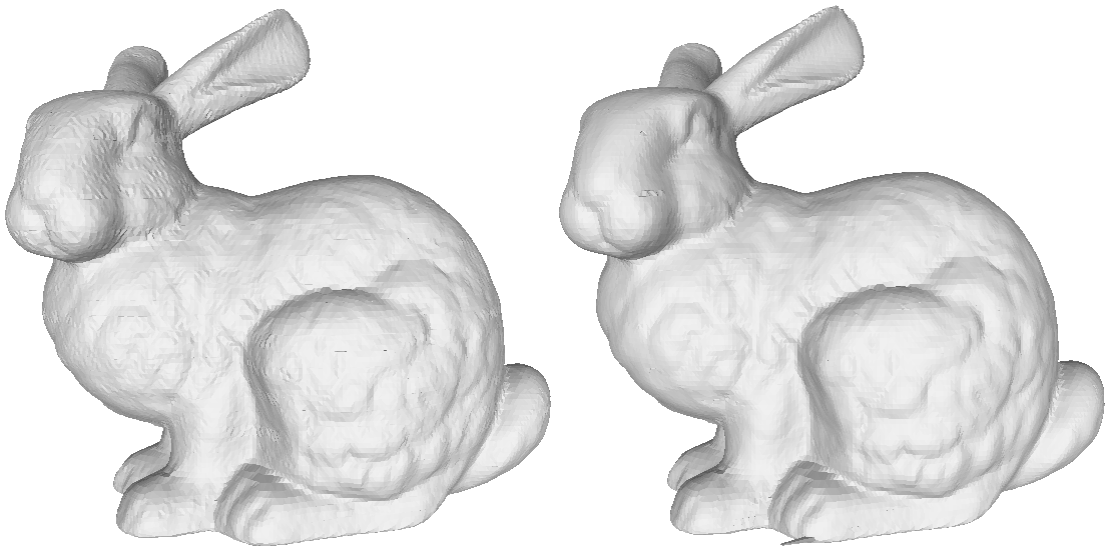
(a) Strength = 1% (b) Strength = 3% (c) Strength = 7%

(d) Strength = 10% (e) Strength = 20% (f) Strength = 30%

(g) Strength = 50% (h) Strength = 70% (i) Strength = 100%

Figure 5.7: Varying the strength of the watermark divided into 60 layers

Increasing the number of layers has a bit more influence on the perceptibility of the watermark than with patches. Figure 5.8 shows that the more layers you create, the fewer modifications are introduced. It is confirmed by the measurements detailed in Appendix C.1. This is due to the fact that we do not consider the vertices located on the border of layers during the selections of vertices to modify. Increasing the number of layers increases the number of cuts and therefore the number of vertices located on cuts. Selecting 10% of fewer vertices imply fewer frequencies modified but also modifications in higher frequencies. Perceptible modifications are thus gathered in smaller areas but are of greater magnitude, as can be seen on the paw of the bunny model.



(a) 40 Layers (b) 100 Layers

Figure 5.8: Varying the number of layers with a fixed strength of 10%

The shape of the curves of the Root-Mean-Square error (Figure 5.9) and the Hausdorff Distance (Figure 5.10) are similar to the previous algorithm with patches. However, the values of those metrics for smaller strengths are a bit bigger than previously, mostly because of the vertices not considered for the insertion of the signature bits.

The metric representing the local smoothness (Figure 5.11) grows linearly in opposition to the other metrics and the other option for patch decomposition. This is probably due to the layers that are discarded due to their important deformations.

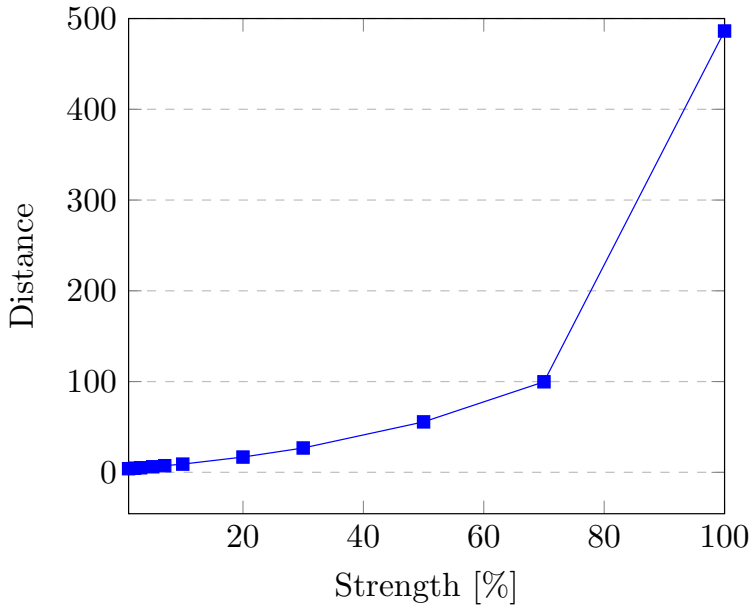


Figure 5.9: Evaluation of the Root-Mean-Square Error for the algorithm based on spectral decomposition with a patch decomposition in layers

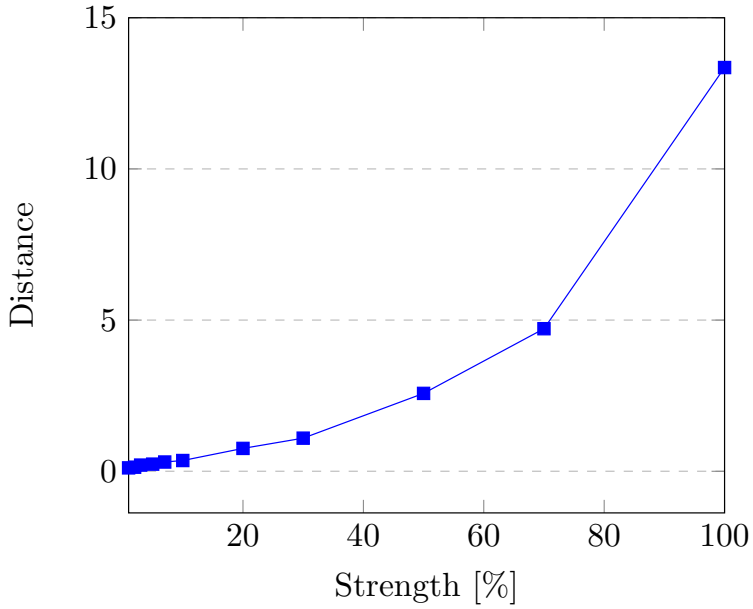


Figure 5.10: Evaluation of the Hausdorff Distance for the algorithm based on spectral decomposition with a patch decomposition in layers

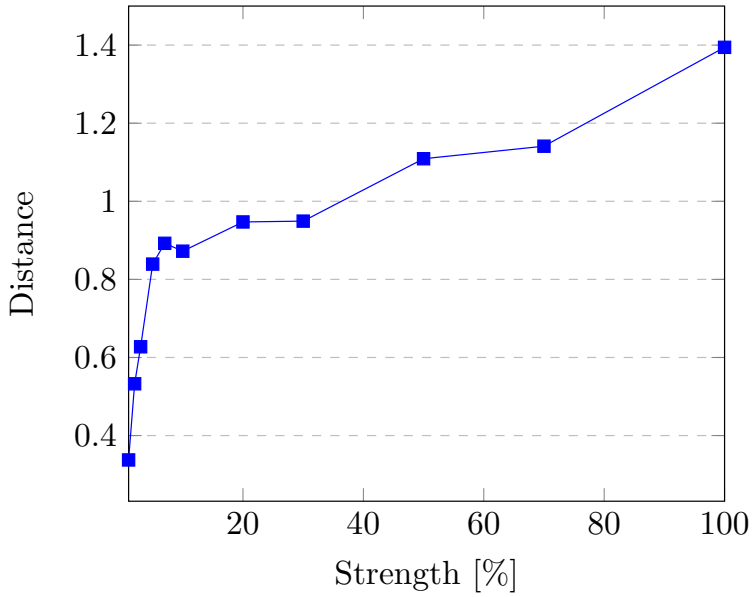


Figure 5.11: Evaluation of the Local Smoothness metric for the algorithm based on spectral decomposition with a patch decomposition in layers

5.2.2 Capacity

As in Section 4.2.2, we've evaluated the capacity depending on the number of layers. Again, we cannot ensure that all bits have been inserted. The lines represented in Figure 5.12, show the number of bits that can be inserted into the whole mesh. The resulting values are inferior to the number of bits when using classical patches because of the ignored vertices on the border of patches. The more layers we create, the more cut we create in the model and the more border as located on a border. This is why the capacity decreases when we create more layers.

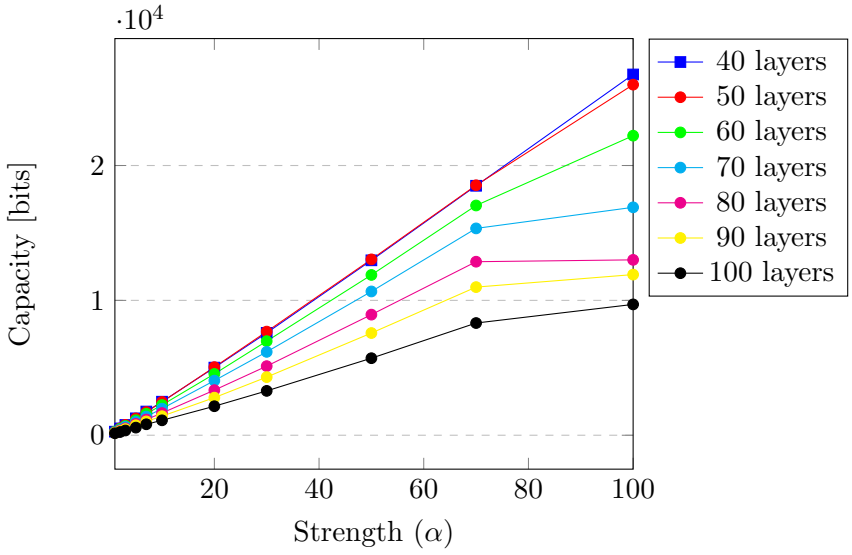


Figure 5.12: Capacity of the watermark depending on the number of layers and the strength

5.2.3 Robustness

Finally, let us assess the performances of this new technique in terms of robustness. The bit error rate of the retrieval performed on a non-attacked mesh is rather small, the maximum error rate being 20% with a strength of only 1%. As we already explained, augmenting the number of layers decreases the usable vertices and therefore restraining the possibilities of introducing redundancy. However, with a smaller number of layers, we obtain really promising results. For example, with 40 layers, a strength of only 2% is enough to get a nil bit error rate.

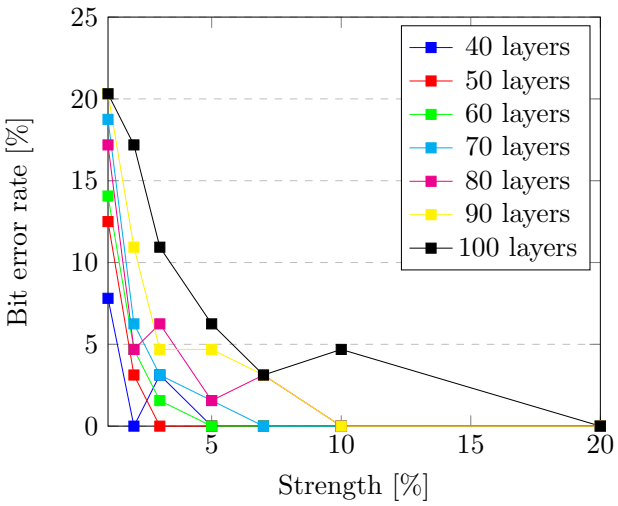


Figure 5.13: Bit error rate of the watermarking algorithm based on spectral decomposition with a patch decomposition in layers

The robustness against smoothing attacks and random noise additions is quite a problem here. Partitioning of the model into layers is performed based on the z values. The maximum and minimum z values are used to compute the interval of accepted vertices into a specific patch. Adding noise or smoothing might move some vertices from one layer to another, shifting the order of the retrieved bits in both patches and making the watermark unreadable. If one of the vertices corresponding to the minimum or maximum z value is moved, it could even change the intervals range and therefore completely modify all layers resulting in a lost signature.

This is what we observe in Figure 5.11. After a single iteration, the bit error rate increases to 50%. In the case of random noise addition, we could not even compute the bit error rate. The unaligned layers division resulted into thousands of disconnected patches, multiplying the time needed for the retrieval by an important factor and therefore making the computation practically unfeasible.

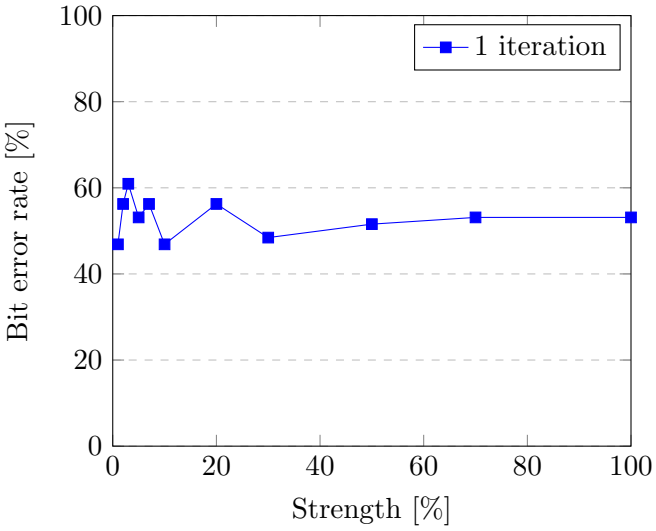


Figure 5.14: Robustness of the watermarking algorithm based on spectral decomposition with a patch decomposition in layers against a smoothing attack

6 Comparison of the Algorithms

We presented, explained and reviewed three different watermarking schemes. We now know their strengths and weaknesses but how do they compare to each other? After a short study of the time needed to perform the embedding and retrieval, we will see how they differ in terms of capacity. Finally, we will close this comparative study with the performances of the different algorithms concerning the trade-off between robustness and imperceptibility, which is the main objective of this work.

6.1 Time

The only constraint we have about time is that the embedding and retrieval process should be performed in a reasonable amount of time. In none of the presented algorithms is the strength an influencing factor for the time needed to perform the operations. The subdivision into patches or layers, however, seems to have a more important impact on the algorithms performances.

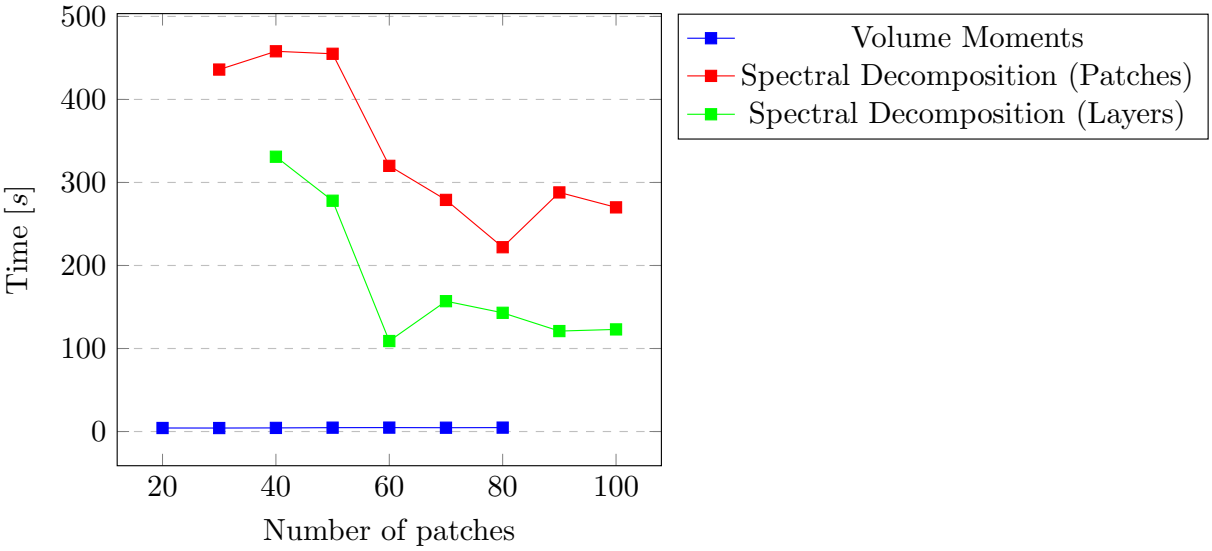


Figure 6.1: Performances of the three presented algorithms for the watermark embedding process

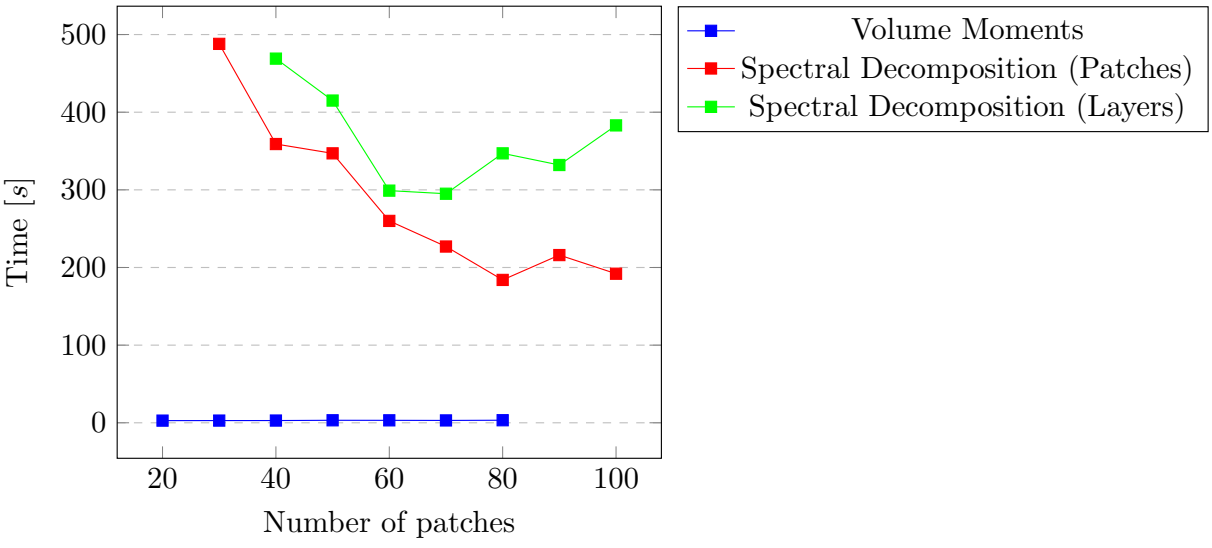


Figure 6.2: Performances of the three presented algorithms for the watermark retrieval process

Figure 6.1 shows the time needed for the embedding of a watermark. It's clear that the algorithm based on volume moments is the fastest and executes in only a few seconds. The other two algorithms take a lot more time but still manage to stay between five and ten minutes, which is acceptable. The pattern is really similar in Figure 6.2 which focuses on the retrieval of the watermark. Again, the algorithm based on volume moments is the fastest and the other ones take five to ten minutes to execute. It is interesting to note that, although the decomposition into layers seems faster than the one in patches for the insertion of the bits, their extraction is the other way around. This can be explained by the number of vertices. During the decomposition into layers, we create a lot more vertices than were initially present in the model. Those vertices also need to be taken into account during the splitting of the retrieval.

6.2 Capacity

It is difficult to compare the three algorithms in terms of capacity, even though the three algorithms return the number of bits that can be inserted in the mesh. The algorithm based on volume moments doesn't need redundancy, while the two algorithms based on spectral decomposition both need to insert multiple times the same bit. The numbers we get from our tests are, therefore, not really representative since it does not tell how many bits we will retrieve. As we already exposed, even with a capacity of thousands of bits, we cannot mathematically ensure that all the bits have been inserted at least once.

Less formally, we can tell that, using the scheme based on volume moments, we could not insert a typical watermark of a length of 64 bits during our tests, while the algorithms using spectral decomposition inserted 64 bits while ensuring enough redundancy with a strength as small as 10%. Therefore, globally, it seems that the volume moments-based algorithm is the one with the smallest capacity. Concerning the two remaining algorithms, the decomposition into layers removes a lot of vertices from the process of insertion of the watermark, it, therefore, has less redundancy than the algorithm using a classical patch decomposition.

6.3 Robustness and Imperceptibility Trade-off

Comparing the robustness and imperceptibility independently would not make much sense here since increasing the robustness of an algorithm would result in decreasing its imperceptibility and inversely. We will, therefore, compare the relation between imperceptibility and robustness. Figure 6.3 shows these ratios for our three different watermarking schemes.

It appears that the algorithm based on volume moments favors imperceptibility over robustness. Some bit error rates even go to nearly 90%. Choosing all the bits at random would give a bit error rate of around 50% so how is that possible? During the test, the inserted watermark was a 64 bits long signature only composed of alternating zeros and ones ([0, 1, 0, 1, 0, 1, ...]). The error probably comes from the skipping of a bit which shifted the rest of the watermark. The bits were then read correctly but their reordering was problematic. The Hausdorff Distance, on the other side, does not get over 0.15.

Some parameters still gave a bit error rate close or equal to zero for really small Hausdorff distances. It seems that the number of patches and the way that the model was patched (n_h, n_θ) has a considerable impact on the performances of the watermark inserted using volume moments.

The spectrum-based algorithms seem to be more permissive by letting the user choose between robustness and imperceptibility. Some parameters also allow the model to have a bit error rate and a Hausdorff distance both close to zero.

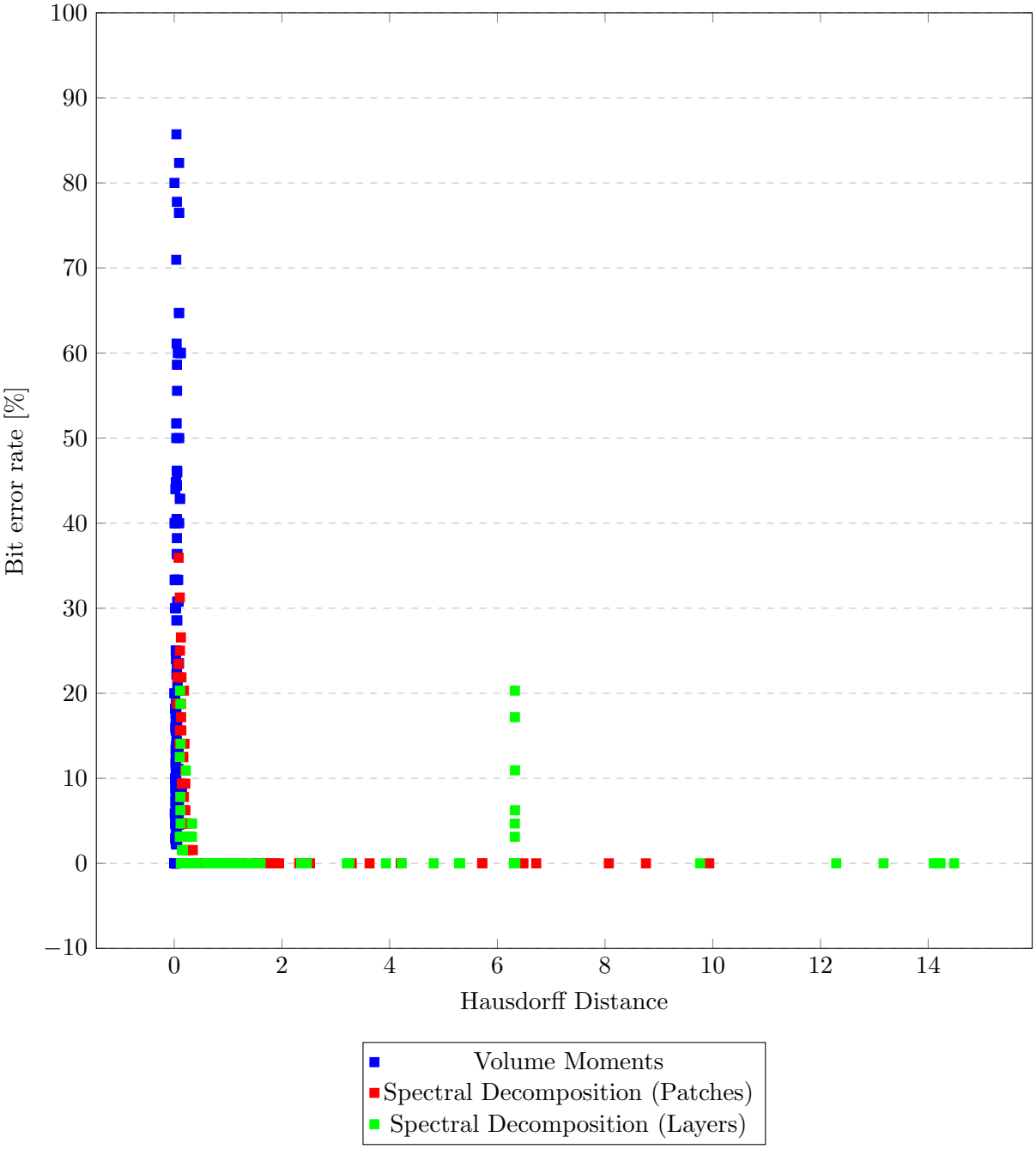


Figure 6.3: Comparison of the three algorithms in terms of imperceptibility and robustness, varying their strength and number of patches

Some strange values appear for the layered patch decomposition. They have an important bit error rate and an important Hausdorff distance. Those values correspond to the models that have been split into 100 layers. The significant bit error rates come from the patches that have been discarded because of the important deformations, which represent an important part of all the patches. During the retrieval process, it is impossible to discern the patches that have been discarded or not. The only option we have is to read the data from all the patches and, if the redundancy of the data in non-discarded patches is important enough, the bit error rate should remain quite low. In this specific case, too many patches have been discarded to ensure a low bit error rate.

7 Software Implementation

This work is also composed of a practical software development aspect. The majority of the code used for the evaluation of the method based on volume moments has been written by Gaëtan Cassiers [2] during a two months long internship at the Massachusetts Institute of Technology and is currently stored on the Center for Bits and Atoms' GitLab. I only contributed to his code by fixing a few bugs that prevented the correct operation of the algorithm and by adding some new features :

- The algorithm could not be applied to certain models due to an error in the hiding process.
- The bit error rate was quite significant even when no attack was performed on the model, this was coming from a rounding problem.
- The number of patches used to run the tests could not be modified.
- I added a new function that allows testing the algorithm without performing an attack on the watermarked model.

For the spectral decomposition-based process, I was inspired by the code written by Patrice Rondão Alface during his thesis at Université Catholique de Louvain. Even though his code was written in C++, I have chosen to adapt it and write it in Python 2.7, which is the same language that was used in the development of the volume moments algorithm. This way, comparing the different algorithms has been much simpler and could be performed using a single script. This code is publicly available on GitHub: `https://github.com/aprieels/3D-watermarking-spectral-decomposition`.

7.1 Structure of the code

The two variants of the algorithm based on spectral decomposition are both located in different directories (`Original` and `Layered`) on the same GitHub repository. The main structure of the codes of both algorithms are really similar:

- `spectral_decomposition.py`: this file contains the main methods used to insert or extract a watermark.
- `partitioning.py`: the functions needed to perform the patch decomposition, layers decomposition, disconnected mesh splitting or multiple meshes merging is located in this second file.
- `embedding.py`: given a patch and a watermark, this function returns a modified version of the patch, containing the watermark.
- `retrieval.py`: the retrieval of the watermark's bits is done in this file.
- `utils.py`: this file contains some utility functions that could be used by multiple functions in different files such as the computation of the lists of neighbors of a patch's vertices.
- `distortion.py`: the computation of the imperceptibility metrics such as the RMS Error, the Hausdorff distance, and the Local Smoothness is performed here. s
- `attacks.py`: this last file contains all the function regarding the tests of the robustness of the models, including the smoothing attack and the random noise addition.

The main difference between the structure of the two codes is the `orientation.py` file. When using a classical patch decomposition, the orientation of the mesh has no impact on the created patches. This is not the case when cutting the mesh into layers. In this case, The model should be oriented so that the printing direction matches the z-axis, which is what is performed inside the `orientation.py` file.

7.2 Libraries and tools used

7.2.1 NumPy

NumPy is defined by its creators as "*the fundamental package for scientific computing with Python*" [5]. It basically allows MATLAB-like computations in Python. NumPy defines a new type of arrays (`numpy.ndarray`) that allows easy matrix manipulation, such as multiplication, addition, eigenvalues and eigenvectors computation...

7.2.2 PyMesh

PyMesh [11] is an open source library created in order to help developers deal with meshes. There exist a lot of different file formats, but they all implement variations of the same structure of meshes which is: a geometry and a connectivity. PyMesh deals with all the different file formats and returns a unique data structure, making our algorithm compatible with all the most commonly used file formats, removing the complexity of having to parse lots of different patterns. The data structure defined by this library has the advantage to be completely compatible with the array structure used by the NumPy library.

We only used PyMesh for its data processing functions :

- `load_mesh`: this function takes a file name as argument and creates a common Mesh data structure, no matter what format the file uses.
- `form_mesh`: this function creates a Mesh data structure from a list of vertices and faces. It is really useful when needing to modify the geometry or the connectivity of a mesh.
- `save_mesh`: finally, we need to store the modified mesh. This last function saves a Mesh data structure to a file while respecting the pattern used by the needed format.

The build and installation of PyMesh was quite a mess. The package is not available on typical python's packages managers such as `pip` and therefore needs to be built locally. It has a lot of dependencies that also need to be built, making the whole process quite complicated. I have included a built version of PyMesh in the code available on GitHub to make it easier to run. A simple "`./setup.py install`" command is enough to run it locally.

7.2.3 SciPy

SciPy is somehow similar to what NumPy does, except that it does not focus on arrays computation. We only used it for the computation of the Hausdorff distance since it has a predefined data type, called "`spatial.KDTree`", optimized to find the closest neighbors.

7.2.4 METIS

The partitioning of meshes in Section 4 was performed using a tool called METIS [7]. This tool provides a fast and quality mesh partitioning based on a file stored as "mesh" format and a number of wanted partitions. Unfortunately, it does not exist as a python library, so we had to run it as a command line in bash that then creates a file containing a mapping of each vertex to a partition number.

7.2.5 MeshLab

Finally, we needed to have a visual representation of the watermarked models in order to determine which deformations were imperceptible. Meshlab [4] is an open source system for processing and editing 3D triangular meshes. It offers some features that turned out to be really useful in

our case. Of course, the principal functionality that we needed was to see the mesh. Meshlab supports very large meshes as efficiently as smaller ones. It also allows to see different meshes in different colors or even to change the color of the meshes based on per face or per-vertex attributes. This was quite useful to test our layers partitioning. Finally, it offers a way to align multiple meshes based on selected particular points.

7.3 Tests

Automated testing was not implemented here for the two following reasons :

1. It can happen that there are some errors during the retrieval, it will depend on the model itself, the watermark to be embedded, the strength of the watermark, ...
2. It is not possible to determine if a watermark is "too perceptible" automatically.

Therefore, the results of the tests that are implemented need to be checked by the user. We have tests for the orientation of the model, the partitioning of the mesh and the whole process of embedding, retrieval, and comparison of the data inserted and retrieved.

We then created a script to perform redundant tasks needed to perform measurements of the algorithms on a batch of different parameters values such as the number of patches we want to create during the decomposition of the model as well as the strength of the watermark. For a lot of different values, this script performs an insertion of a watermark, its extraction, the comparison of the data inserted and retrieved as well as the computation of the different imperceptibility metrics.

The tests for the robustness of the watermark against smoothing attacks and random noise addition is performed in the last script that performs attacks using a different number of iterations or amplitude of different watermark strengths.

8 Conclusion and Future Work

The purpose of this thesis was to find a way to apply a watermarking technique that would be robust against a process of three-dimensional printing and scanning. The main difficulty of such a constraint is that the connectivity of the mesh might completely change and that its geometry might undergo a certain level of inaccuracy. We, therefore, need a process that is based on the geometry of the mesh but that also supports some inaccuracies. We proposed three alternatives to solve this problem and compared them in terms of robustness, imperceptibility, capacity and time.

The first watermarking technique we reviewed is based on volume moments. The basic idea behind this algorithm is that a good watermarking scheme should be intrinsically linked to the 3D shape behind the mesh. In other words, to make the watermark less visible, you should first insert the watermark in areas that are already rough and that will endure much more important modifications while keeping them unnoticed.

After being normalized, the model is divided into patches which are then sorted into three categories regarding their volume moments: discarded, cover and compensation. Discarded patches are either too small or too flat to accept deformations. Cover patches will be the ones that host the watermark bits. This insertion is performed by pushing the moment value towards a quantized value. Last, compensation patches, which are the ones that will accept the most modifications, will be deformed in order to retrieve the same center of gravity and principal axes than before the embedding so that the normalization of the model during the extraction of the watermark returns the same patches as during the embedding.

We then decided to compare this first algorithm to another one, based on a completely different principle. This second algorithm inserts the data bits into the spectral coefficients of the model. By working in the spectral domain, the frequencies that we modify are directly related to the visual deformations that are introduced in the model. Modifications of lower frequencies will result in important, localized visual deformations contrary to higher frequencies that will result in smooth global changes.

The coordinates along the three axes will be transformed into their spectral correspondents. Those three spectral coefficients are ordered from the smallest to the largest. We then change the values of the intermediate coefficient relative to the mean of the two other ones. If it is greater than the mean, the value that was inserted is a 1, otherwise, it is a 0.

A problem with this technique appeared quite rapidly. The patch decomposition we used in this technique is performed using a tool called METIS. This tool creates patches based on the connectivity of the mesh. But, we know that the probability of retrieving the exact same connectivity after a print-scan process is nigh zero. The retrieved patches would thus be completely different during the insertion and extraction steps.

Patch decomposition is a recurrent problem in digital watermarking of three-dimensional models. Retrieving the same patches before and after the print-scan of a model is complicated. The main problem comes from the fact that the whole model might be completely re-meshed after this kind of procedure, making all techniques based on connectivity useless.

The partitioning of meshes into layers seems to be a promising approach, creating patches whose shapes are absolutely not dependent on the connectivity of the mesh itself but rather on its geometry. Of course, there are still some elements that could be improved.

As stated in Section 5.1.1, the main advantage is that we only need to retrieve a single direction after the scanning of the model, the direction perpendicular to the created layers. By choosing this direction wisely during the insertion of the watermark's bits, the retrieval of this direction can be pretty straightforward. For example, here, we chose the printing direction. We can then use the slicing constraint described in Section 2.5 to our advantage to make this process easier by using the normals of the faces.

The idea of completely removing the vertices located on the border of patches, at the cuts, from the bits embedding process might not be the best option. By doing so, we create two new problems. First, we remove a lot of frequencies that represent possibilities of additional redundancy, making the watermark a bit less robust than we could have. Then, we might remove some of the lowest frequencies, decreasing at the same time the imperceptibility of the watermark.

A possible lead to counter this problem would be to consider the border vertices for the embedding of the watermark in only one of the two patches that the vertex belongs to, limiting both imperceptibility and redundancy problems to half of their previous cost. This would imply modifying the coordinates in the other of the two patches to match its new value from the first patch.

Another option to increment the number of bits inserted would be to implement the second solution presented in Section 5.1.4. Instead of leaving aside frequencies whose C_{inter} corresponds to the z coordinates, we could modify the corresponding C_{min} and C_{max}. This would increase the number of inserted bits by roughly 50% but it would also mean much more perceptible changes since, in those cases, we modify two values instead of one.

We compared this solution with the two previous algorithms in terms of robustness and imperceptibility trade-off. Increasing the robustness of an algorithm will always induce a decrease of its imperceptibility. It resulted that the algorithm based on volume moments privileges the imperceptibility over robustness. On the other hand, schemes using spectral decomposition tend to be more permissive and offer the user the possibility to choose between robustness and imperceptibility. With a wise choice of partitioning, all the three alternatives offer highly resistant and nearly imperceptible watermarks.

Regarding attacks, the three schemes offer different levels of robustness. The volume moments-based scheme offers some resistance to both noise addition and smoothing as long as the amplitude and noise addition are both limited. The spectral decomposition offers a solution that is highly resistant to noise addition but not at all to smoothing. Finally, our proposition of layers partitioning is not at all resistant to smoothing attacks and random noise additions. Such attacks might significantly change the maximum and minimum z-values of the models that we used to compute the interval of accepted values in the different layers. Modifications into the intervals would create patches completely different from before the attack and completely scramble the retrieved bits. There are no obvious fixes to this problem, which would require some more reflection.

Finally, we could imagine adapting this decomposition into layers to many more watermarking algorithms less dependent on the connectivity of the mesh than the spectral decomposition in order to make them more resistant to re-meshing. We hope that this work will lead to other similar ideas and, maybe, help researchers develop more robust watermarking schemes in the future.

References

[1] N. Aspert, D. Santa-Cruz, and T. Ebrahimi. Mesh: measuring errors between surfaces using the hausdorff distance. In *Proceedings. IEEE International Conference on Multimedia and Expo*, volume 1, pages 705–708 vol.1, 2002.

[2] G. Cassiers and F. X Standaert. Information embedding, physical authentication and ip detection/protection mechanisms for additive manufacturing. Report from Summer 2017's internship at the MIT, 2017.

[3] F. Cayre, P. Rondao-Alface, B. Macq, and H. Maître. Application of spectral decomposition to compression and watermarking of 3D triangle mesh geometry. *Signal Processing: Image Communication*, 18(4):309–319, 2003.

[4] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia. MeshLab: an Open-Source Mesh Processing Tool. In Vittorio Scarano, Rosario De Chiara, and Ugo Erra, editors, *Eurographics Italian Chapter Conference*. The Eurographics Association, 2008.

[5] The SciPy Community. *NumPy's documentation*, 1.14 edition, 2018.

[6] I. Cox, M. Miller, J. Bloom, J. Fridrich, and T. Kalker. *Digital Watermarking and Steganography*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2 edition, 2008.

[7] G. Karypis and V. Kumar. Metis - a software package for partitioning unstructured graphs, partitioning meshes and computing fill-reducing ordering of sparse matrices. 1997.

[8] J.-P Kruth, M. Leu, and T. Nakagawa. Progress in additive manufacturing and rapid prototyping. 47:525–540, 01 1998.

[9] P. Kulkarni, A. Marsan, and D. Dutta. A review of process planning techniques in layered manufacturing. *Rapid Prototyping Journal*, 6(1):18–35, 2000.

[10] K. Wang, G. Lavoué, F. Denis, and A. Baskurt. Robust and blind watermarking of polygonal meshes based on volume moments. 2009.

[11] Q. Zhou. *PyMesh Documentation*, 0.1 edition, 2018.

Appendices

A Volume Moments Evaluation

A.1 Global Measurements

α	n_h	n_θ	Cap	Err	RMS Error	Hausdorff Distance	Local Smoothness
0.5	2	7	1	0	0.057766187439	0.001993754689	0.000134247073
0.5	2	8	3	0	0.211452065926	0.011566286855	0.000453885866
0.5	3	5	2	0	0.096164618587	0.003283250451	0.000232353977
0.5	3	6	5	0	0.287740848395	0.018867728866	0.000557588728
0.5	3	7	8	0	0.302568286556	0.010848409879	0.000672651396
0.5	3	8	11	0	0.564000635235	0.032578454331	0.001210871341
0.5	4	4	3	0	0.128825955498	0.016467022884	0.000137424150
0.5	4	5	7	0	0.311188106278	0.027910673459	0.000635702473
0.5	4	6	11	2	0.322395667351	0.040118733879	0.000591436635
0.5	4	7	15	1	0.329351739075	0.030134744392	0.000747166161
0.5	4	8	19	2	0.599666124237	0.028219035158	0.001807199569
0.5	5	3	1	0	0.164361336769	0.033710006463	0.000202403733
0.5	5	4	5	3	0.353837111423	0.066629139015	0.000722890509
0.5	5	5	9	1	0.189146037766	0.033554694031	0.000477215384
0.5	5	6	13	4	1.606517565600	0.054425136631	0.004828374430
0.5	5	7	17	1	0.524428532933	0.085425876236	0.001288320866
0.5	5	8	22	2	0.389624090543	0.054634865568	0.001157162137
0.5	6	3	1	0	0.028576495049	0.003525761593	0.000043603314
0.5	6	4	5	1	0.122129071654	0.002938586492	0.000355371418
0.5	6	5	9	2	0.446748719393	0.039964776294	0.001136119076
0.5	6	6	14	4	0.571466069884	0.039500886673	0.001501103654
0.5	6	7	18	2	1.140559503850	0.043138811159	0.002806244469
0.5	6	8	23	0	0.632659610202	0.030449325801	0.001757960034
0.5	7	3	2	0	0.127767107810	0.005713104580	0.000266114335
0.5	7	4	7	0	0.310079565998	0.009856785246	0.000847455085
0.5	7	5	12	0	0.266000335343	0.012444744007	0.000566878912
0.5	7	6	17	0	0.434719153493	0.018407463978	0.001028883003
0.5	7	7	21	1	0.508303582974	0.018314306599	0.001385198157
0.5	7	8	27	0	0.582260661621	0.019639336321	0.001705305703
0.5	8	3	5	0	0.207840169344	0.010606662653	0.000420517115
0.5	8	4	11	2	0.459717978329	0.014920161218	0.001427450930
0.5	8	5	17	1	0.402605758605	0.016916104506	0.001076615855
0.5	8	6	22	0	0.380222708168	0.018969128927	0.001104518151
0.5	8	7	27	2	0.591138390898	0.022351880761	0.001724070349
0.5	8	8	34	3	0.528713910230	0.019091093250	0.001505104143
0.5	9	2	1	0	0.037493703337	0.001470384378	0.000075664442
0.5	9	3	8	2	0.358410408159	0.028351317972	0.001016647360
0.5	9	4	15	2	0.326893813563	0.023197630318	0.000922389510
0.5	9	5	22	1	0.463794919327	0.019707489529	0.001232197911
0.5	9	6	29	4	1.111680332610	0.033727623283	0.002940494275
0.5	9	7	34	1	0.441043009631	0.018776094021	0.001119449265
0.5	9	8	42	3	1.013592763330	0.022490857916	0.003010742452
0.5	10	2	3	0	0.182348684967	0.007877744494	0.000381621402
0.5	10	3	10	0	0.342822930181	0.011120987360	0.000881216632

α	n_h	n_θ	**Cap**	**Err**	**RMS Error**	**Hausdorff Distance**	**Local Smoothness**
0.5	10	4	17	1	0.438254222038	0.050863728868	0.001127429551
0.5	10	5	24	4	0.410508196458	0.026446513919	0.001122658958
0.5	10	6	31	3	0.469535631883	0.025081319062	0.001171229997
0.5	10	7	37	1	0.569973799008	0.027525844008	0.001421316799
0.5	10	8	45	1	1.043061665620	0.046533293965	0.002552425174
0.5	11	2	3	1	0.336188717988	0.008846346756	0.000827916509
0.5	11	3	10	2	0.277813825774	0.010974416632	0.000583430803
0.5	11	4	17	1	0.308331050576	0.010145066210	0.000739725671
0.5	11	5	25	4	0.562229038216	0.018570200285	0.001432830035
0.5	11	6	32	5	0.907976798777	0.025669463032	0.002294642058
0.5	11	7	35	2	0.861155050825	0.025569586698	0.002022681057
0.5	11	8	44	2	1.357841354210	0.038369006295	0.003212483533
0.6	2	7	1	0	0.070567478019	0.002436811287	0.000164770774
0.6	2	8	3	0	0.254135238080	0.013631695222	0.000525349182
0.6	3	5	2	0	0.123628931574	0.004221322008	0.000298057687
0.6	3	6	5	0	0.336373875015	0.023375144782	0.000628890935
0.6	3	7	8	0	0.405289091957	0.014090838792	0.000922360960
0.6	3	8	11	0	0.783435912963	0.041558412896	0.001693716874
0.6	4	4	3	0	0.153833745672	0.020172103032	0.000170993276
0.6	4	5	7	0	0.372921462963	0.031194282101	0.000787128474
0.6	4	6	11	1	0.405951553516	0.046345661657	0.000779864118
0.6	4	7	15	1	0.412891877847	0.032660103454	0.000971941440
0.6	4	8	19	0	0.505123177599	0.034294086941	0.001231266954
0.6	5	3	1	0	0.196221439447	0.040154134795	0.000244312911
0.6	5	4	5	2	0.402041699929	0.081944351122	0.000825570566
0.6	5	5	9	1	0.224523833825	0.039279617401	0.000584867175
0.6	5	6	13	4	1.758149856750	0.066692149560	0.005181975422
0.6	5	7	17	13	0.724936331702	0.090753234409	0.002079763907
0.6	5	8	22	1	0.474467341530	0.061958695757	0.001424549832
0.6	6	3	1	0	0.028576495049	0.003525761593	0.000043603314
0.6	6	4	5	4	0.194480750731	0.004701738388	0.000589208152
0.6	6	5	9	1	0.619145537155	0.045170854410	0.001696073523
0.6	6	6	14	2	0.627580730656	0.047623043209	0.001609986129
0.6	6	7	18	3	1.206066294260	0.045303249195	0.003002682519
0.6	6	8	23	3	1.056380611910	0.037012681108	0.002913925970
0.6	7	3	2	0	0.159309011308	0.007337493881	0.000328094259
0.6	7	4	7	0	0.390847973126	0.011199042977	0.001128369838
0.6	7	5	12	0	0.318250549728	0.014000337007	0.000719061191
0.6	7	6	17	0	0.579598239359	0.022237314430	0.001235124266
0.6	7	7	21	1	0.629748183391	0.021172017599	0.001731215389
0.6	7	8	27	0	0.721203047968	0.023865780809	0.001994657761
0.6	8	3	5	0	0.263995068133	0.012963698799	0.000529676298
0.6	8	4	11	0	0.332904874641	0.009787561678	0.000783091294
0.6	8	5	17	0	0.475802090373	0.019647215671	0.001312850361
0.6	8	6	22	0	0.516286241137	0.023566503306	0.001409785392
0.6	8	7	27	0	0.661992533935	0.027732889092	0.001612197628
0.6	8	8	34	2	0.641590200048	0.023044331304	0.001712288153
0.6	9	2	1	0	0.046854747701	0.001837980473	0.000094303343
0.6	9	3	8	2	0.400029450317	0.034138997171	0.001110252776
0.6	9	4	15	1	0.443378901136	0.027617081139	0.001261906735

α	n_h	n_θ	**Cap**	**Err**	**RMS Error**	**Hausdorff Distance**	**Local Smoothness**
0.6	9	5	22	2	0.547357722773	0.025455508007	0.001448917202
0.6	9	6	29	13	1.317801096410	0.032798952805	0.003862336158
0.6	9	7	34	1	0.559230095360	0.020883611437	0.001497858905
0.6	9	8	42	1	1.181511585910	0.028054286989	0.003355409346
0.6	10	2	3	0	0.201969681224	0.008892937054	0.000417284923
0.6	10	3	10	1	0.415409475761	0.015309871709	0.001056249291
0.6	10	4	17	0	0.500705595549	0.053329658439	0.001240112876
0.6	10	5	24	4	0.493188292796	0.035397641708	0.001371649838
0.6	10	6	31	2	0.567312558915	0.027903210714	0.001397144448
0.6	10	7	37	0	0.678921706545	0.033426813692	0.001732419538
0.6	10	8	45	1	1.132914238210	0.055942524683	0.002866495103
0.6	11	2	3	1	0.404347293566	0.010606777122	0.001002128581
0.6	11	3	10	2	0.315029003159	0.012307448109	0.000648538107
0.6	11	4	17	1	0.362081883702	0.012346878825	0.000845924100
0.6	11	5	25	11	0.708333857790	0.021095961442	0.001840143535
0.6	11	6	32	6	0.980363814184	0.031799155200	0.002366680744
0.6	11	7	35	4	0.967072652434	0.027571679616	0.002354342835
0.6	11	8	44	0	1.483722538970	0.043517049068	0.003519130081
0.7	2	7	1	0	0.083384810729	0.002879867884	0.000195663896
0.7	2	8	3	0	0.315649018916	0.016110185263	0.000682171812
0.7	3	5	2	0	0.144406284925	0.004924875676	0.000347790248
0.7	3	6	5	0	0.397033486072	0.029277545966	0.000714179103
0.7	3	7	8	0	0.515904136432	0.018196252350	0.001179248201
0.7	3	8	11	0	1.060835820560	0.049180935865	0.002307036845
0.7	4	4	3	0	0.179095105388	0.023877183181	0.000205670783
0.7	4	5	7	0	0.429947212102	0.033246537503	0.000926893862
0.7	4	6	11	0	0.478921082203	0.049166428548	0.000972288075
0.7	4	7	15	0	0.374034030859	0.034585634806	0.000710854659
0.7	4	8	19	0	0.613267787948	0.040523060715	0.001605389274
0.7	5	3	1	0	0.228093442435	0.046642749307	0.000285916498
0.7	5	4	5	2	0.444982538653	0.092516144335	0.000909268661
0.7	5	5	9	2	0.261271177763	0.044592376033	0.000711613569
0.7	5	6	13	4	1.817589116650	0.082464488218	0.005530778705
0.7	5	7	17	14	0.811450440578	0.090753234409	0.002372468158
0.7	5	8	22	1	0.615231322809	0.075664433238	0.001807640912
0.7	6	3	1	0	0.038093910834	0.004701015457	0.000058041165
0.7	6	4	5	2	0.246254497072	0.007052607581	0.000777782116
0.7	6	5	9	3	0.880827844624	0.046727861256	0.002565517810
0.7	6	6	14	4	0.675395013622	0.057918795525	0.001711908058
0.7	6	7	18	11	1.254641628150	0.044801737211	0.003270151332
0.7	6	8	23	1	1.367058390900	0.042819163934	0.003827648109
0.7	7	3	2	0	0.192733133376	0.008584431222	0.000391269742
0.7	7	4	7	0	0.490837532817	0.013106604019	0.001480222954
0.7	7	5	12	0	0.377052656270	0.016333726509	0.000898744604
0.7	7	6	17	1	0.762517637121	0.028984653249	0.001979327693
0.7	7	7	21	0	0.560402287868	0.017686140119	0.001715203098
0.7	7	8	27	1	0.839885606446	0.029220791216	0.002182133823
0.7	8	3	5	0	0.324811482378	0.017014426034	0.000648293579
0.7	8	4	11	0	0.384171094406	0.011201116721	0.000893558380
0.7	8	5	17	0	0.597737120012	0.022274376424	0.001542104432

α	n_h	n_θ	**Cap**	**Err**	**RMS Error**	**Hausdorff Distance**	**Local Smoothness**
0.7	8	6	22	0	1.010056630750	0.025363436163	0.002957554632
0.7	8	7	27	0	0.779135945397	0.032979441001	0.002121844374
0.7	8	8	34	3	0.740915468833	0.026158430129	0.001984075881
0.7	9	2	1	0	0.056252061276	0.002205576568	0.000113181594
0.7	9	3	8	0	0.428517553682	0.040421019825	0.001119320903
0.7	9	4	15	1	0.421198006366	0.032543960178	0.001375652002
0.7	9	5	22	2	0.623687129427	0.033671548324	0.001661338773
0.7	9	6	29	13	1.345465341890	0.033685410988	0.003965172124
0.7	9	7	34	0	0.664137627524	0.022993067137	0.001758443021
0.7	9	8	42	2	1.396259878050	0.038926677510	0.003899728845
0.7	10	2	3	0	0.222346218889	0.009538039455	0.000453569645
0.7	10	3	10	1	0.498931442146	0.019861455189	0.001273889854
0.7	10	4	17	1	0.572400768541	0.019688466713	0.001409539923
0.7	10	5	24	4	0.616746980635	0.044205095704	0.001803174408
0.7	10	6	31	22	1.247077279480	0.037815627994	0.003378178715
0.7	10	7	37	2	0.798090206477	0.054761080771	0.002083295878
0.7	10	8	45	4	1.003815384980	0.065456015239	0.002848701029
0.7	11	2	3	0	0.486797111110	0.012349545506	0.001204773162
0.7	11	3	10	2	0.346009684869	0.013674942344	0.000702891968
0.7	11	4	17	1	0.424928860818	0.013482361685	0.000959196173
0.7	11	5	25	2	0.827631572133	0.023746708258	0.002020126470
0.7	11	6	32	8	1.109226678150	0.032472866116	0.002682604799
0.7	11	7	35	3	1.137424456140	0.032342037234	0.002790639285
0.7	11	8	44	16	1.875387535780	0.046605969178	0.004580694130
0.8	2	7	1	0	0.096166235088	0.003322924482	0.000226638740
0.8	2	8	3	0	0.489239877735	0.018175593630	0.001200311406
0.8	3	5	2	0	0.164721678160	0.005628429344	0.000395699838
0.8	3	6	5	0	0.463762972607	0.034850276148	0.000782365974
0.8	3	7	8	0	0.633569980573	0.022439335016	0.001447338319
0.8	3	8	11	0	0.684563269751	0.027618593495	0.001586161983
0.8	4	4	3	0	0.202181131643	0.027437360036	0.000241714909
0.8	4	5	7	0	0.482327809251	0.033752604187	0.001061670732
0.8	4	6	11	0	0.564866588599	0.050877220138	0.001196451450
0.8	4	7	15	0	0.588161191615	0.043954135504	0.001474353249
0.8	4	8	19	0	0.763451052319	0.045348789642	0.002219557178
0.8	5	3	1	0	0.259988397160	0.053159562649	0.000327105895
0.8	5	4	5	3	0.487294261164	0.103168526071	0.000980437745
0.8	5	5	9	2	0.317931074898	0.051572678160	0.000891907116
0.8	5	6	13	6	1.783965675660	0.048467483199	0.005321261154
0.8	5	7	17	13	0.890264803991	0.089753027271	0.002584954500
0.8	5	8	22	1	1.041724965720	0.097752286474	0.003139381945
0.8	6	3	1	0	0.038093910834	0.004701015457	0.000058041165
0.8	6	4	5	2	0.331062903727	0.010578911372	0.001028745443
0.8	6	5	9	2	0.605130460396	0.048954877803	0.001353685590
0.8	6	6	14	7	0.738210732083	0.092683201631	0.001986812518
0.8	6	7	18	8	1.375238620440	0.046473443824	0.003741409769
0.8	6	8	23	0	1.247530190400	0.049706926818	0.003425384555
0.8	7	3	2	0	0.228072653387	0.010341659670	0.000457231645
0.8	7	4	7	0	0.633733832186	0.015211383700	0.001969492714
0.8	7	5	12	0	0.454905277566	0.020169754902	0.001137034229

α	n_h	n_θ	**Cap**	**Err**	**RMS Error**	**Hausdorff Distance**	**Local Smoothness**
0.8	7	6	17	1	0.767702524145	0.033191178824	0.001770394271
0.8	7	7	21	0	0.621093937105	0.019703025822	0.002045563829
0.8	7	8	27	2	0.872910789815	0.036054340941	0.002219327013
0.8	8	3	5	0	0.397333962553	0.021957203461	0.000772709317
0.8	8	4	11	0	0.607345153822	0.012267977613	0.001826850574
0.8	8	5	17	0	0.658140391654	0.027224262599	0.001678527132
0.8	8	6	22	0	0.598606786445	0.030317060227	0.001469370956
0.8	8	7	27	2	0.866486952424	0.037176516472	0.002362686852
0.8	8	8	34	2	0.868736251159	0.030071085033	0.002551662308
0.8	9	2	1	0	0.065628482934	0.002573172662	0.000131991722
0.8	9	3	8	2	0.559341631935	0.046458004134	0.001535762537
0.8	9	4	15	1	0.528792831806	0.037577844422	0.001746478182
0.8	9	5	22	0	0.630312602191	0.036349826134	0.001593824318
0.8	9	6	29	15	1.362365109490	0.038030341658	0.004033746225
0.8	9	7	34	2	0.832190029918	0.035970015048	0.002322739852
0.8	9	8	42	17	1.603751748970	0.042827146199	0.004421154924
0.8	10	2	3	1	0.958831410546	0.022250028244	0.002617223433
0.8	10	3	10	2	0.681385416772	0.017526224429	0.001825367762
0.8	10	4	17	3	0.683160474907	0.022684537734	0.001706224204
0.8	10	5	24	4	0.697747885292	0.055153700106	0.002045340898
0.8	10	6	31	1	1.487809105460	0.051807954385	0.004086564527
0.8	10	7	37	2	0.887013016622	0.051138831538	0.002366582798
0.8	10	8	45	5	1.456896233390	0.076635052956	0.004023307927
0.8	11	2	3	0	0.567267399333	0.014118673200	0.001403057618
0.8	11	3	10	3	0.379815556725	0.014901348241	0.000752958764
0.8	11	4	17	1	0.487593789522	0.015920729879	0.001072167462
0.8	11	5	25	3	0.905549639885	0.026074783474	0.002185367954
0.8	11	6	32	6	1.164100267200	0.035182708999	0.003000070148
0.8	11	7	35	4	1.264482756940	0.035370531886	0.003164885411
0.8	11	8	44	16	1.992235603980	0.060383582334	0.004739491328
0.9	2	7	1	0	0.102550816110	0.003544452780	0.000242202745
0.9	2	8	3	0	0.483677370506	0.020447542833	0.001165739341
0.9	3	5	2	0	0.192315994907	0.006566500902	0.000461342607
0.9	3	6	5	0	0.548410028486	0.041662973306	0.000924615080
0.9	3	7	8	0	0.733662386347	0.026157250688	0.001670565891
0.9	3	8	11	0	0.829654385073	0.040775276975	0.002079538317
0.9	4	4	3	0	0.223434633565	0.030602773679	0.000274938775
0.9	4	5	7	1	0.517643893481	0.042154467513	0.000883576992
0.9	4	6	11	0	0.599155287906	0.054350831026	0.001116610087
0.9	4	7	15	0	0.732498313543	0.050306600556	0.001946501190
0.9	4	8	19	1	0.872026967098	0.047043825519	0.002345359606
0.9	5	3	1	0	0.291891369282	0.059695340316	0.000367749001
0.9	5	4	5	3	0.532701735491	0.114935053212	0.001043797796
0.9	5	5	9	3	0.416032510423	0.060517841890	0.001192329216
0.9	5	6	13	6	1.683408144800	0.052765728790	0.005089849854
0.9	5	7	17	11	1.012093021530	0.086220350191	0.002863771378
0.9	5	8	22	0	0.660111587259	0.062176973815	0.001859879007
0.9	6	3	1	0	0.047599518181	0.005876269321	0.000072414391
0.9	6	4	5	1	0.407814996784	0.014692932461	0.001266547064
0.9	6	5	9	4	0.703282096537	0.048664194404	0.001721939587

α	n_h	n_θ	**Cap**	**Err**	**RMS Error**	**Hausdorff Distance**	**Local Smoothness**
0.9	6	6	14	6	0.813833802900	0.102310261834	0.002189983893
0.9	6	7	18	14	1.610960242160	0.050381681986	0.004398496091
0.9	6	8	23	0	1.393868562130	0.053796757532	0.003800206089
0.9	7	3	2	0	0.265338160792	0.012187101209	0.000526890487
0.9	7	4	7	0	0.799113585243	0.018132417322	0.002504117554
0.9	7	5	12	0	0.592396603457	0.027373342697	0.001584937257
0.9	7	6	17	0	0.757033020790	0.029412154999	0.001855928705
0.9	7	7	21	1	0.743819054205	0.028527053134	0.001894379783
0.9	7	8	27	2	1.139662647970	0.038606506144	0.003019568180
0.9	8	3	5	0	0.451593956263	0.023673457972	0.000938148682
0.9	8	4	11	0	0.670068961673	0.013182337162	0.002063561575
0.9	8	5	17	0	0.680377142442	0.035711087083	0.001447830094
0.9	8	6	22	0	0.661890047217	0.035626139474	0.001610854387
0.9	8	7	27	3	0.975426655159	0.040277553469	0.002825390828
0.9	8	8	34	1	0.968913983622	0.032381075928	0.002638875836
0.9	9	2	1	0	0.070328513392	0.002756970709	0.000141567075
0.9	9	3	8	1	0.549565467008	0.052559746025	0.001490157489
0.9	9	4	15	1	0.629901019317	0.042680889315	0.002043293539
0.9	9	5	22	0	0.703832192710	0.035979522931	0.001772188924
0.9	9	6	29	15	1.272737714590	0.043648049207	0.003607780979
0.9	9	7	34	2	1.019673170150	0.027211978539	0.002844494034
0.9	9	8	42	17	1.778743862040	0.048059271259	0.004920299549
0.9	10	2	3	1	0.961769370030	0.019836793077	0.002690877811
0.9	10	3	10	2	0.718228711630	0.019861455189	0.001893867809
0.9	10	4	17	0	0.760281926776	0.086641792909	0.001963745308
0.9	10	5	24	5	0.772929183849	0.063865164220	0.002285664673
0.9	10	6	31	2	1.230238413920	0.034306510725	0.003374373219
0.9	10	7	37	17	0.974362646405	0.057336704453	0.002644482840
0.9	10	8	45	6	1.714750686690	0.082446440883	0.004937693456
0.9	11	2	3	0	0.644707071907	0.015825466002	0.001582386269
0.9	11	3	10	3	0.418948291541	0.016006036132	0.000816399992
0.9	11	4	17	1	0.560774173728	0.018856085631	0.001187270499
0.9	11	5	25	3	0.986400953485	0.029285991441	0.002385518942
0.9	11	6	32	4	1.225002152100	0.032405495024	0.003156762264
0.9	11	7	35	4	0.948260125831	0.029890454858	0.002364473061
0.9	11	8	44	1	1.393867125420	0.033726613832	0.003465046792
1.0	2	7	1	0	0.115350070258	0.003987509378	0.000273456408
1.0	2	8	3	0	0.477591865896	0.022719492037	0.001113878969
1.0	3	5	2	0	0.213141941961	0.007270054570	0.000510979402
1.0	3	6	5	0	0.653543968887	0.047983766372	0.001118557290
1.0	3	7	8	0	0.663294026336	0.043817777617	0.001432279993
1.0	3	8	11	0	1.266157853240	0.054401842007	0.003229487673
1.0	4	4	3	0	0.245004955860	0.033792659193	0.000308545308
1.0	4	5	7	1	0.540415051510	0.046621133586	0.000894772157
1.0	4	6	11	0	0.649833677348	0.061070869866	0.001263962820
1.0	4	7	15	0	0.700555318147	0.056128914916	0.001648888897
1.0	4	8	19	1	0.879641334747	0.050632075259	0.002275565469
1.0	5	3	1	0	0.318504347085	0.065152191472	0.000401279684
1.0	5	4	5	3	0.577441063777	0.124340208859	0.001093899775
1.0	5	5	9	3	0.611218724401	0.073909877448	0.001819610823

α	n_h	n_θ	**Cap**	**Err**	**RMS Error**	**Hausdorff Distance**	**Local Smoothness**
1.0	5	6	13	3	1.870036189500	0.049325497048	0.005374670417
1.0	5	7	17	11	1.008717662690	0.091790780311	0.002765077764
1.0	5	8	22	0	0.711980212077	0.080525451344	0.001877501759
1.0	6	3	1	0	0.052350566859	0.006463896253	0.000079566594
1.0	6	4	5	2	0.560089562840	0.022920974639	0.001699208762
1.0	6	5	9	4	1.169460904090	0.049150702619	0.003007180358
1.0	6	6	14	6	0.886898764514	0.109581459471	0.002389764923
1.0	6	7	18	10	1.916328491970	0.052637578194	0.005198921158
1.0	6	8	23	2	1.629700348680	0.142819801043	0.003856232457
1.0	7	3	2	0	0.308059478852	0.014255323387	0.000600667029
1.0	7	4	7	6	1.401087652930	0.040822251877	0.004115041255
1.0	7	5	12	1	0.949942562700	0.041122386606	0.002644285199
1.0	7	6	17	1	1.093448848180	0.037270543876	0.002845790945
1.0	7	7	21	2	0.868878548043	0.028187214317	0.002201493274
1.0	7	8	27	4	1.570291142860	0.040607922676	0.004630019876
1.0	8	3	5	0	0.448502588695	0.021596838852	0.000963340653
1.0	8	4	11	0	0.571713151604	0.012799915181	0.001684506354
1.0	8	5	17	0	0.739006109003	0.034032976941	0.001793620036
1.0	8	6	22	0	0.725092571234	0.038866860301	0.001718388370
1.0	8	7	27	1	1.045820558210	0.039378929761	0.003043181011
1.0	8	8	34	13	2.058293016920	0.049899016254	0.005562381242
1.0	9	2	1	0	0.075024281359	0.002940768757	0.000151125816
1.0	9	3	8	0	0.501895985325	0.058481236831	0.001304924057
1.0	9	4	15	1	0.732000113075	0.047727833070	0.002363110180
1.0	9	5	22	11	1.823641489920	0.037504383845	0.005432090433
1.0	9	6	29	17	1.432524327720	0.050586233884	0.004161253199
1.0	9	7	34	3	1.104615724750	0.043817777617	0.003276242862
1.0	9	8	42	2	1.474783734340	0.044952263774	0.004387423396
1.0	10	2	3	1	0.980037121766	0.018871499010	0.002768332925
1.0	10	3	10	3	0.856683156909	0.035256411262	0.002221402905
1.0	10	4	17	4	1.045510089480	0.090276270926	0.003114348056
1.0	10	5	24	5	1.289643592210	0.056355384441	0.003710763969
1.0	10	6	31	3	1.237483968420	0.038393923084	0.003355282180
1.0	10	7	37	0	1.181028030430	0.075893717740	0.002946158773
1.0	10	8	45	5	2.022976216220	0.061021958440	0.005844611457
1.0	11	2	3	0	0.705591339868	0.018707624526	0.001684899751
1.0	11	3	10	4	0.467532627337	0.016616250374	0.000939204798
1.0	11	4	17	2	0.678433904319	0.023405918173	0.001435229785
1.0	11	5	25	6	1.073061575820	0.033129325426	0.002588495110
1.0	11	6	32	4	1.295596935350	0.030923331010	0.003451395695
1.0	11	7	35	3	1.072746119200	0.032916034920	0.002733601748
1.0	11	8	44	3	1.507602558490	0.089189139406	0.004045378241

A.2 Robustness against a smoothing attack

α	n_h	n_θ	**Iterations**	**Errors**
5	11	8	1	10
5	11	8	2	16
5	11	8	3	16
5	11	8	5	14
5	11	8	7	19
5	11	8	10	19
6	11	8	1	6
6	11	8	2	13
6	11	8	3	14
6	11	8	5	15
6	11	8	7	19
6	11	8	10	24
7	11	8	1	20
7	11	8	2	18
7	11	8	3	21
7	11	8	5	20
7	11	8	7	19
7	11	8	10	22
8	11	8	1	16
8	11	8	2	20
8	11	8	3	21
8	11	8	5	19
8	11	8	7	15
8	11	8	10	20
9	11	8	1	8
9	11	8	2	10
9	11	8	3	13
9	11	8	5	14
9	11	8	7	16
9	11	8	10	19
10	11	8	1	9
10	11	8	2	11
10	11	8	3	11
10	11	8	5	16
10	11	8	7	19
10	11	8	10	21

A.3 Robustness against random noise addition

α	n_h	n_θ	**Amplitude**	**Errors**
5	11	8	0.01	0
5	11	8	0.02	1
5	11	8	0.03	9
5	11	8	0.05	9
5	11	8	0.07	11
5	11	8	0.1	16
6	11	8	0.01	2
6	11	8	0.02	6
6	11	8	0.03	7
6	11	8	0.05	7
6	11	8	0.07	14
6	11	8	0.1	21
7	11	8	0.01	15
7	11	8	0.02	17
7	11	8	0.03	20
7	11	8	0.05	18
7	11	8	0.07	18
7	11	8	0.1	24
8	11	8	0.01	17
8	11	8	0.02	17
8	11	8	0.03	17
8	11	8	0.05	18
8	11	8	0.07	18
8	11	8	0.1	17
9	11	8	0.01	1
9	11	8	0.02	6
9	11	8	0.03	8
9	11	8	0.05	9
9	11	8	0.07	12
9	11	8	0.1	13
10	11	8	0.01	3
10	11	8	0.02	7
10	11	8	0.03	10
10	11	8	0.05	9
10	11	8	0.07	14
10	11	8	0.1	15

B Spectral Decomposition Evaluation

B.1 Global Measurements

Str [%]	Patches	Cap	Err	RMS Error	Hausdorff Distance	Local Smoothness
1	100	449	16	1.4121490222	0.10670692920	0.005046440856
2	100	835	12	2.2417122440	0.10670692920	0.009173540655
3	100	1232	8	3.3299489885	0.15975198215	0.014264280428
5	100	2025	6	5.4080031966	0.20579465452	0.024002436463
7	100	2816	0	7.4726671687	0.25852040298	0.033412146896
10	100	3995	0	10.534348244	0.36015191024	0.048196963392
20	100	7940	0	22.665589479	0.58518428031	0.129802769647
30	100	11898	0	38.196149974	1.21534066477	0.238794957750
50	100	19780	0	80.093528693	1.76271170084	0.644373151467
70	100	27694	0	135.21799367	4.20552923973	1.367851470940
100	100	39502	0	315.34166052	6.48391236596	2.442642289040
1	90	436	23	1.3356160159	0.08179576278	0.005071438878
2	90	825	13	2.4145903773	0.18001337326	0.009841282752
3	90	1229	5	3.5064054149	0.18025777738	0.015084210221
5	90	2009	4	5.7238806319	0.20704826337	0.025644762901
7	90	2794	2	7.6438481973	0.22685196967	0.034950508341
10	90	3972	1	10.855699623	0.34541315549	0.050180320280
20	90	7899	0	23.174360677	0.52866360913	0.141689291289
30	90	11834	0	38.548494563	1.00589079024	0.254903866203
50	90	19678	0	79.246284475	1.94781057954	0.514742337060
70	90	27559	0	134.98145298	3.29548526173	1.217808920110
100	90	39310	0	464.40294350	8.75911724848	2.603976114320
1	80	430	20	1.5426466043	0.10420882131	0.005978731328
2	80	823	14	2.6418040396	0.10362099590	0.011250011195
3	80	1210	9	3.6556463110	0.18983840800	0.015851213633
5	80	1998	3	5.4947577659	0.21442943190	0.024927439701
7	80	2774	2	7.6294767118	0.28725009131	0.034856547833
10	80	3945	0	10.681907550	0.34769386019	0.050144950432
20	80	7846	0	22.440833241	0.55967912174	0.146618121560
30	80	11756	0	36.578577656	1.02314849674	0.247768964585
50	80	19554	0	74.865582300	1.86120489560	0.487619856370
70	80	27388	0	130.71975819	3.62677460134	1.245493124670
100	80	39072	0	310.64468135	6.31271458383	2.302424043380
1	70	419	15	1.4260931470	0.07905767508	0.005591174230
2	70	805	12	2.4105532233	0.12393632985	0.010081004856
3	70	1194	14	3.5607552151	0.13524935233	0.015707533675
5	70	1968	6	5.3811797933	0.14021133843	0.024769139615
7	70	2748	7	7.3473055750	0.20642106331	0.034152573432
10	70	3902	2	10.416996190	0.25991524075	0.048779552421
20	70	7773	0	21.984352076	0.52874792383	0.111094246802
30	70	11656	0	36.112376819	0.83080034487	0.221403504607
50	70	19392	0	73.024366180	1.64169013851	0.389306114095
70	70	27158	0	124.24448284	2.40785194616	1.041899571580
100	70	38754	0	335.89367717	9.93498646023	2.306887707410
1	60	410	17	1.3477216533	0.12388676208	0.005330351443
2	60	796	10	2.3643990076	0.13191687740	0.009928191072
3	60	1183	4	3.4829155252	0.13123768733	0.015394869425

Str [%]	Patches	Cap	Err	RMS Error	Hausdorff Distance	Local Smoothness
5	60	1949	3	5.6683685214	0.17978251519	0.025712107452
7	60	2715	1	7.6863266399	0.19463931965	0.035631409871
10	60	3866	1	10.761973836	0.27121537460	0.050542929566
20	60	7704	0	22.237516571	0.58725585422	0.115106038266
30	60	11550	0	35.678547811	0.80471014989	0.213527941910
50	60	19217	0	71.054131290	1.50419088810	0.349281758825
70	60	26914	0	119.30150999	2.43043937583	0.994784131574
100	60	38407	0	369.57629000	8.07070492875	2.468130956800
1	50	407	14	1.3868453287	0.05731822310	0.005826908174
2	50	785	11	2.3684484664	0.12724651728	0.010657018947
3	50	1166	8	3.3613237269	0.12788030187	0.015412950514
5	50	1923	3	5.3205373097	0.16789728859	0.024788436171
7	50	2687	3	7.3242962740	0.23788162820	0.034455832180
10	50	3821	0	10.184502384	0.27281501350	0.048161397696
20	50	7621	0	21.121611814	0.45701758348	0.109367835901
30	50	11422	0	34.462376367	0.80978774030	0.210346229324
50	50	19009	0	67.765602406	1.71344977423	0.327857182043
70	50	26617	0	112.13410120	2.52071433240	0.822960460116
100	50	37990	0	314.37176836	5.72018601035	2.268094029650
1	40	397	12	1.2991171082	0.05111242642	0.005517321272
2	40	778	8	2.4137910544	0.16917470153	0.010855777918
3	40	1148	4	3.3646144619	0.16917036862	0.015278410457
5	40	1903	0	5.4509866312	0.18710956840	0.025292832150
7	40	2657	0	7.4591462616	0.20413883880	0.034907962372
10	40	3787	0	10.542488697	0.26010216567	0.050036840405
20	40	7553	0	21.324298224	0.50694758896	0.107172268064
30	40	11323	0	33.811495712	0.75270795396	0.200611201318
50	40	18850	0	65.865903875	1.55693988248	0.327818178175
70	40	26393	0	110.65564902	2.31996949650	0.774111792661
100	40	37679	0	318.36018197	5.71594016944	2.064875389710
1	30	387	9	1.3403437225	0.09598305536	0.005791130011
2	30	762	10	2.3881711362	0.10439869259	0.011024555773
3	30	1135	5	3.5888100778	0.11482555774	0.016661830489
5	30	1882	2	5.4845504224	0.17222512472	0.025798346777
7	30	2629	3	7.4793617122	0.22062849317	0.035566673414
10	30	3748	0	10.362728168	0.26105556816	0.049419830007
20	30	7485	0	20.905776491	0.46653079377	0.104240129350
30	30	11224	0	33.142859571	0.64395170724	0.202366621842
50	30	18691	0	63.579769370	1.26549956371	0.313945427047
70	30	26173	0	105.11688505	2.34156536786	0.719959677804
100	30	37371	0	294.14982756	6.72132339694	2.021974202340

B.2 Robustness against a smoothing attack

Strength [%]	Patches	Iterations	Errors
1	60	1	22
1	60	2	25
1	60	3	30
1	60	5	31
1	60	7	31
1	60	10	31
2	60	1	23
2	60	2	27
2	60	3	30
2	60	5	32
2	60	7	31
2	60	10	32
3	60	1	29
3	60	2	25
3	60	3	28
3	60	5	32
3	60	7	32
3	60	10	33
5	60	1	31
5	60	2	36
5	60	3	34
5	60	5	35
5	60	7	35
5	60	10	34
7	60	1	28
7	60	2	33
7	60	3	33
7	60	5	34
7	60	7	35
7	60	10	37
10	60	1	28
10	60	2	33
10	60	3	31
10	60	5	35
10	60	7	33
10	60	10	35
20	60	1	30
20	60	2	30
20	60	3	32
20	60	5	33
20	60	7	37
20	60	10	32
30	60	1	26
30	60	2	29
30	60	3	32
30	60	5	34
30	60	7	36
30	60	10	35

Strength [%]	Patches	Iterations	Errors
50	60	1	22
50	60	2	30
50	60	3	29
50	60	5	29
50	60	7	30
50	60	10	30
70	60	1	18
70	60	2	29
70	60	3	32
70	60	5	28
70	60	7	31
70	60	10	33
100	60	1	20
100	60	2	31
100	60	3	34
100	60	5	34
100	60	7	36
100	60	10	36

B.3 Robustness against random noise addition

Strength [%]	Patches	Amplitude	Errors
1	60	0.01	24
1	60	0.02	23
1	60	0.03	33
1	60	0.05	35
1	60	0.07	33
1	60	0.1	42
2	60	0.01	22
2	60	0.02	31
2	60	0.03	28
2	60	0.05	26
2	60	0.07	35
2	60	0.1	35
3	60	0.01	11
3	60	0.02	18
3	60	0.03	32
3	60	0.05	32
3	60	0.07	28
3	60	0.1	31
5	60	0.01	9
5	60	0.02	11
5	60	0.03	25
5	60	0.05	31
5	60	0.07	35
5	60	0.1	31
7	60	0.01	3
7	60	0.02	8
7	60	0.03	21
7	60	0.05	26

Strength [%]	**Patches**	**Amplitude**	**Errors**
7	60	0.07	32
7	60	0.1	31
10	60	0.01	1
10	60	0.02	9
10	60	0.03	13
10	60	0.05	21
10	60	0.07	22
10	60	0.1	32
20	60	0.01	0
20	60	0.02	0
20	60	0.03	6
20	60	0.05	7
20	60	0.07	12
20	60	0.1	20
30	60	0.01	0
30	60	0.02	0
30	60	0.03	0
30	60	0.05	5
30	60	0.07	6
30	60	0.1	13
50	60	0.01	0
50	60	0.02	0
50	60	0.03	0
50	60	0.05	0
50	60	0.07	1
50	60	0.1	2
70	60	0.01	0
70	60	0.02	0
70	60	0.03	0
70	60	0.05	0
70	60	0.07	0
70	60	0.1	0
100	60	0.01	0
100	60	0.02	0
100	60	0.03	0
100	60	0.05	0
100	60	0.07	0
100	60	0.1	0

C Layered Spectral Decomposition Evaluation

C.1 Global Measurements

Str [%]	Layers	Cap	Err	RMS Error	Hausdorff Distance	Local Smoothness
1	40	263	5	1.11917728640	0.11400172854	0.81066863367
2	40	506	0	2.00386370543	0.12190549687	1.34983344739
3	40	758	2	2.80475773351	0.18846799024	1.32584131342
5	40	1254	0	4.40464140469	0.24885645641	1.23378285980
7	40	1757	0	6.08533436882	0.31378076808	1.18411794253
10	40	2486	0	8.85371844526	0.32029503622	1.14661647335
20	40	5014	0	19.5510583094	0.74530926715	1.06384626646
30	40	7590	0	31.2914100353	0.84678379870	1.02673408102
50	40	12970	0	59.6660716894	1.60286053753	0.97313459305
70	40	18498	0	107.903663367	5.28941990700	0.95163441201
100	40	267523	0	415.143232131	9.76450350135	1.38507956702
1	50	261	8	1.08854081646	0.10299195542	0.91921820770
2	50	511	2	1.81325944928	0.10299195542	1.27275192498
3	50	769	0	2.53495248650	0.15548187617	1.14049575341
5	50	1248	0	4.08234424456	0.19094555460	1.08577270607
7	50	1718	0	5.62646771581	0.22169228648	1.05997350602
10	50	2470	0	8.34155360147	0.41376326202	1.03231585221
20	50	5051	0	18.2948174814	0.60482678656	0.94120536535
30	50	7684	0	29.8195586190	1.00558691548	0.91418242378
50	50	13040	0	57.7835184401	2.35492397777	0.89753608556
70	50	18536	0	98.7278478155	3.93161363967	0.88248222140
100	50	259972	0	557.490985666	14.4822466587	1.60712233020
1	60	266	9	0.92438134025	0.11816223815	0.07104111043
2	60	476	3	1.54444546727	0.11816223815	0.17690768165
3	60	690	1	2.11455247157	0.14228584680	0.28389337900
5	60	1113	0	3.48356384853	0.24885645641	1.40030645821
7	60	1593	0	5.24595281249	0.31672015920	1.39988073598
10	60	2242	0	7.17066409718	0.36192220226	1.02377482701
20	60	4544	0	16.5499202097	0.61866199332	1.06838826445
30	60	6976	0	28.0142514400	0.94676724967	0.87739334574
50	60	11886	0	57.4952151831	2.46291173204	0.85276387185
70	60	17036	0	94.3920763994	4.22485997103	0.86037504720
100	60	222137	0	540.886391734	14.1068108797	1.39664706321
1	70	225	12	0.80709918123	0.11496897957	0.11101619081
2	70	419	4	1.28927260266	0.11496897957	0.24544108968
3	70	626	2	1.83503534878	0.15950025109	0.26294634603
5	70	998	1	2.77253801443	0.16165454513	0.47660962151
7	70	1383	0	4.17589437095	0.34600798924	1.00426383875
10	70	1971	0	6.26343067637	0.34996631708	1.22202615288
20	70	4049	0	14.7001420770	0.64661630929	1.02431164034
30	70	6181	0	25.4450451764	1.42235300537	1.02586687966
50	70	10661	0	56.0709923764	3.25182165377	1.02055275749
70	70	15337	0	101.681722208	5.30894730305	0.97081584076
100	70	169000	0	559.123024790	14.2281804623	1.50179002728
1	80	195	11	18.7402917124	1.89458999563	0.01884414071
2	80	353	3	18.7614287137	1.89458999563	0.16743602760
3	80	515	4	18.7827591783	1.89458999563	0.19690867401

Str [%]	Layers	Cap	Err	RMS Error	Hausdorff Distance	Local Smoothness
5	80	844	1	18.8688269394	1.89458999563	0.24622106384
7	80	1164	2	19.0609973016	1.89458999563	0.46460168477
10	80	1646	0	19.4049738806	1.89458999563	0.50046422844
20	80	3343	0	22.3858827556	1.89458999563	1.10181213658
30	80	5123	0	-1	-1	-1
50	80	8942	0	-1	-1	-1
70	80	12867	0	-1	-1	-1
100	80	13004	0	463.216471799	13.1711524482	1.45114203412
1	90	169	13	0.41923805675	0.10761186599	0.07535650924
2	90	306	7	0.89890402393	0.21555771156	0.08382034403
3	90	452	3	1.34895185322	0.32966085946	0.10675151830
5	90	739	3	2.07219374631	0.32966085946	0.13694933895
7	90	1023	2	2.76208810050	0.32966085946	0.16623596949
10	90	1412	0	3.68925516443	0.32966085946	0.20900039412
20	90	2787	0	10.1483746505	1.16408251514	0.64700184960
30	90	4310	0	20.3309199857	1.25309904742	0.85743626160
50	90	7577	0	52.6616307873	3.20535685645	1.65572065068
70	90	10984	0	100.786516321	4.81845448783	1.83444398531
100	90	11912	0	482.088929729	14.1812637126	1.31456005446
1	100	145	13	19.5921392028	6.32566840849	0.03802972595
2	100	239	11	19.5958945700	6.32535754517	0.06686629433
3	100	351	7	19.6034922615	6.32707916754	0.64432146634
5	100	564	4	19.8245380831	6.32702711664	0.70142206591
7	100	812	2	19.8799592258	6.32436415277	0.53979962481
10	100	1106	3	20.0002956695	6.32347965058	0.59971212346
20	100	2150	0	21.6935788177	6.32211504463	0.93764874337
30	100	3288	0	25.9108098636	6.30660769264	0.99365222434
50	100	5712	0	49.6876750296	6.31353066675	1.25407784420
70	100	8319	0	94.8813303750	6.33337452093	1.34686285240
100	100	9701	0	362.870578598	12.2924088248	1.16055176178

C.2 Robustness against a smoothing attack

Strength [%]	Layers	Iterations	Errors
1	60	1	30
1	60	2	31
1	60	3	31
1	60	5	37
1	60	7	34
1	60	10	31
2	60	1	36
2	60	2	30
2	60	3	33
2	60	5	39
2	60	7	34
2	60	10	27
3	60	1	39
3	60	2	32
3	60	3	32
3	60	5	33

Strength [%]	**Layers**	**Iterations**	**Errors**
3	60	7	32
3	60	10	31
5	60	1	34
5	60	2	33
5	60	3	29
5	60	5	30
5	60	7	34
5	60	10	33
7	60	1	36
7	60	2	34
7	60	3	31
7	60	5	33
7	60	7	34
7	60	10	36
10	60	1	30
10	60	2	38
10	60	3	32
10	60	5	32
10	60	7	33
10	60	10	37
20	60	1	36
20	60	2	36
20	60	3	30
20	60	5	29
20	60	7	35
20	60	10	35
30	60	1	31
30	60	2	36
30	60	3	31
30	60	5	30
30	60	7	38
30	60	10	38
50	60	1	33
50	60	2	32
50	60	3	26
50	60	5	35
50	60	7	32
50	60	10	27
70	60	1	34
70	60	2	34
70	60	3	26
70	60	5	33
70	60	7	32
70	60	10	30
100	60	1	34
100	60	2	33
100	60	3	27
100	60	5	32
100	60	7	32
100	60	10	34

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve **www.uclouvain.be/epl**