Evaluation of the Performance of Randomized FFD Control Grids

Master Thesis

at the

AG Computer Graphics

at the Faculty of Technology
of Bielefeld University

by

Stefan Dresselhaus

October 24, 2017

Supervisor: Prof. Dr. Mario Botsch
Dipl. Math. Andreas Richter

DRAFT

Contents

1 Introduction **3**

2 Background **7**

2.1 What is Freeform–Deformation (FFD)? 7

2.1.1 Why is FFD a good deformation function? 10

2.2 What is evolutionary optimization? 12

2.3 Advantages of evolutionary algorithms 13

2.4 Criteria for the evolvability of linear deformations 14

2.4.1 Variability 15

2.4.2 Regularity 16

2.4.3 Improvement Potential 16

3 Implementation of Freeform–Deformation (FFD) **19**

3.1 Adaption of FFD 19

3.2 Adaption of FFD for a 3D–Mesh 20

3.3 Deformation Grid 22

4 Scenarios for testing evolvability criteria using Freeform–Deformation (FFD) **25**

4.1 Test Scenario: 1D Function Approximation 25

4.2 Procedure: 1D Function Approximation 27

4.3 Test Scenario: 3D Function Approximation 28

4.4 Procedure: 3D Function Approximation 30

DRAFT

5 Evaluation of Scenarios 33
5.1 Results of 1D Function Approximation 34
5.1.1 Variability 34
5.1.2 Regularity 35
5.1.3 Improvement Potential 36
5.2 Results of 3D Function Approximation 37
6 Schluss 41
Appendix i
A Bibliography iii
B Abbreviations vii
C List of Figures ix

DRAFT

How to read this Thesis

As a guide through the nomenclature used in the formulas we prepend this chapter.

Unless otherwise noted the following holds:

- lowercase letters x,y,z

  refer to real variables and represent the coordinates of a point in 3D–Space.
- lowercase letters u,v,w

  refer to real variables between 0 and 1 used as coefficients in a 3D B–Spline grid.
- other lowercase letters

  refer to other scalar (real) variables.
- lowercase **bold** letters (e.g. $\mathbf{x}$,$\mathbf{y}$)

  refer to 3D coordinates
- uppercase **BOLD** letters (e.g. $\mathbf{D}, \mathbf{M}$)

  refer to Matrices

DRAFT

DRAFT

1 Introduction

Improvement: Mehr Bilder

Many modern industrial design processes require advanced optimization methods due to the increased complexity resulting from more and more degrees of freedom as methods refine and/or other methods are used. Examples for this are physical domains like aerodynamic (i.e. drag), fluid dynamics (i.e. throughput of liquid) — where the complexity increases with the temporal and spatial resolution of the simulation — or known hard algorithmic problems in informatics (i.e. layouting of circuit boards or stacking of 3D–objects). Moreover these are typically not static environments but requirements shift over time or from case to case.

Evolutionary algorithms cope especially well with these problem domains while addressing all the issues at hand[1]. One of the main concerns in these algorithms is the formulation of the problems in terms of a *genome* and *fitness–function*. While one can typically use an arbitrary cost–function for the *fitness–functions* (i.e. amount of drag, amount of space, etc.), the translation of the problem–domain into a simple parametric representation (the *genome*) can be challenging.

This translation is often necessary as the target of the optimization may have too many degrees of freedom. In the example of an aerodynamic simulation of drag onto an object, those objects–designs tend to have a high number of vertices to adhere to various requirements (visual, practical, physical, etc.). A simpler representation of the same object in

DRAFT

only a few parameters that manipulate the whole in a sensible matter are desirable, as this often decreases the computation time significantly.

Additionally one can exploit the fact, that drag in this case is especially sensitive to non–smooth surfaces, so that a smooth local manipulation of the surface as a whole is more advantageous than merely random manipulation of the vertices.

The quality of such a low-dimensional representation in biological evolution is strongly tied to the notion of *evolvability*[2], as the parametrization of the problem has serious implications on the convergence speed and the quality of the solution[3]. However, there is no consensus on how *evolvability* is defined and the meaning varies from context to context[4], so there is need for some criteria we can measure, so that we are able to compare different representations to learn and improve upon these.

One example of such a general representation of an object is to generate random points and represent vertices of an object as distances to these points — for example via Radial Basis Function (RBF). If one (or the algorithm) would move such a point the object will get deformed locally (due to the RBF). As this results in a simple mapping from the parameter-space onto the object one can try out different representations of the same object and evaluate the *evolvability*. This is exactly what Richter et al.[5] have done.

As we transfer the results of Richter et al.[5] from using Radial Basis Function (RBF) as a representation to manipulate geometric objects to the use of Freeform–Deformation (FFD) we will use the same definition for evolvability the original author used, namely *regularity*, *variability*, and *improvement potential*. We introduce these term in detail in Chapter 2.4. In the original publication the author could show a correlation between these evolvability–criteria with the quality and convergence speed of such optimization.

We will replicate the same setup on the same objects but use Freeform–Deformation (FFD) instead of Radial Basis Function (RBF) to create a local deformation near the control points and evaluate if the evolution–criteria still work as a predictor for *evolvability* of the representation given the different deformation scheme, as suspected in [5].

DRAFT

First we introduce different topics in isolation in Chapter 2. We take an abstract look at the definition of FFD for a one–dimensional line (in 2.1) and discuss why this is a sensible deformation function (in 2.1.1). Then we establish some background–knowledge of evolutionary algorithms (in 2.2) and why this is useful in our domain (in 2.3). In a third step we take a look at the definition of the different evolvability criteria established in [5].

In Chapter 3 we take a look at our implementation of FFD and the adaptation for 3D–meshes that were used.

Next, in Chapter 4, we describe the different scenarios we use to evaluate the different evolvability–criteria incorporating all aspects introduced in Chapter 2. Following that, we evaluate the results in Chapter 5 with further on discussion in Chapter 6.

DRAFT

DRAFT

2 Background

2.1 What is Freeform–Deformation (FFD)?

First of all we have to establish how a FFD works and why this is a good tool for deforming geometric objects (esp. meshes in our case) in the first place. For simplicity we only summarize the 1D–case from [6] here and go into the extension to the 3D case in chapter 3.2.

The main idea of FFD is to create a function $s : [0,1[^d \mapsto \mathbb{R}^d$ that spans a certain part of a vector–space and is only linearly parametrized by some special control points p_i and an constant attribution–function $a_i(u)$, so

$$s(u) = \sum_i a_i(u) p_i$$

can be thought of a representation of the inside of the convex hull generated by the control points where each point can be accessed by the right $u \in [0,1[$.

In the example in figure 2.1, the control–points are indicated as red dots and the color-gradient should hint at the u–values ranging from 0 to 1.

We now define a Freeform–Deformation (FFD) by the following:
Given an arbitrary number of points p_i alongside a line, we map a scalar value $\tau_i \in [0,1[$

DRAFT

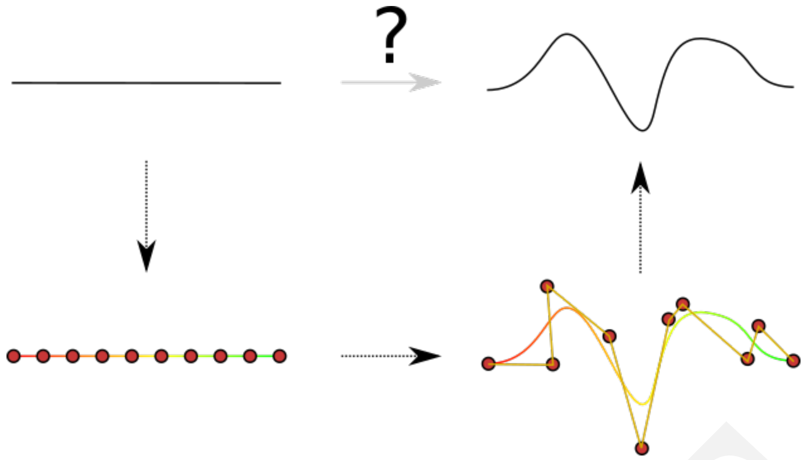

Figure 2.1: Example of a parametrization of a line with corresponding deformation to generate a deformed objet

to each point with $\tau_i < \tau_{i+1} \forall i$ according to the position of p_i on said line. Additionally, given a degree of the target polynomial d we define the curve $N_{i,d,\tau_i}(u)$ as follows:

$$N_{i,0,\tau}(u) = \begin{cases} 1, & u \in [\tau_i, \tau_{i+1}[\\ 0, & \text{otherwise} \end{cases} \tag{2.1}$$

and

$$N_{i,d,\tau}(u) = \frac{u - \tau_i}{\tau_{i+d}} N_{i,d-1,\tau}(u) + \frac{\tau_{i+d+1} - u}{\tau_{i+d+1} - \tau_{i+1}} N_{i+1,d-1,\tau}(u) \tag{2.2}$$

If we now multiply every p_i with the corresponding $N_{i,d,\tau_i}(u)$ we get the contribution of each point p_i to the final curve–point parameterized only by $u \in [0,1[$. As can be seen from (2.2) we only access points $[p_i..p_{i+d}]$ for any given i[1], which gives us, in combination with choosing p_i and τ_i in order, only a local interference of $d + 1$ points.

We can even derive this equation straightforward for an arbitrary N[2]:

---

[1] one more for each recursive step.

[2] *Warning:* in the case of $d = 1$ the recursion–formula yields a 0 denominator, but N is also 0. The right solution for this case is a derivative of 0

DRAFT

$$\frac{\partial}{\partial u} N_{i,d,r}(u) = \frac{d}{\tau_{i+d} - \tau_i} N_{i,d-1,\tau}(u) - \frac{d}{\tau_{i+d+1} - \tau_{i+1}} N_{i+1,d-1,\tau}(u)$$

For a B–Spline

$$s(u) = \sum_i N_{i,d,\tau_i}(u) p_i$$

these derivations yield $\frac{\partial^d}{\partial u} s(u) = 0$.

Another interesting property of these recursive polynomials is that they are continuous (given $d \geq 1$) as every p_i gets blended in between τ_i and τ_{i+d} and out between τ_{i+1}, and τ_{i+d+1} as can bee seen from the two coefficients in every step of the recursion.

This means that all changes are only a local linear combination between the control–point p_i to p_{i+d+1} and consequently this yields to the convex–hull–property of B-Splines — meaning, that no matter how we choose our coefficients, the resulting points all have to lie inside convex–hull of the control–points.

For a given point v_i we can then calculate the contributions $n_{i,j} := N_{j,d,\tau}$ of each control point p_j to get the projection from the control–point–space into the object–space:

$$v_i = \sum_j n_{i,j} \cdot p_j = \mathbf{n}_i^T \mathbf{p}$$

or written for all points at the same time:

$$\mathbf{v} = \mathbf{N}\mathbf{p}$$

where $\mathbf{N}$ is the $n \times m$ transformation–matrix (later on called **deformation matrix**) for n object–space–points and m control–points.

Furthermore B–splines–basis–functions form a partition of unity for all, but the first and last d control-points[7]. Therefore we later on use the border-points $d+1$ times, such that $\sum_j n_{i,j} p_j = p_i$ for these points.

DRAFT

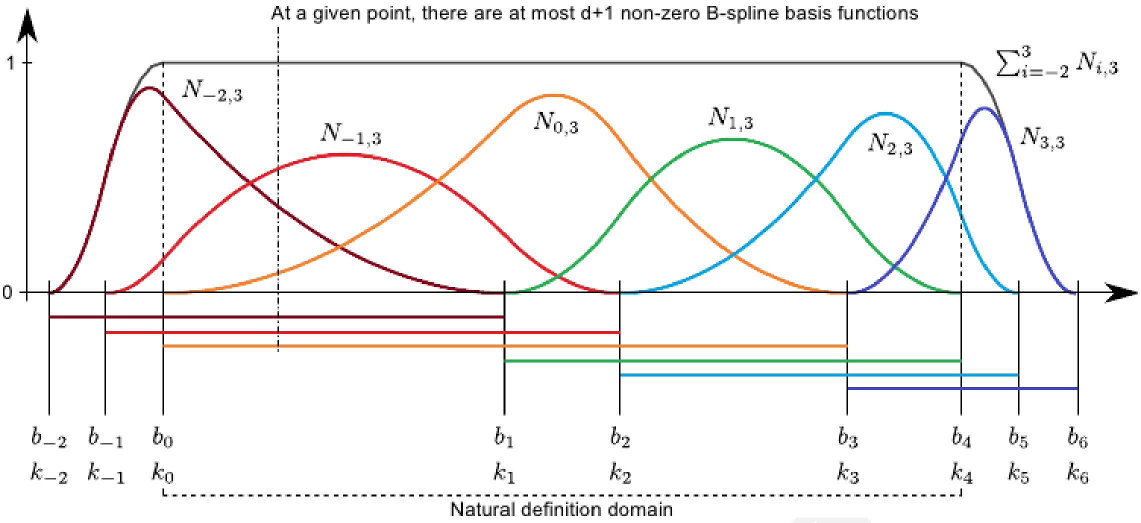

Figure 2.2: From [7, Figure 2.13]:

„Some interesting properties of the B–splines. On the natural definition domain of the B–spline ($[k_0, k_4]$ on this figure), the B–spline basis functions sum up to one (partition of unity). In this example, we use B–splines of degree 2. The horizontal segment below the abscissa axis represents the domain of influence of the B–splines basis function, i.e. the interval on which they are not null. At a given point, there are at most $d+1$ non-zero B–spline basis functions (compact support).“
Note, that Brunet starts his index at $-d$ opposed to our definition, where we start at 0.

The locality of the influence of each control–point and the partition of unity was beautifully pictured by Brunet, which we included here as figure 2.2.

2.1.1 Why is FFD a good deformation function?

The usage of FFD as a tool for manipulating follows directly from the properties of the polynomials and the correspondence to the control points. Having only a few control points gives the user a nicer high–level–interface, as she only needs to move these points and the model follows in an intuitive manner. The deformation is smooth as the underlying polygon is smooth as well and affects as many vertices of the model as needed. Moreover the changes are always local so one risks not any change that a user cannot immediately see.

But there are also disadvantages of this approach. The user loses the ability to directly

DRAFT

influence vertices and even seemingly simple tasks as creating a plateau can be difficult to achieve[8, chapter 3.2][9].

This disadvantages led to the formulation of Direct Manipulation Freeform–Deformation (DM–FFD)[8, chapter 3.3] in which the user directly interacts with the surface–mesh. All interactions will be applied proportionally to the control–points that make up the parametrization of the interaction–point itself yielding a smooth deformation of the surface *at* the surface without seemingly arbitrary scattered control–points. Moreover this increases the efficiency of an evolutionary optimization[10], which we will use later on.

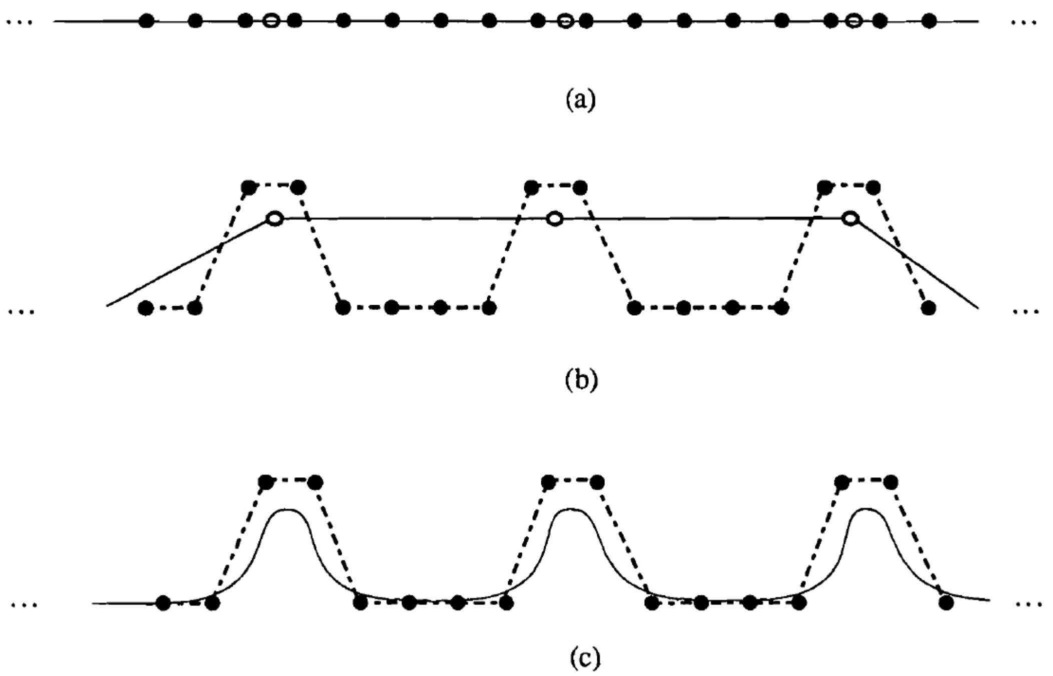

Figure 7. Aliasing because of too many control points relative to the number of object points. Open circles denote object points, filled circles denote control points. No control polygon is drawn. (a) Initial condition. (b) Resulting control-point location when the three object points are moved upward. (c) Possible aliasing if the resolution of the object is dramatically increased.

Figure 2.3: Figure 7 from [8].

But this approach also has downsides as can be seen in figure 2.3, as the tessellation of the invisible grid has a major impact on the deformation itself.

All in all FFD and DM–FFD are still good ways to deform a high–polygon mesh albeit the downsides.

2.2 What is evolutionary optimization?

In this thesis we are using an evolutionary optimization strategy to solve the problem of finding the best parameters for our deformation. This approach, however, is very generic and we introduce it here in a broader sense.

Algorithm 1 An outline of evolutionary algorithms

```
t := 0;
initialize $P(0) := \{\mathbf{a}_1(0), \ldots, \mathbf{a}_\mu(0)\} \in I^\mu$;
evaluate $F(0) : \{\Phi(x) | x \in P(0)\}$;
while $c(F(t)) \neq$ true do
    recombine: $P(t) := r(P(t))$;
    mutate: $P''(t) := m(P(t))$;
    evaluate $F''(t) : \{\Phi(x) | x \in P''(t)\}$
    select: $P(t+1) := s(P''(t) \cup Q, \Phi)$;
    t := t + 1;
```

The general shape of an evolutionary algorithm (adapted from [11]) is outlined in Algorithm 1. Here, $P(t)$ denotes the population of parameters in step t of the algorithm. The population contains μ individuals a_i from the possible individual–set I that fit the shape of the parameters we are looking for. Typically these are initialized by a random guess or just zero. Further on we need a so–called *fitness–function* $\Phi : I \mapsto M$ that can take each parameter to a measurable space M (usually $M = \mathbb{R}$) along a convergence–function $c : I \mapsto \mathbb{B}$ that terminates the optimization.

Biologically speaking the set I corresponds to the set of possible *Genotypes* while M represents the possible observable *Phenotypes*.

The main algorithm just repeats the following steps:

- **Recombine** with a recombination–function $r : I^\mu \mapsto I^\lambda$ to generate λ new individuals based on the characteristics of the μ parents.

DRAFT

This makes sure that the next guess is close to the old guess.

- **Mutate** with a mutation–function $m : I^\lambda \mapsto I^\lambda$ to introduce new effects that cannot be produced by mere recombination of the parents.
  Typically this just adds minor defects to individual members of the population like adding a random gaussian noise or amplifying/dampening random parts.
- **Selection** takes a selection–function $s : (I^\lambda \cup I^{\mu+\lambda}, \Phi) \mapsto I^\mu$ that selects from the previously generated I^λ children and optionally also the parents (denoted by the set Q in the algorithm) using the fitness–function Φ. The result of this operation is the next Population of μ individuals.

All these functions can (and mostly do) have a lot of hidden parameters that can be changed over time. One can for example start off with a high mutation–rate that cools off over time (i.e. by lowering the variance of a gaussian noise).

2.3 Advantages of evolutionary algorithms

The main advantage of evolutionary algorithms is the ability to find optima of general functions just with the help of a given fitness–function. With this most problems of simple gradient–based procedures, which often target the same error–function which measures the fitness, as an evolutionary algorithm, but can easily get stuck in local optima.

Components and techniques for evolutionary algorithms are specifically known to help with different problems arising in the domain of optimization[12]. An overview of the typical problems are shown in figure 2.4.

Most of the advantages stem from the fact that a gradient–based procedure has only one point of observation from where it evaluates the next steps, whereas an evolutionary strategy starts with a population of guessed solutions. Because an evolutionary strategy modifies the solution randomly, keeps the best solutions and purges the worst, it can also target

DRAFT

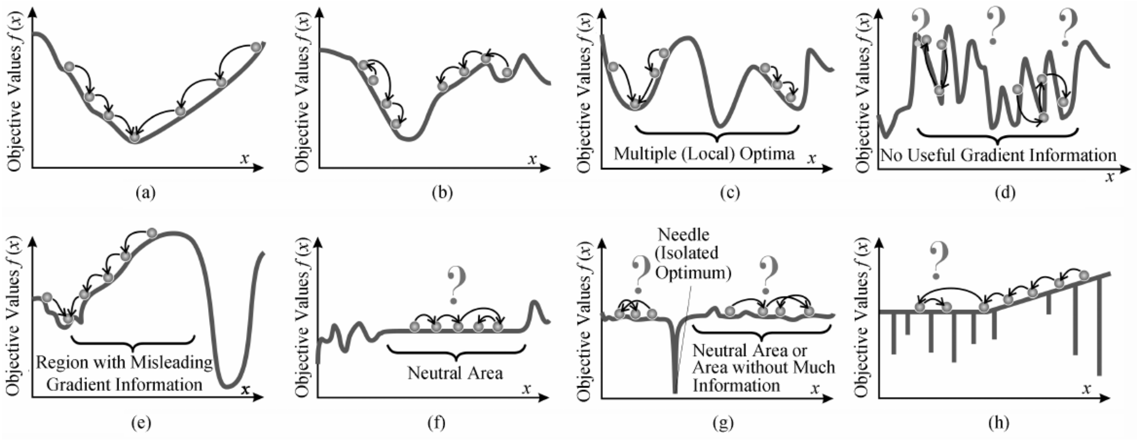

Fig.3. Examples of different possible scenarios in the fitness landscape (under minimization). (a) Best case. (b) Multi-modal with low total variation. (c) Multi-modal with higher total variation. (d) Rugged (multi-modal + high total variation). (e) Deceptive. (f) Neutral. (g) Needle-in-a-haystack. (h) Nightmare.

Figure 2.4: Fig. 3. taken from [12]

multiple different hypothesis at the same time where the local optima die out in the face of other, better candidates.

If an analytic best solution exists and is easily computable (i.e. because the error–function is convex) an evolutionary algorithm is not the right choice. Although both converge to the same solution, the analytic one is usually faster.

But in reality many problems have no analytic solution, because the problem is either not convex or there are so many parameters that an analytic solution (mostly meaning the equivalence to an exhaustive search) is computationally not feasible. Here evolutionary optimization has one more advantage as you can at least get suboptimal solutions fast, which then refine over time.

DRAFT

2.4 Criteria for the evolvability of linear deformations

As we have established in chapter 2.1, we can describe a deformation by the formula

$$V = UP$$

where V is a $n \times d$ matrix of vertices, U are the (during parametrization) calculated deformation–coefficients and P is a $m \times d$ matrix of control–points that we interact with during deformation.

We can also think of the deformation in terms of differences from the original coordinates

$$\Delta V = U \cdot \Delta P$$

which is isomorphic to the former due to the linear correlation in the deformation. One can see in this way, that the way the deformation behaves lies solely in the entries of U, which is why the three criteria focus on this.

DRAFT

2.4.1 Variability

In [5] *variability* is defined as

$$V(\mathbf{U}) := \frac{\operatorname{rank}(\mathbf{U})}{n},$$

whereby $\mathbf{U}$ is the $n \times m$ deformation–Matrix used to map the m control points onto the n vertices.

Unsure: Nicht $(n \cdot d) \times m$? Wegen u,v,w?

Given $n = m$, an identical number of control–points and vertices, this quotient will be $= 1$ if all control points are independent of each other and the solution is to trivially move every control–point onto a target–point.

In praxis the value of $V(\mathbf{U})$ is typically $\ll 1$, because as there are only few control–points for many vertices, so $m \ll n$.

2.4.2 Regularity

Regularity is defined[5] as

$$R(\mathbf{U}) := \frac{1}{\kappa(\mathbf{U})} = \frac{\sigma_{min}}{\sigma_{max}}$$

where σ_{min} and σ_{max} are the smallest and greatest right singular value of the deformation–matrix $\mathbf{U}$.

As we deform the given Object only based on the parameters as $\mathbf{p} \mapsto f(\mathbf{x} + \mathbf{U}\mathbf{p})$ this makes sure that $\|\mathbf{U}\mathbf{p}\| \propto \|\mathbf{p}\|$ when $\kappa(\mathbf{U}) \approx 1$. The inversion of $\kappa(\mathbf{U})$ is only performed to map the criterion–range to $[0..1]$, whereas 1 is the optimal value and 0 is the worst value.

On the one hand this criterion should be characteristic for numeric stability[13, chapter 2.7] and on the other hand for the convergence speed of evolutionary algorithms[5] as it is tied to the notion of locality[12, 14].

2.4.3 Improvement Potential

In contrast to the general nature of *variability* and *regularity*, which are agnostic of the fitness–function at hand the third criterion should reflect a notion of potential.

As during optimization some kind of gradient g is available to suggest a direction worth pursuing we use this to guess how much change can be achieved in the given direction.

The definition for an *improvement potential* P is[5]:

$$P(\mathbf{U}) := 1 - \|(\mathbf{1} - \mathbf{U}\mathbf{U}^+)(G)\|^2_F$$

given some approximate $n \times d$ fitness–gradient $\mathbf{G}$, normalized to $\|\mathbf{G}\|_F = 1$, whereby $\| \cdot \|_F$ denotes the Frobenius–Norm.

DRAFT

3 Implementation of Freeform–Deformation (FFD)

The general formulation of B–Splines has two free parameters d and τ which must be chosen beforehand.

As we usually work with regular grids in our FFD we define τ statically as $\tau_i = {}^{i}/_{n}$ whereby n is the number of control–points in that direction.

d defines the *degree* of the B–Spline–Function (the number of times this function is differentiable) and for our purposes we fix d to 3, but give the formulas for the general case so it can be adapted quite freely.

3.1 Adaption of FFD

As we have established in Chapter 2.1 we can define an FFD–displacement as

$$\Delta_x(u) = \sum_i N_{i,d,\tau_i}(u) \Delta_x c_i \tag{3.1}$$

Note that we only sum up the Δ–displacements in the control points c_i to get the change in position of the point we are interested in.

DRAFT

In this way every deformed vertex is defined by

$$\text{Deform}(v_x) = v_x + \Delta_x(u)$$

with $u \in [0..1[$ being the variable that connects the high–detailed vertex–mesh to the low–detailed control–grid. To actually calculate the new position of the vertex we first have to calculate the u–value for each vertex. This is achieved by finding out the parametrization of v in terms of c_i

$$v_x \overset{!}{=} \sum_i N_{i,d,\tau_i}(u) c_i$$

so we can minimize the error between those two:

$$\underset{u}{\text{argmin}}\, Err(u,v_x) = \underset{u}{\text{argmin}}\, 2 \cdot \|v_x - \sum_i N_{i,d,\tau_i}(u) c_i\|_2^2$$

As this error–term is quadratic we just derive by u yielding

$$\begin{aligned} & \tfrac{\partial}{\partial u} \quad v_x - \textstyle\sum_i N_{i,d,\tau_i}(u) c_i \\ = & \quad -\textstyle\sum_i \left(\tfrac{d}{\tau_{i+d} - \tau_i} N_{i,d-1,\tau}(u) - \tfrac{d}{\tau_{i+d+1} - \tau_{i+1}} N_{i+1,d-1,\tau}(u) \right) c_i \end{aligned}$$

and do a gradient–descend to approximate the value of u up to an ε of 0.0001.

For this we use the Gauss–Newton algorithm[15] as the solution to this problem may not be deterministic, because we usually have way more vertices than control points ($\#v \gg \#c$).

3.2 Adaption of FFD for a 3D–Mesh

This is a straightforward extension of the 1D–method presented in the last chapter. But this time things get a bit more complicated. As we have a 3–dimensional grid we may have a different amount of control–points in each direction.

DRAFT

Given n,m,o control points in x,y,z–direction each Point on the curve is defined by

$$V(u,v,w) = \sum_i \sum_j \sum_k N_{i,d,\tau_i}(u) N_{j,d,\tau_j}(v) N_{k,d,\tau_k}(w) \cdot C_{ijk}.$$

In this case we have three different B–Splines (one for each dimension) and also 3 variables u,v,w for each vertex we want to approximate.

Given a target vertex $\mathbf{p}^*$ and an initial guess $\mathbf{p} = V(u,v,w)$ we define the error–function for the gradient–descent as:

$$Err(u,v,w,\mathbf{p}^*) = \mathbf{p}^* - V(u,v,w)$$

And the partial version for just one direction as

$$Err_x(u,v,w,\mathbf{p}^*) = p^*_x - \sum_i \sum_j \sum_k N_{i,d,\tau_i}(u) N_{j,d,\tau_j}(v) N_{k,d,\tau_k}(w) \cdot c_{ijk_x}$$

To solve this we derive partially, like before:

$$\begin{aligned} \frac{\partial Err_x}{\partial u} & \quad p^*_x - \sum_i \sum_j \sum_k N_{i,d,\tau_i}(u) N_{j,d,\tau_j}(v) N_{k,d,\tau_k}(w) \cdot c_{ijk_x} \\ &= \quad - \sum_i \sum_j \sum_k N'_{i,d,\tau_i}(u) N_{j,d,\tau_j}(v) N_{k,d,\tau_k}(w) \cdot c_{ijk_x} \end{aligned}$$

The other partial derivatives follow the same pattern yielding the Jacobian:

$$J(Err(u,v,w)) = \begin{pmatrix} \frac{\partial Err_x}{\partial u} & \frac{\partial Err_x}{\partial v} & \frac{\partial Err_x}{\partial w} \\ \frac{\partial Err_y}{\partial u} & \frac{\partial Err_y}{\partial v} & \frac{\partial Err_y}{\partial w} \\ \frac{\partial Err_z}{\partial u} & \frac{\partial Err_z}{\partial v} & \frac{\partial Err_z}{\partial w} \end{pmatrix}$$

DRAFT

$$= \begin{pmatrix} -\sum\limits_{i,j,k} N'_i(u)N_j(v)N_k(w) \cdot c_{ijk_x} & -\sum\limits_{i,j,k} N_i(u)N'_j(v)N_k(w) \cdot c_{ijk_x} & -\sum\limits_{i,j,k} N_i(u)N_j(v)N'_k(w) \cdot c_{ijk_x} \\ -\sum\limits_{i,j,k} N'_i(u)N_j(v)N_k(w) \cdot c_{ijk_y} & -\sum\limits_{i,j,k} N_i(u)N'_j(v)N_k(w) \cdot c_{ijk_y} & -\sum\limits_{i,j,k} N_i(u)N_j(v)N'_k(w) \cdot c_{ijk_y} \\ -\sum\limits_{i,j,k} N'_i(u)N_j(v)N_k(w) \cdot c_{ijk_z} & -\sum\limits_{i,j,k} N_i(u)N'_j(v)N_k(w) \cdot c_{ijk_z} & -\sum\limits_{i,j,k} N_i(u)N_j(v)N'_k(w) \cdot c_{ijk_z} \end{pmatrix}$$

With the Gauss–Newton algorithm we iterate via the formula

$$J(Err(u,v,w)) \cdot \Delta \begin{pmatrix} u \\ v \\ w \end{pmatrix} = -Err(u,v,w)$$

and use Cramers rule for inverting the small Jacobian and solving this system of linear equations.

3.3 Deformation Grid

As mentioned in chapter 2.2, the way of choosing the representation to map the general problem (mesh–fitting/optimization in our case) into a parameter-space it very important for the quality and runtime of evolutionary algorithms[3].

Because our control–points are arranged in a grid, we can accurately represent each vertex–point inside the grids volume with proper B–Spline–coefficients between $[0,1[$ and — as a consequence — we have to embed our object into it (or create constant “dummy”-points outside).

The great advantage of B–Splines is the locality, direct impact of each control point without having a $1:1$–correlation, and a smooth deformation. While the advantages are great, the issues arise from the problem to decide where to place the control–points and how many.

One would normally think, that the more control–points you add, the better the result will be, but this is not the case for our B–Splines. Given any point p only the $2 \cdot (d-1)$ control–

DRAFT

points contribute to the parametrization of that point[1]. This means, that a high resolution can have many control-points that are not contributing to any point on the surface and are thus completely irrelevant to the solution.

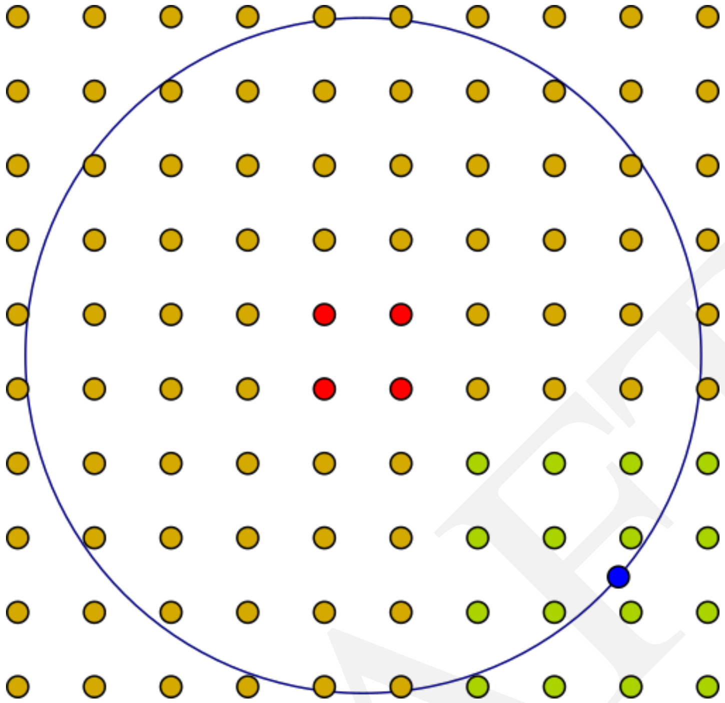

Figure 3.1: A high resolution (10×10) of control–points over a circle. Yellow/green points contribute to the parametrization, red points don't.
An Example–point (blue) is solely determined by the position of the green control–points.

We illustrate this phenomenon in figure 3.1, where the four red central points are not relevant for the parametrization of the circle.

Unsure: erwähnen, dass man aus **D** einfach die Null–Spalten entfernen kann?

For our tests we chose different uniformly sized grids and added noise onto each control-point[2] to simulate different starting-conditions.

Unsure: verweis auf DM–FFD?

DRAFT

---

[1] Normally these are $d - 1$ to each side, but at the boundaries the number gets increased to the inside to meet the required smoothness

[2] For the special case of the outer layer we only applied noise away from the object, so the object is still confined in the convex hull of the control–points.

DRAFT

4 Scenarios for testing evolvability criteria using Freeform–Deformation (FFD)

DRAFT

In our experiments we use the same two testing–scenarios, that were also used by [5]. The first scenario deforms a plane into a shape originally defined in [16], where we setup control-points in a 2–dimensional manner merely deform in the height–coordinate to get the resulting shape.

In the second scenario we increase the degrees of freedom significantly by using a 3–dimensional control–grid to deform a sphere into a face. So each control point has three degrees of freedom in contrast to first scenario.

4.1 Test Scenario: 1D Function Approximation

In this scenario we used the shape defined by Giannelli et al.[16], which is also used by Richter et al.[5] using the same discretization to 150×150 points for a total of $n = 22\,500$

vertices. The shape is given by the following definition

$$t(x,y) = \begin{cases} 0.5\cos(4\pi \cdot q^{0.5}) + 0.5 & q(x,y) < \frac{1}{16}, \\ 2(y-x) & 0 < y - x < 0.5, \\ 1 & 0.5 < y - x \end{cases}$$

with $(x,y) \in [0,2] \times [0,1]$ and $q(x,y) = (x-1.5)^2 + (y-0.5)^2$, which we have visualized in figure 4.1.

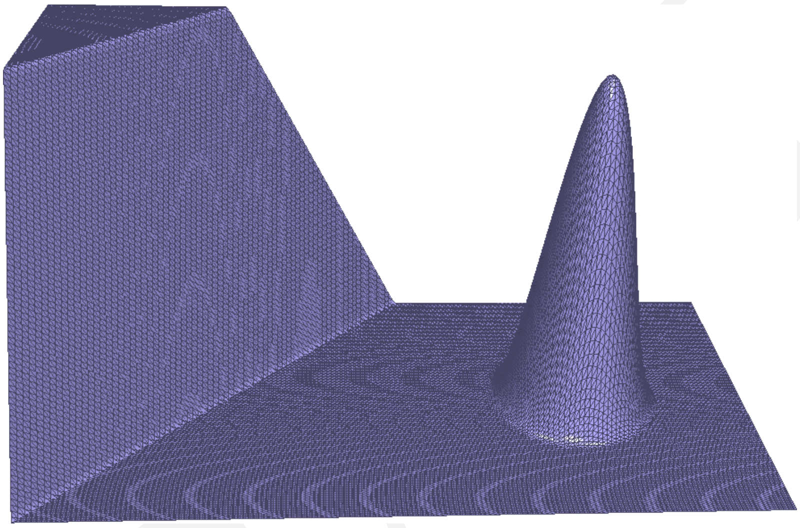

Figure 4.1: The target–shape for our 1–dimensional optimization–scenario including a wireframe–overlay of the vertices.

As the starting-plane we used the same shape, but set all z–coordinates to 0, yielding a flat plane, which is partially already correct.

Regarding the *fitness–function* $f(\mathbf{p})$, we use the very simple approach of calculating the squared distances for each corresponding vertex

$$\text{f(p)} = \sum_{i=1}^{n} \|(\mathbf{Up})_i - t_i\|_2^2 = \|\mathbf{Up} - \mathbf{t}\|^2 \rightarrow \min$$

where t_i are the respective target–vertices to the parametrized source–vertices[1] with the

---

[1] The parametrization is encoded in $\mathbf{U}$ and the initial position of the control points. See 3.1

DRAFT

current deformation–parameters $\mathbf{p} = (p_1, \ldots, p_m)$. We can do this one–to–one–correspondence because we have exactly the same number of source and target-vertices do to our setup of just flattening the object.

This formula is also the least–squares approximation error for which we can compute the analytic solution $\mathbf{p}^* = \mathbf{U}^+\mathbf{t}$, yielding us the correct gradient in which the evolutionary optimizer should move.

4.2 Procedure: 1D Function Approximation

For our setup we first compute the coefficients of the deformation–matrix and use then the formulas for *variability* and *regularity* to get our predictions. Afterwards we solve the problem analytically to get the (normalized) correct gradient that we use as guess for the *improvement potential*. To check we also consider a distorted gradient $\mathbf{g}_\mathrm{d}$

$$\mathbf{g}_\mathrm{d} = \frac{\mathbf{g}_\mathrm{c} + \mathbb{1}}{\|\mathbf{g}_\mathrm{c} + \mathbb{1}\|}$$

where $\mathbb{1}$ is the vector consisting of 1 in every dimension and $\mathbf{g}_\mathrm{c} = \mathbf{p}^*$ the calculated correct gradient.

DRAFT

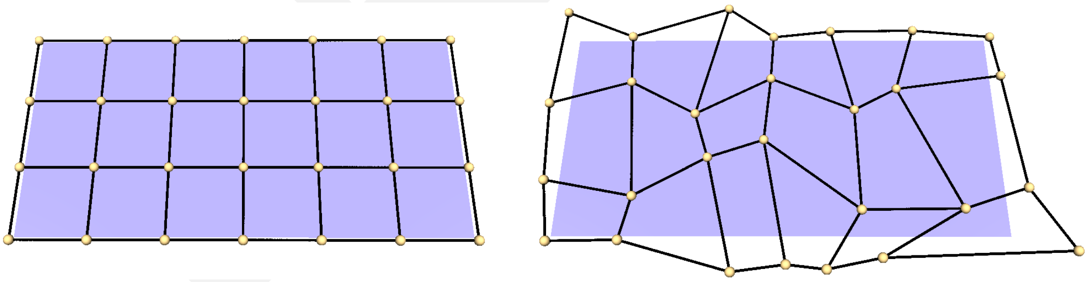

Figure 4.2:
Left: A regular 7×4–grid
Right: The same grid after a random distortion to generate a testcase.

We then set up a regular 2–dimensional grid around the object with the desired grid resolutions. To generate a testcase we then move the grid–vertices randomly inside the x–

y–plane. As self-intersecting grids get tricky to solve with our implemented newtons–method we avoid the generation of such self–intersecting grids for our testcases.

This is a reasonable thing to do, as self-intersecting grids violate our desired property of locality, as the then farther away control–point has more influence over some vertices as the next-closer.

To achieve that we select a uniform distributed number $r \in [-0.25, 0.25]$ per dimension and shrink the distance to the neighbours (the smaller neighbour for $r < 0$, the larger for $r > 0$) by the factor r^2.

Improvement: update!! gaussian, not uniform!!

An Example of such a testcase can be seen for a 7×4–grid in figure 4.2.

DRAFT

4.3 Test Scenario: 3D Function Approximation

Opposed to the 1–dimensional scenario before, the 3–dimensional scenario is much more complex — not only because we have more degrees of freedom on each control point, but also because the *fitness–function* we will use has no known analytic solution and multiple local minima.

First of all we introduce the set up: We have given a triangulated model of a sphere consisting of 10,807 vertices, that we want to deform into a the target–model of a face with a total of 12,024 vertices. Both of these Models can be seen in figure 4.3.

Opposed to the 1D–case we cannot map the source and target–vertices in a one–to–one–correspondence, which we especially need for the approximation of the fitting–error. Hence we state that the error of one vertex is the distance to the closest vertex of the other model.

---

[2] Note: On the Edges this displacement is only applied outwards by flipping the sign of r, if appropriate.

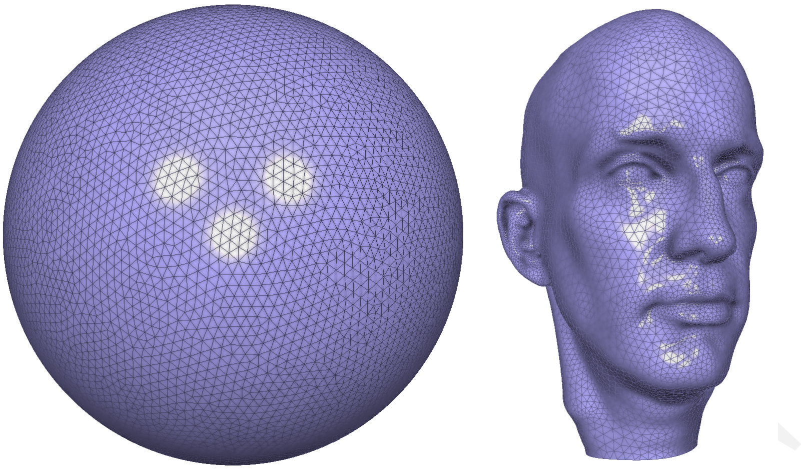

Figure 4.3:
Left: The sphere we start from with 10 807 vertices
Right: The face we want to deform the sphere into with 12 024 vertices.

We therefore define the *fitness–function* to be:

$$f(\mathbf{P}) = \frac{1}{n}\underbrace{\sum_{i=1}^{n} \|\mathbf{c_T}(\mathbf{s_i}) - \mathbf{s_i}\|_2^2}_{\text{source-to-target–distance}} + \frac{1}{m}\underbrace{\sum_{i=1}^{m} \|\mathbf{c_S}(\mathbf{t_i}) - \mathbf{t_i}\|_2^2}_{\text{target-to-source–distance}} + \lambda \cdot \text{regularization}(\mathbf{P})$$

where $\mathbf{c_T}(\mathbf{s_i})$ denotes the target–vertex that is corresponding to the source–vertex $\mathbf{s_i}$ and $\mathbf{c_S}(\mathbf{t_i})$ denotes the source–vertex that corresponds to the target–vertex $\mathbf{t_i}$. Note that the target–vertices are given and fixed by the target–model of the face we want to deform into, whereas the source–vertices vary depending on the chosen parameters $\mathbf{P}$, as those get calculated by the previously introduces formula $\mathbf{S} = \mathbf{UP}$ with $\mathbf{S}$ being the $n \times 3$–matrix of source–vertices, $\mathbf{U}$ the $n \times m$–matrix of calculated coefficients for the FFD — analog to the 1D case — and finally $\mathbf{P}$ being the $m \times 3$–matrix of the control–grid defining the whole deformation.

As regularization-term we add a weighted Laplacian of the deformation that has been used before by Aschenbach et al.[17, Section 3.2] on similar models and was shown to

DRAFT

lead to a more precise fit. The Laplacian

$$\text{regularization}(\mathbf{P}) = \frac{1}{\sum_i A_i} \sum_{i=1}^{n} A_i \cdot \left(\sum_{\mathbf{s_j} \in \mathcal{N}(\mathbf{s_i})} w_j \cdot \|\Delta \mathbf{s_j} - \Delta \bar{\mathbf{s}}_{\mathbf{j}}\|^2 \right)$$

is determined by the cotangent weighted displacement w_j of the to s_i connected vertices $\mathcal{N}(s_i)$ and A_i is the Voronoi–area of the corresponding vertex $\mathbf{s_i}$. We leave out the $\mathbf{R}_i$–term from the original paper as our deformation is merely linear.

This regularization–weight gives us a measure of stiffness for the material that we will influence via the λ–coefficient to start out with a stiff material that will get more flexible per iteration.

Unsure: Andreas: hast du nen cite, wo gezeigt ist, dass das so sinnvoll ist?

4.4 Procedure: 3D Function Approximation

Initially we set up the correspondences $\mathbf{c_T}(\dots)$ and $\mathbf{c_S}(\dots)$ to be the respectively closest vertices of the other model. We then calculate the analytical solution given these correspondences via $\mathbf{P}^* = \mathbf{U}^+\mathbf{T}$, and also use the first solution as guessed gradient for the calculation of the *improvement–potential*, as the optimal solution is not known. We then let the evolutionary algorithm run up within 1.05 times the error of this solution and afterwards recalculate the correspondences $\mathbf{c_T}(\dots)$ and $\mathbf{c_S}(\dots)$. For the next step we then halve the regularization–impact λ and calculate the next incremental solution $\mathbf{P}^* = \mathbf{U}^+\mathbf{T}$ with the updated correspondences to get our next target–error. We repeat this process as long as the target–error keeps decreasing.

The grid we use for our experiments is just very coarse due to computational limitations. We are not interested in a good reconstruction, but an estimate if the mentioned evolvability criteria are good.

DRAFT

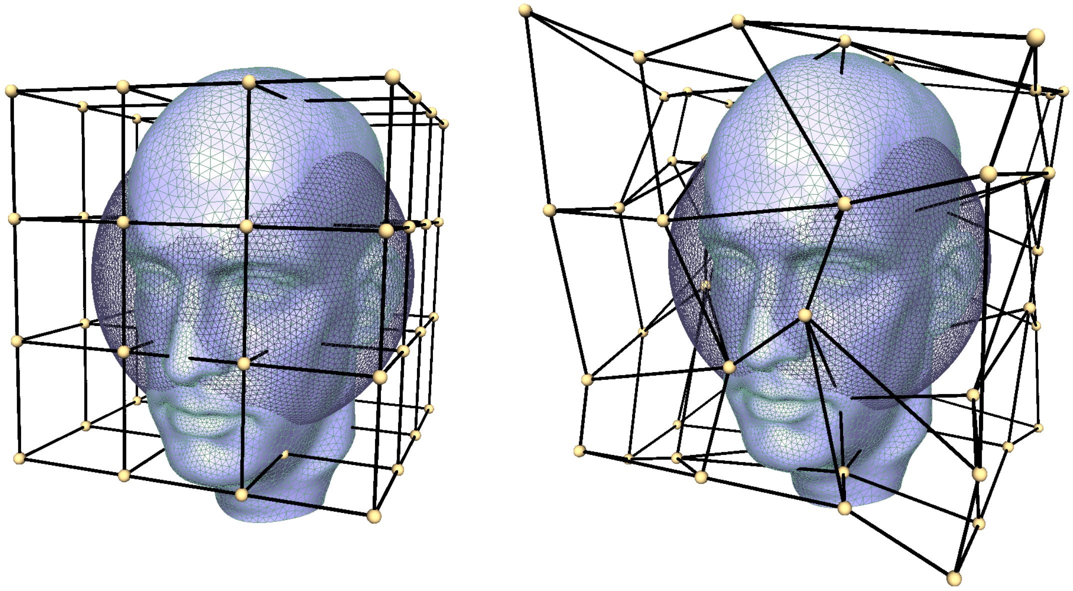

Figure 4.4:
Left: The 3D–setup with a $4 \times 4 \times 4$–grid.
Right: The same grid after added noise to the control–points.

In figure 4.4 we show an example setup of the scene with a $4 \times 4 \times 4$–grid. Identical to the 1–dimensional scenario before, we create a regular grid and move the control-points uniformly random between their neighbours, but in three instead of two dimensions[3].

As is clearly visible from figure 4.3, the target–model has many vertices in the facial area, at the ears and in the neck–region. Therefore we chose to increase the grid-resolutions for our tests in two different dimensions and see how well the criteria predict a suboptimal placement of these control-points.

DRAFT

---

[3] Again, we flip the signs for the edges, if necessary to have the object still in the convex hull.

DRAFT

5 Evaluation of Scenarios

To compare our results to the ones given by Richter et al.[5], we also use Spearman's rank correlation coefficient. Opposed to other popular coefficients, like the Pearson correlation coefficient, which measures a linear relationship between variables, the Spearmans's coefficient assesses „how well an arbitrary monotonic function can descripbe the relationship between two variables, without making any assumptions about the frequency distribution of the variables“[18].

As we don't have any prior knowledge if any of the criteria is linear and we are just interested in a monotonic relation between the criteria and their predictive power, the Spearman's coefficient seems to fit out scenario best.

For interpretation of these values we follow the same interpretation used in [5], based on [19]: The coefficient intervals $r_S \in [0{,}0.2[$, $[0.2{,}0.4[$, $[0.4{,}0.6[$, $[0.6{,}0.8[$, and $[0.8{,}1]$ are classified as *very weak*, *weak*, *moderate*, *strong* and *very strong*. We interpret p–values smaller than 0.1 as *significant* and cut off the precision of p–values after four decimal digits (thus often having a p–value of 0 given for p–values $< 10^{-4}$).

As we are looking for anti–correlation (i.e. our criterion should be maximized indicating a minimal result in — for example — the reconstruction–error) instead of correlation we flip the sign of the correlation–coefficient for readability and to have the correlation–coefficients be in the classification–range given above.

For the evolutionary optimization we employ the CMA–ES (covariance matrix adaptation

DRAFT

evolution strategy) of the shark3.1 library [20], as this algorithm was used by [5] as well. We leave the parameters at their sensible defaults as further explained in [21, Appendix A: Table 1].

5.1 Results of 1D Function Approximation

In the case of our 1D–Optimization–problem, we have the luxury of knowing the analytical solution to the given problem–set. We use this to experimentally evaluate the quality criteria we introduced before. As an evolutional optimization is partially a random process, we use the analytical solution as a stopping-criteria. We measure the convergence speed as number of iterations the evolutional algorithm needed to get within 1.05% of the optimal solution.

We used different regular grids that we manipulated as explained in Section 4.2 with a different number of control points. As our grids have to be the product of two integers, we compared a 5×5–grid with 25 control–points to a 4×7 and 7×4–grid with 28 control–points. This was done to measure the impact an „improper“setup could have and how well this is displayed in the criteria we are examining.

Additionally we also measured the effect of increasing the total resolution of the grid by taking a closer look at 5×5, 7×7 and 10×10 grids.

5.1.1 Variability

Variability should characterize the potential for design space exploration and is defined in terms of the normalized rank of the deformation matrix $\mathbf{U}$: $V(\mathbf{U}) := \frac{\text{rank}(\mathbf{U})}{n}$, whereby n is the number of vertices. As all our tested matrices had a constant rank (being $m = x \cdot y$ for a $x \times y$ grid), we have merely plotted the errors in the boxplot in figure 5.1

DRAFT

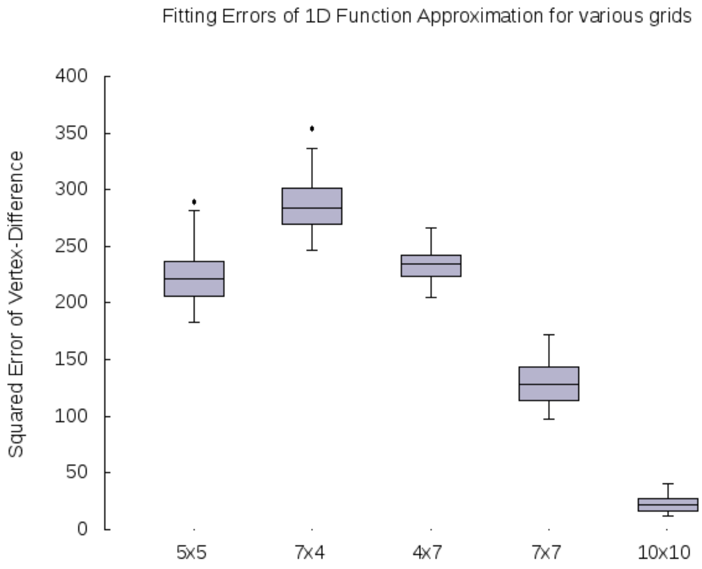

Figure 5.1: The squared error for the various grids we examined.
Note that 7×4 and 4×7 have the same number of control–points.

It is also noticeable, that although the 7×4 and 4×7 grids have a higher variability, they perform not better than the 5×5 grid. Also the 7×4 and 4×7 grids differ distinctly from each other, although they have the same number of control–points. This is an indication the impact a proper or improper grid–setup can have. We do not draw scientific conclusions from these findings, as more research on non-squared grids seem necessary.

machen wir die noch? :D

Leaving the issue of the grid–layout aside we focused on grids having the same number of prototypes in every dimension. For the 5×5, 7×7 and 10×10 grids we found a *very strong* correlation ($-r_S = 0.94, p = 0$) between the variability and the evolutionary error.

5.1.2 Regularity

5×5	7×4	4×7	7×7	10×10
0.28 (0.0045)	0.21 (0.0396)	0.1 (0.3019)	0.01 (0.9216)	0.01 (0.9185)

Table 5.1: Spearman's correlation (and p-values) between regularity and convergence speed for the 1D function approximation problem.
Not significant entries are marked in red.

Regularity should correspond to the convergence speed (measured in iteration–steps of

DRAFT

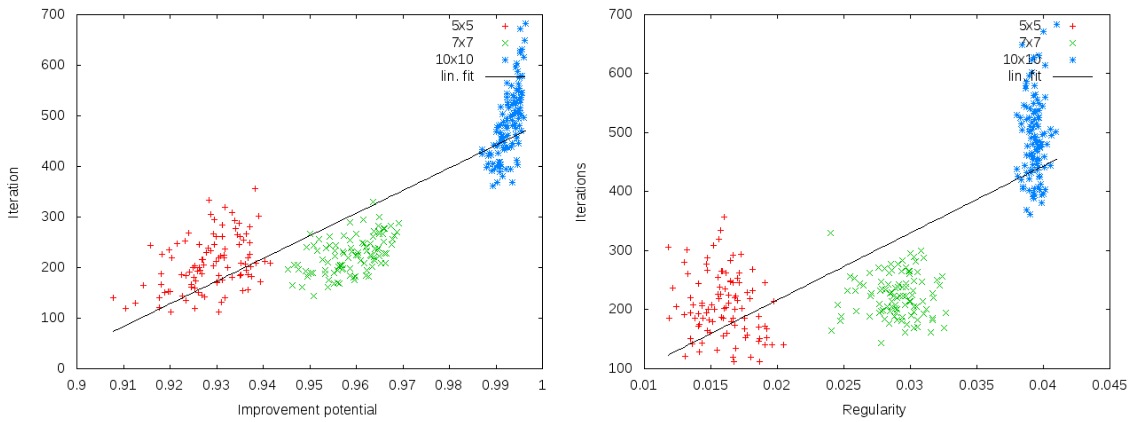

Figure 5.2:
Left: Improvement potential against steps until convergence
Right: Regularity against steps until convergence
Coloured by their grid–resolution, both with a linear fit over the whole dataset.

the evolutionary algorithm), and is computed as inverse condition number $\kappa(\mathbf{U})$ of the deformation–matrix.

As can be seen from table 5.1, we could only show a *weak* correlation in the case of a 5×5 grid. As we increment the number of control–points the correlation gets worse until it is completely random in a single dataset. Taking all presented datasets into account we even get a *strong* correlation of $-r_S = -0.72, p = 0$, that is opposed to our expectations.

To explain this discrepancy we took a closer look at what caused these high number of iterations. In figure 5.2 we also plotted the improvement-potential against the steps next to the regularity–plot. Our theory is that the *very strong* correlation $(-r_S = -0.82, p = 0)$ between improvement–potential and number of iterations hints that the employed algorithm simply takes longer to converge on a better solution (as seen in figure 5.3) offsetting any gain the regularity–measurement could achieve.

DRAFT

5.1.3 Improvement Potential

- Alle Spearman 1 und p-value 0.

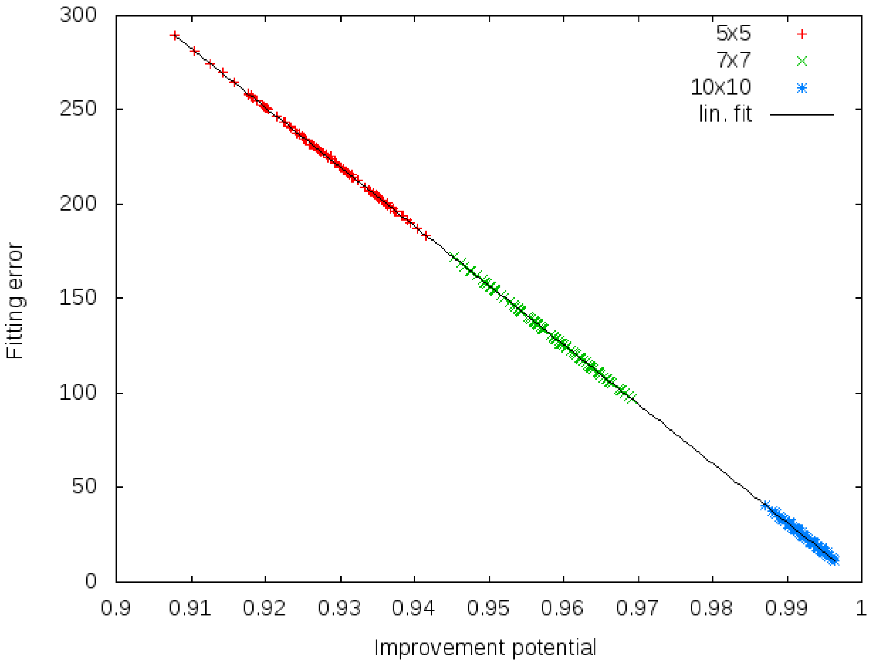

Figure 5.3: Improvement potential plotted against the error yielded by the evolutionary optimization for different grid–resolutions

5.2 Results of 3D Function Approximation

DRAFT

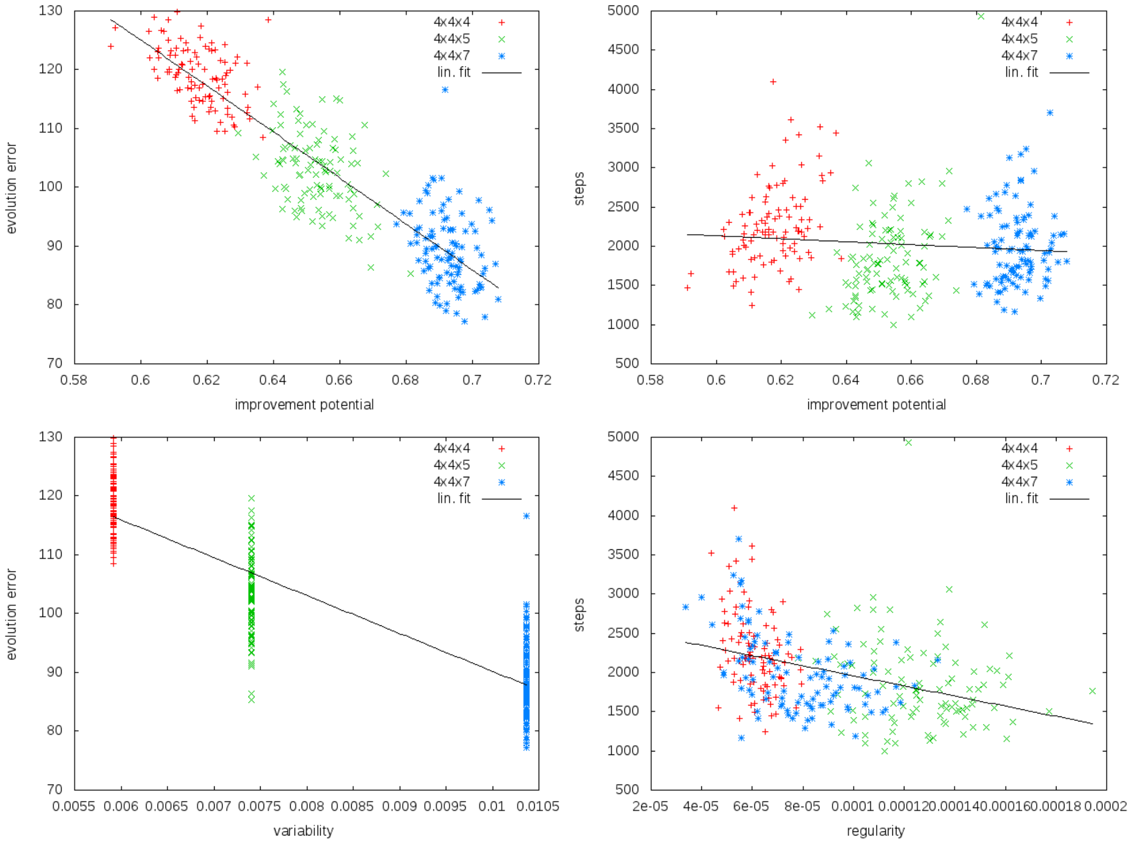

Figure 5.4: Results 3D for 4x4xX

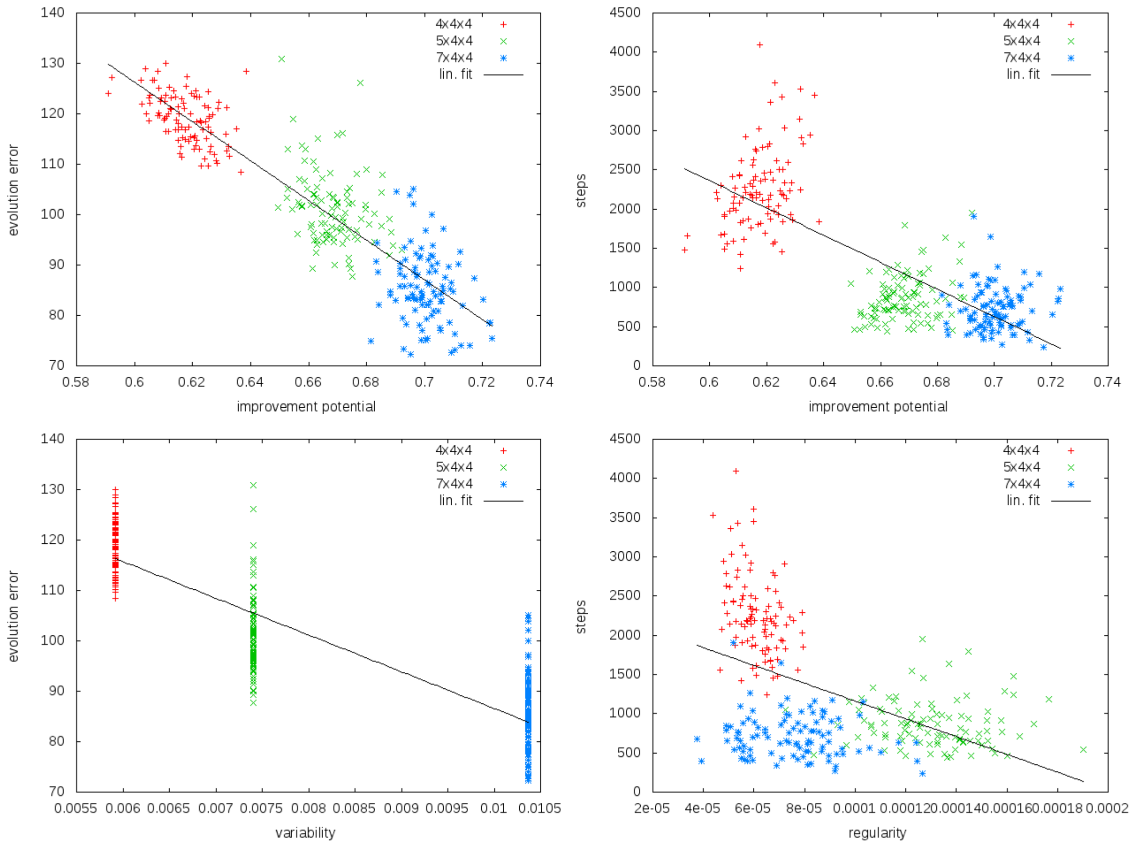

Figure 5.5: Results 3D for Xx4x4

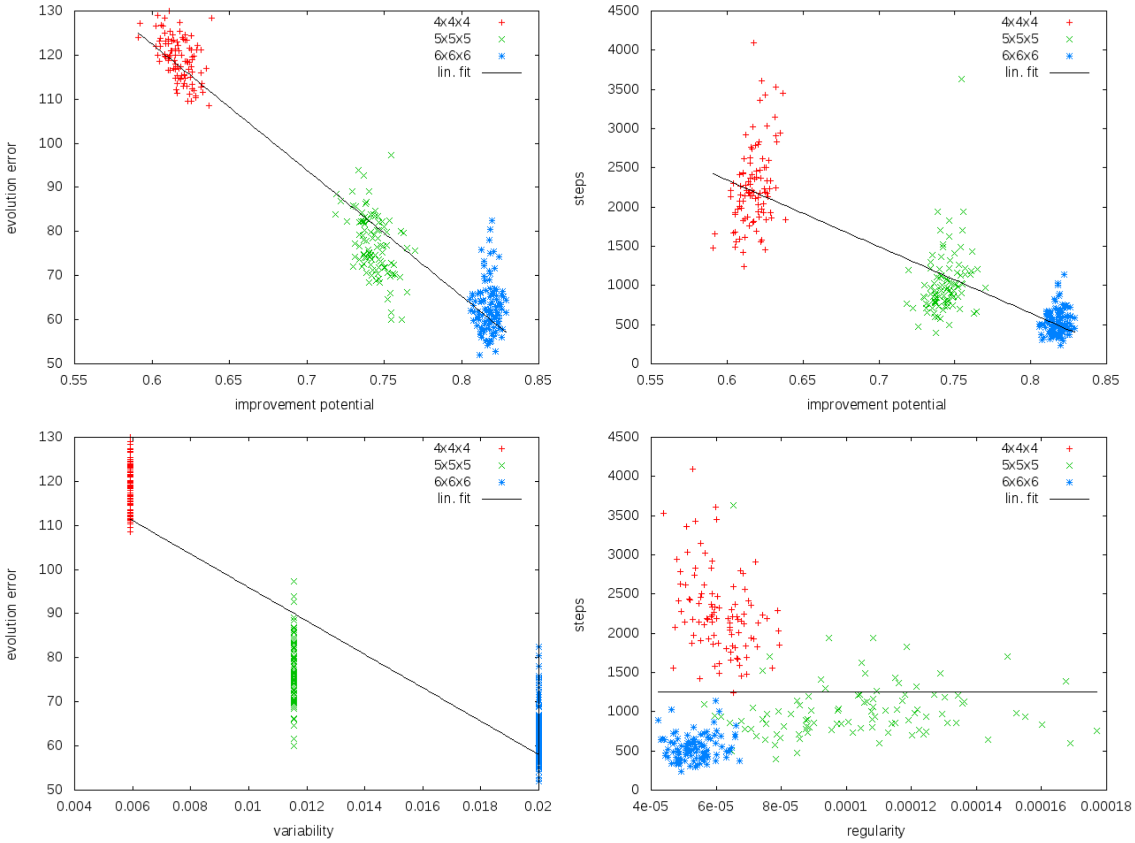

Figure 5.6: Results 3D for YxYxY for Y ∈ [4,5,6]

6 Schluss

- Regularity ist kacke für unser setup. Bessere Vorschläge? EW/EV?

Improvement: Bibliotheksverzeichnis links anpassen. DOI überschreibt Direktlinks des Autors.

DRAFT

DRAFT

Appendix

DRAFT

DRAFT

A Bibliography

[1] Minai, Ali A. ; Braha, Dan ; Bar-Yam, Yaneer: Complex engineered systems: A new paradigm. In: *Complex engineered systems: Science meets technology* (2006), 1–21. `https://www.researchgate.net/profile/Yaneer_Bar-Yam/publication/225104044_Complex_Engineered_Systems_A_New_Paradigm/links/59107f20a6fdccbfd57eb84d/Complex-Engineered-Systems-A-New-Paradigm.pdf`

[2] Wagner, Gunter P. ; Altenberg, Lee: Complex adaptations and the evolution of evolvability. In: *Evolution* 50 (1996), Nr. 3, 967–976. `http://arep.med.harvard.edu/pdf/Wagner96.pdf`

[3] *Kapitel* 2. In: Rothlauf, Franz: *Representations for Genetic and Evolutionary Algorithms*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2006. – ISBN 978–3–540–32444–7, 9–32

[4] Richter, Andreas ; Botsch, Mario ; Menzel, Stefan: Evolvability of representations in complex system engineering: a survey. In: *Evolutionary Computation (CEC), 2015 IEEE Congress on* IEEE, 2015, 1327–1335

[5] Richter, Andreas ; Achenbach, Jascha ; Menzel, Stefan ; Botsch, Mario: Evolvability as a Quality Criterion for Linear Deformation Representations in Evolutionary Optimization. (2016). – `http://graphics.uni-bielefeld.de/publications/cec16.pdf`, `https://pub.uni-bielefeld.de/publication/2902698`

DRAFT

[6] SPITZMÜLLER, Klaus: Partial derivatives of Bèzier surfaces. In: *Computer-Aided Design* 28 (1996), Nr. 1, 67–72. `https://doi.org/10.1016/0010-4485(95)00044-5`

[7] BRUNET, Florent: Contributions to parametric image registration and 3d surface reconstruction. In: *European Ph. D. in Computer Vision, Université dAuvergne, Clérmont-Ferrand, France, and Technische Universitat Munchen, Germany* (2010). `http://www.brnt.eu/phd/`

[8] HSU, William M.: A direct manipulation interface to free-form deformations. In: *Master's thesis, Brown University* (1991). `https://cs.brown.edu/research/pubs/theses/masters/1991/hsu.pdf`

[9] HSU, William M. ; HUGHES, John F. ; KAUFMAN, Henry: Direct Manipulation of Free-Form Deformations. In: *Computer Graphics* 26 (1992), 2. `http://graphics.cs.brown.edu/~jfh/papers/Hsu-DMO-1992/paper.pdf`

[10] MENZEL, Stefan ; OLHOFER, Markus ; SENDHOFF, Bernhard: Direct Manipulation of Free Form Deformation in Evolutionary Design Optimisation. In: *Proceedings of the 9th International Conference on Parallel Problem Solving from Nature*. Berlin, Heidelberg : Springer-Verlag, 2006 (PPSN'06). – ISBN 3–540–38990–3, 978–3–540–38990–3, 352–361

[11] BÄCK, Thomas ; SCHWEFEL, Hans-Paul: An overview of evolutionary algorithms for parameter optimization. In: *Evolutionary computation* 1 (1993), Nr. 1, 1–23. `https://www.researchgate.net/profile/Hans-Paul_Schwefel/publication/220375001_An_Overview_of_Evolutionary_Algorithms_for_Parameter_Optimization/links/543663d00cf2dc341db30452.pdf`

[12] WEISE, Thomas ; CHIONG, Raymond ; TANG, Ke: Evolutionary Optimization: Pitfalls and Booby Traps. In: *J. Comput. Sci. & Technol* 27

DRAFT

(2012), Nr. 5. `http://jcst.ict.ac.cn:8080/jcst/EN/article/downloadArticleFile.do?attachType=PDF&id=9543`

[13] GOLUB, Gene H. ; VAN LOAN, Charles F.: *Matrix computations*. Bd. 3. JHU Press, 2012

[14] THORHAUER, Ann ; ROTHLAUF, Franz: On the locality of standard search operators in grammatical evolution. In: *International Conference on Parallel Problem Solving from Nature* Springer, 2014, 465–475

[15] MARQUARDT, Donald W.: An Algorithm for Least-Squares Estimation of Nonlinear Parameters. In: *Journal of the Society for Industrial and Applied Mathematics* 11 (1963), Nr. 2, 431-441. `http://dx.doi.org/10.1137/0111030`. – DOI 10.1137/0111030

[16] GIANNELLI, Carlotta ; JÜTTLER, Bert ; SPELEERS, Hendrik: THB-splines: The truncated basis for hierarchical splines. In: *Computer Aided Geometric Design* 29 (2012), Nr. 7, 485–498. `http://dx.doi.org/10.1016/j.cagd.2012.03.025`. – DOI 10.1016/j.cagd.2012.03.025

[17] ACHENBACH, Jascha ; ZELL, Eduard ; BOTSCH, Mario: Accurate Face Reconstruction through Anisotropic Fitting and Eye Correction. In: *Proceedings of Vision, Modeling and Visualization* (2015), 1–8. `http://graphics.uni-bielefeld.de/publications/disclaimer.php?dlurl=vmv15.pdf`. ISBN 978–3–905674–95–8

[18] HAUKE, Jan ; KOSSOWSKI, Tomasz: Comparison of values of Pearson's and Spearman's correlation coefficients on the same sets of data. In: *Quaestiones geographicae* 30 (2011), Nr. 2, 87. `https://www.degruyter.com/downloadpdf/j/quageo.2011.30.issue-2/v10117-011-0021-1/v10117-011-0021-1.pdf`

[19] WEIR, I: Spearmans correlation. In: *Retrieved from statstutor* (2015). `http://www.statstutor.ac.uk/resources/uploaded/spearmans.pdf`

DRAFT

[20] IGEL, Christian ; HEIDRICH-MEISNER, Verena ; GLASMACHERS, Tobias: Shark. In: *Journal of Machine Learning Research* 9 (2008), 993-996. `http://image.diku.dk/shark/index.html`

[21] HANSEN, Nikolaus: The CMA evolution strategy: A tutorial. In: *arXiv preprint arXiv:1604.00772* (2016). `https://arxiv.org/abs/1604.00772`

DRAFT

B Abbreviations

FFD Freeform–Deformation

DM–FFD Direct Manipulation Freeform–Deformation

RBF Radial Basis Function

DRAFT

DRAFT

C List of Figures

2.1 Example of B-Splines 8
2.2 B–spline–basis–function as partition of unity 10
2.3 Figure 7 from [8]. 11
2.4 Fig. 3. taken from [12] 14

3.1 Example of a high resolution control–grid 23

4.1 The 1D–target–shape 26
4.2 Example of a 1D–grid 27
4.3 3D source and target meshes 29
4.4 Example of a 3D–grid 31

5.1 1D Fitting Errors for various grids 35
5.2 Improvement potential and regularity vs. steps 36
5.3 Correlation 1D Improvement vs. Error 37
5.4 Results 3D for 4x4xX 38
5.5 Results 3D for Xx4x4 39
5.6 Results 3D for YxYxY for Y $\in [4,5,6]$ 40

DRAFT

DRAFT

Todo list

🟪 **Improvement:** Mehr Bilder 3

🟥 **Unsure:** Nicht $(n \cdot d) \times m$? Wegen u,v,w? 15

🟥 **Unsure:** erwähnen, dass man aus **D** einfach die Null–Spalten entfernen kann? . 23

🟥 **Unsure:** verweis auf DM–FFD? 23

🟪 **Improvement:** update!! gaussian, not uniform!! 28

🟥 **Unsure:** Andreas: hast du nen cite, wo gezeigt ist, dass das so sinnvoll ist? . . . 30

🟧 machen wir die noch? :D 35

🟪 **Improvement:** Bibliotheksverzeichnis links anpassen. DOI überschreibt Direktlinks des Autors. 41

🟪 **Improvement:** write proper declaration.. xii

DRAFT

Declaration of own work(?)

I hereby declare that this thesis is my own work and effort. Where other sources of information have been used, they have been acknowledged.

Improvement: write proper declaration..

Bielefeld, den October 24, 2017 . . . .

Stefan Dresselhaus

DRAFT