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 29, 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? 11

2.2 What is evolutionary optimization? 12

2.3 Advantages of evolutionary algorithms 15

2.4 Criteria for the evolvability of linear deformations 16

2.4.1 Variability 17

2.4.2 Regularity 18

2.4.3 Improvement Potential 19

3 Implementation of Freeform–Deformation (FFD) **21**

3.1 Adaption of FFD 21

3.2 Adaption of FFD for a 3D–Mesh 23

3.3 Deformation Grid 26

4 Scenarios for testing evolvability criteria using FFD **29**

4.1 Test Scenario: 1D Function Approximation 29

DRAFT

4.2 Test Scenario: 3D Function Approximation 31

5 Evaluation of Scenarios **35**

5.1 Procedure: 1D Function Approximation 36

5.2 Results of 1D Function Approximation 38

5.2.1 Variability 38

5.2.2 Regularity 40

5.2.3 Improvement Potential 41

5.3 Procedure: 3D Function Approximation 42

5.4 Results of 3D Function Approximation 44

5.4.1 Variability 45

5.4.2 Regularity 47

5.4.3 Improvement Potential 50

6 Discussion and outlook **53**

Appendix **i**

A Bibliography **iii**

B Abbreviations **ix**

C List of Figures **xi**

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

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 aerodynamics (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.

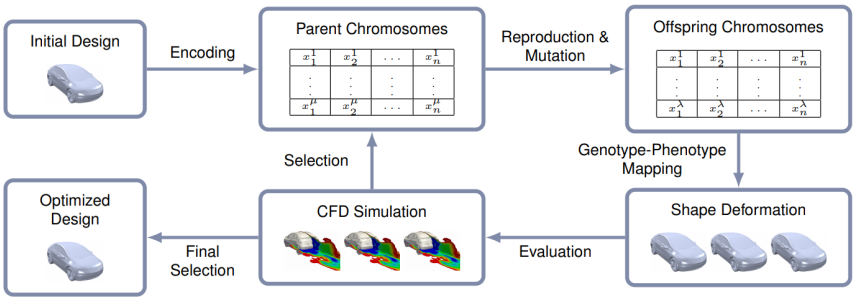

Figure 1.1: Example of the use of evolutionary algorithms in automotive design (from [1]).

Evolutionary algorithms cope especially well with these problem domains while

DRAFT

addressing all the issues at hand[2]. 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 object–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 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*[3], as the parametrization of the problem has serious implications on the convergence speed and the quality of the solution[4]. However, there is no consensus on how *evolvability* is defined and the meaning varies from context to context[5]. As a consequence 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

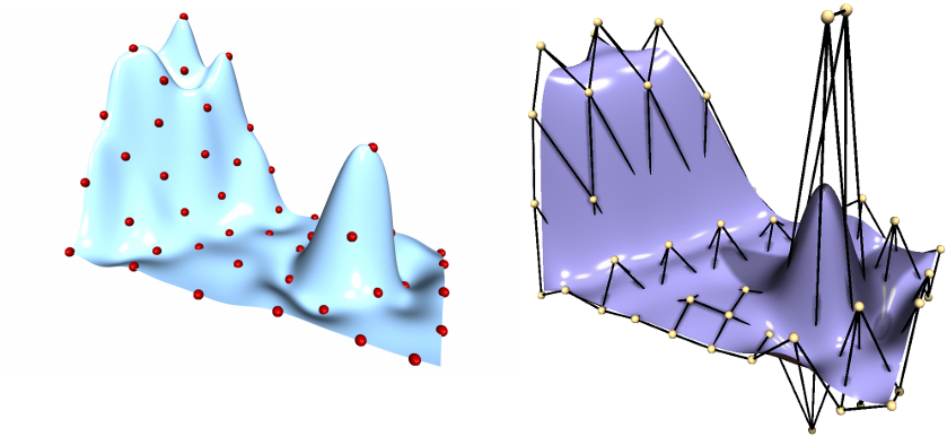

Figure 1.2: Example of RBF–based deformation and FFD targeting the same mesh.

via Radial Basis Function (RBF). If one (or the algorithm) would move such a point the object will get deformed only 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 which criteria may be suited to describe this notion of *evolvability*. This is exactly what Richter et al.[1] have done.

As we transfer the results of Richter et al.[1] 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

DRAFT

for *evolvability* of the representation given the different deformation scheme, as suspected in [1].

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) followed by the definition of the different evolvability criteria established in [1] (in 2.4).

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, summary and outlook in Chapter 6.

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 (especially 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

$$s(\mathbf{u}) = \sum_i a_i(\mathbf{u})\mathbf{p_i}$$

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 $u \in [0,1[^d$.

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

DRAFT

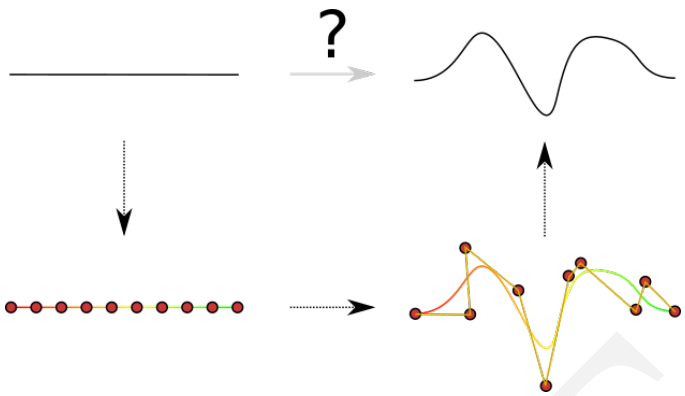

Figure 2.1: Example of a parametrization of a line with corresponding deformation to generate a deformed objet

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 $\tau_i \in [0,1[$ 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 parametrized only by $u \in [0,1[$. As can

DRAFT

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]:

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

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

[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

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.

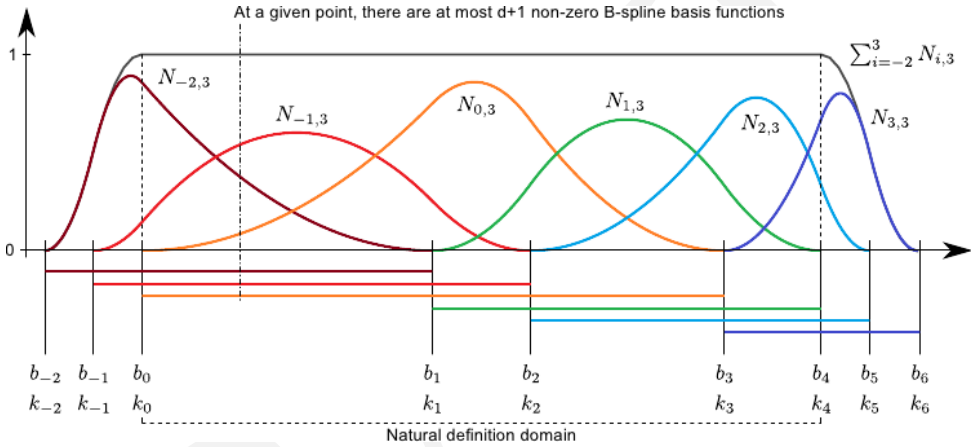

Figure 2.2: From [7, Figure 2.13]:
„Some interesting properties of the B–splines. On the natural definition domain of the B–spline ($[k_0,k_4]$ on this figure), the B–Spline basis functions sum up to one (partition of unity). In this example, we use B–Splines of degree 2. The horizontal segment below 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.

Furthermore B–Spline–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.

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 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.

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.

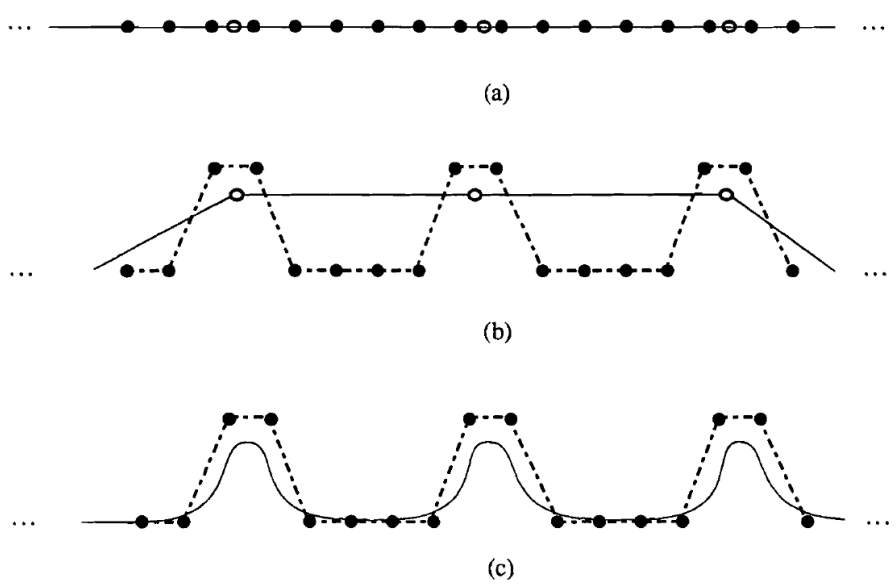

Figure 7. Aliasing because of too many control points relative to the numb-r of object points. Open circles denote object points, filled circles denote control points. No control polygon is drawn. (a) Initial condition. (b) Resulting control-point location when the three object points are moved upward.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

Figure 2.3: Figure 7 from [8].

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.

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 1 An outline of evolutionary algorithms

```
t := 0;
initialize P(0) := {a_1(0), ..., a_μ(0)} ∈ I^μ;
evaluate F(0) : {Φ(x)|x ∈ P(0)};
while c(F(t)) ≠ true do
    recombine: P(t) := r(P(t));
    mutate: P''(t) := m(P(t));
    evaluate F''(t) : {Φ(x)|x ∈ P''(t)}
    select: P(t + 1) := s(P''(t) ∪ Q,Φ);
    t := t + 1;
```

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*. *Genotypes* define all initial properties of an individual, but their properties are not directly observable. It is the genes, that evolve over time (and thus correspond to the parameters we are tweaking in our algorithms or the genes in nature), but only the *phenotypes* make certain behaviour observable (algorithmically through our *fitness–function*, biologically by the ability to survive and produce offspring). Any individual in our algorithm thus experience a biologically motivated life cycle of inheriting genes from the parents, modified by mutations occurring, performing according to a fitness–metric and generating offspring based on this. Therefore each iteration in the while–loop above is also often named generation.

One should note that there is a subtle difference between *fitness–function* and a so called *genotype–phenotype–mapping*. The first one directly applies the *genotype–*

phenotype–mapping and evaluates the performance of an individual, thus going directly from genes/parameters to reproduction–probability/score. In a concrete example the *genotype* can be an arbitrary vector (the genes), the *phenotype* is then a deformed object, and the performance can be a single measurement like an air–drag–coefficient. The *genotype–phenotype–mapping* would then just be the generation of different objects from that starting–vector, whereas the *fitness–function* would go directly from such a starting–vector to the coefficient that we want to optimize.

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.
  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. A good overview of this is given in [12], so we only give a small excerpt here.

For example the mutation can consist of merely a single σ determining the strength

of the gaussian defects in every parameter — or giving a different σ to every component of those parameters. An even more sophisticated example would be the „1/5 success rule“ from [13].

Also in the selection–function it may not be wise to only take the best–performing individuals, because it may be that the optimization has to overcome a barrier of bad fitness to achieve a better local optimum.

Recombination also does not have to be mere random choosing of parents, but can also take ancestry, distance of genes or groups of individuals into account.

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. Components and techniques for evolutionary algorithms are specifically known to help with different problems arising in the domain of optimization[14]. 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 can be modified according to the problem–domain (i.e. by the ideas given above) it can also approximate very difficult problems in an efficient manner and even self–tune parameters depending on the ancestry at runtime[3].

[3] Some examples of this are explained in detail in [12]

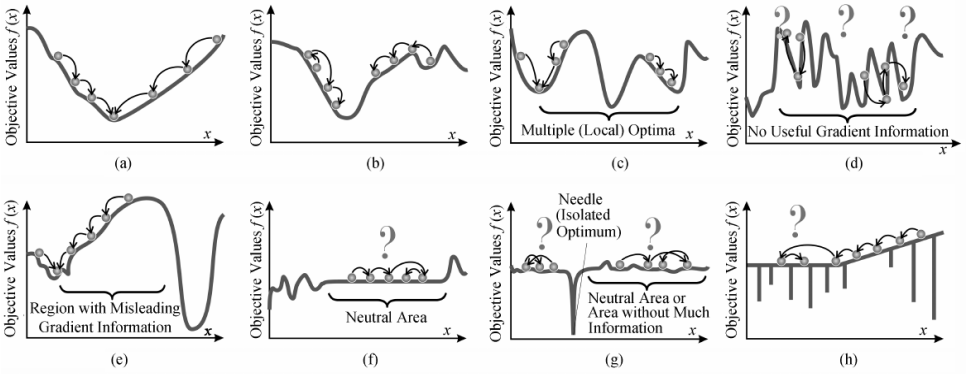

Fig.3. Examples of different possible scenarios in the fitness landscape (under minimization). (a) Best case. (b) Multi-modal with low total variation. (c) Multi-modal with higher total variation. (d) Rugged (multi-modal + high total variation). (e) Deceptive. (f) Neutral. (g) Needle-in-a-haystack. (h)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

Figure 2.4: Fig. 3. taken from [14]

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 one can at least get sub-optimal solutions fast, which then refine over time and still converge to a decent solution much faster than an exhaustive search.

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

$$\mathbf{S} = \mathbf{UP}$$

where $\mathbf{S}$ is a $n \times d$ matrix of vertices[4], $\mathbf{U}$ 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

$$\Delta\mathbf{S} = \mathbf{U} \cdot \Delta\mathbf{P}$$

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 $\mathbf{U}$, which is why the three criteria focus on this.

2.4.1 Variability

In [1] *variability* is defined as

$$\text{variability}(\mathbf{U}) := \frac{\text{rank}(\mathbf{U})}{n},$$

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

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 $m \ll n$.

[4] We use $\mathbf{S}$ in this notation, as we will use this parametrization of a source–mesh to manipulate $\mathbf{S}$ into a target–mesh $\mathbf{T}$ via $\mathbf{P}$

DRAFT

This criterion should correlate to the degrees of freedom the given parametrization has. This can be seen from the fact, that $\mathrm{rank}(\mathbf{U})$ is limited by $\min(m,n)$ and — as n is constant — can never exceed n.

The rank itself is also interesting, as control–points could theoretically be placed on top of each other or be linear dependent in another way — but will in both cases lower the rank below the number of control–points m and are thus measurable by the *variability*.

2.4.2 Regularity

Regularity is defined[1] as

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

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[15, chapter 2.7] and on the other hand for the convergence speed of evolutionary algorithms[1] as it is tied to the notion of locality[14, 16].

DRAFT

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 the potential for optimization, taking a guess into account.

Most of the times some kind of gradient g is available to suggest a direction worth pursuing; either from a previous iteration or by educated guessing. We use this to guess how much change can be achieved in the given direction.

The definition for an *improvement potential* P is[1]:

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

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

DRAFT

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

DRAFT

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

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.

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

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

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

$$\begin{array}{ll} \frac{\partial}{\partial u} & v_x - \sum_i N_{i,d,\tau_i}(u)c_i \\ = & -\sum_i \left(\frac{d}{\tau_{i+d}-\tau_i} N_{i,d-1,\tau}(u) - \frac{d}{\tau_{i+d+1}-\tau_{i+1}} N_{i+1,d-1,\tau}(u)\right) c_i \end{array}$$

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

For this we employ the Gauss–Newton algorithm[17], which converges into the least–squares solution. An exact solution of this problem is impossible most of the times, 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.

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

DRAFT

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

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:

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

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

DRAFT

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 Cramer's rule for inverting the small Jacobian and solving this system of linear equations.

As there is no strict upper bound of the number of iterations for this algorithm, we just iterate it long enough to be within the given ε–error above. This takes — depending on the shape of the object and the grid — about 3 to 5 iterations that we observed in practice.

Another issue that we observed in our implementation is, that multiple local optima may exist on self–intersecting grids. We solve this problem by defining self–intersecting grids to be *invalid* and do not test any of them.

This is not such a big problem as it sounds at first, as self–intersections mean, that control–points being further away from a given vertex have more influence over the deformation than control–points closer to this vertex. Also this contradicts the notion of locality that we want to achieve and deemed beneficial for a good behaviour of the evolutionary algorithm.

DRAFT

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 is very important for the quality and runtime of evolutionary algorithms[4].

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 local, 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 to place at all.

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 $\mathbf{p}$ only the $2 \cdot (d-1)$ control–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.

We illustrate this phenomenon in figure 3.1, where the four red central points are not relevant for the parametrization of the circle. This leads to artefacts in the deformation–matrix $\mathbf{U}$, as the columns corresponding to those control–points are 0.

[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

DRAFT

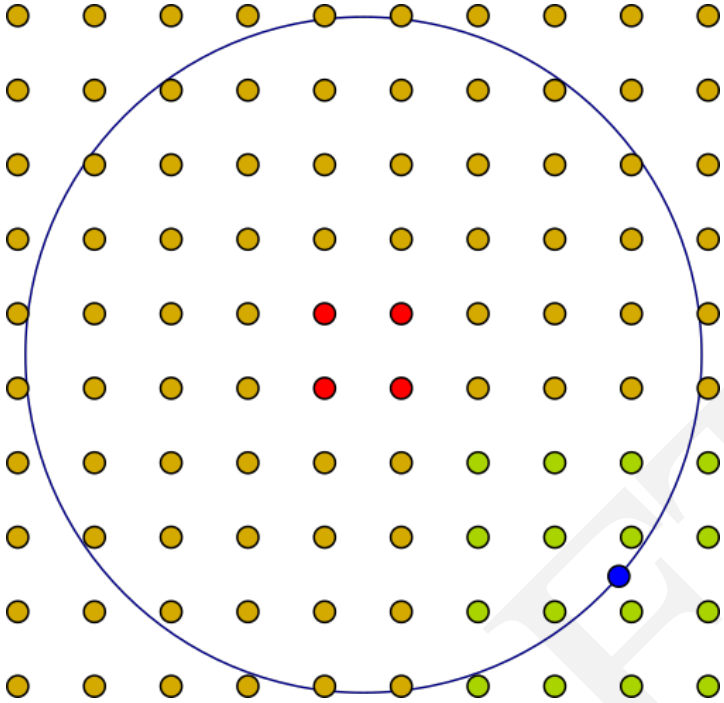

Figure 3.1: A high resolution (10×10) of control–points over a circle. Yellow/green points contribute to the parametrization, red points don't.
An Example–point (blue) is solely determined by the position of the green control–points.

This leads to useless increased complexity, as the parameters corresponding to those points will never have any effect, but a naive algorithm will still try to optimize them yielding numeric artefacts in the best and non–terminating or ill–defined solutions[2] at worst.

One can of course neglect those columns and their corresponding control–points, but this raises the question why they were introduced in the first place. We will address this in a special scenario in 5.4.1.

For our tests we chose different uniformly sized grids and added noise onto each

[2] One example would be, when parts of an algorithm depend on the inverse of the minimal right singular value leading to a division by 0.

DRAFT

control-point[3] to simulate different starting-conditions.

DRAFT

[3]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.

4 Scenarios for testing evolvability criteria using FFD

In our experiments we use the same two testing–scenarios, that were also used by [1]. The first scenario deforms a plane into a shape originally defined in [18], where we setup control-points in a 2–dimensional manner and 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.[18], which is also used by Richter et al.[1] using the same discretization to 150×150 points for a total of $n = 22\,500$ vertices. The shape is given by the following definition

DRAFT

$$t(x,y) = \begin{cases} 0.5\cos(4\pi \cdot q^{0.5}) + 0.5 & q(x,y) < \frac{1}{16}, \\ 2(y-x) & 0 < y - x < 0.5, \\ 1 & 0.5 < y - x \end{cases} \tag{4.1}$$

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.

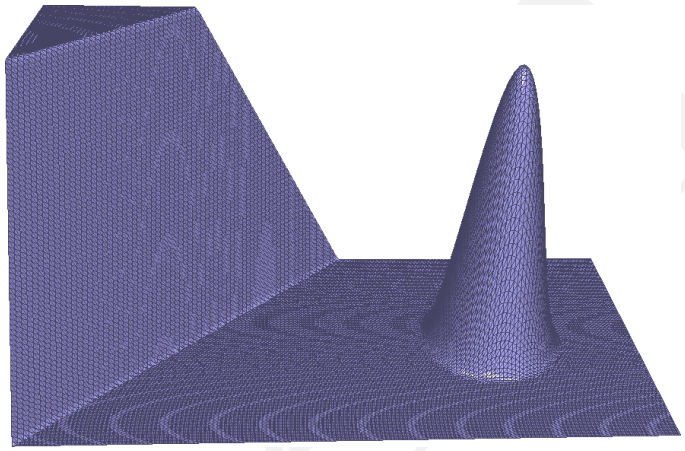

Figure 4.1: The target–shape for our 1–dimensional optimization–scenario including a wireframe–overlay of the vertices.

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

$$\mathrm{f}(\mathbf{p}) = \sum_{i=1}^{n} \|(\mathbf{Up})_i - t_i\|_2^2 = \|\mathbf{Up} - \mathbf{t}\|^2 \to \min \tag{4.2}$$

where t_i are the respective target–vertices to the parametrized source–vertices[1] with the 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 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 4.2.

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 and sum up the error from the respective source and target.

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

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

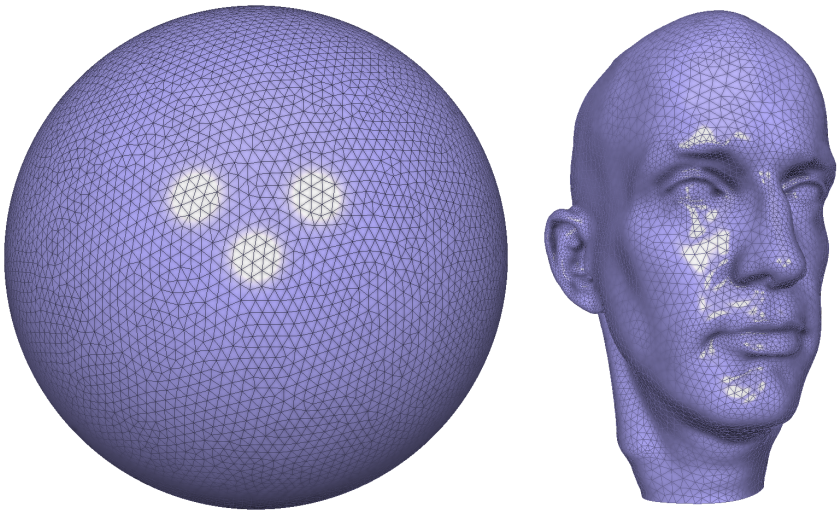

Figure 4.2:
Left: The sphere we start from with 10 807 vertices
Right: The face we want to deform the sphere into with 12 024 vertices.

$$\mathrm{f}(\mathbf{P}) = \underbrace{\frac{1}{n}\sum_{i=1}^{n} \|\mathbf{c_T}(\mathbf{s_i}) - \mathbf{s_i}\|_2^2}_{\text{source-to-target–distance}} + \underbrace{\frac{1}{m}\sum_{i=1}^{m} \|\mathbf{c_S}(\mathbf{t_i}) - \mathbf{t_i}\|_2^2}_{\text{target-to-source–distance}} + \lambda \cdot \text{regularization}(\mathbf{P}) \tag{4.3}$$

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.[19, Section 3.2] on similar models and was shown to lead to a more precise fit. The Laplacian

$$\text{regularization}(\mathbf{P}) = \frac{1}{\sum_i A_i} \sum_{i=1}^{n} A_i \cdot \left(\sum_{\mathbf{s}_j \in \mathcal{N}(\mathbf{s}_i)} w_j \cdot \|\Delta\mathbf{s}_j - \Delta\mathbf{s}_i\|^2 \right) \tag{4.4}$$

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. As a side–effect this also limits the effects of overagressive movement of the control–points in the beginning of the fitting process and thus should limit the generation of ill–defined grids mentioned in section 3.3.

DRAFT

DRAFT

5 Evaluation of Scenarios

To compare our results to the ones given by Richter et al.[1], 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 describe the relationship between two variables, without making any assumptions about the frequency distribution of the variables"[20].

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 and was also used before by Richter et al.[1]

For interpretation of these values we follow the same interpretation used in [1], based on [21]: 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 0.01 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 $< 10^{-4}$).

DRAFT

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 of the shark3.1 library [22], as this algorithm was used by [1] as well. We leave the parameters at their sensible defaults as further explained in [23, Appendix A: Table 1].

5.1 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 $\mathbf{g}_\mathrm{d}$

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

where $\mathbb{1}$ is the vector consisting of 1 in every dimension, $\mathbf{g}_\mathrm{c} = \mathbf{p}^* - \mathbf{p}$ is the calculated correct gradient, and μ is used to blend between $\mathbf{g}_\mathrm{c}$ and $\mathbb{1}$. As we always start with a gradient of $p = \mathbb{0}$ this means shortens $\mathbf{g}_\mathrm{c} = \mathbf{p}^*$.

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 (see section 3.2).

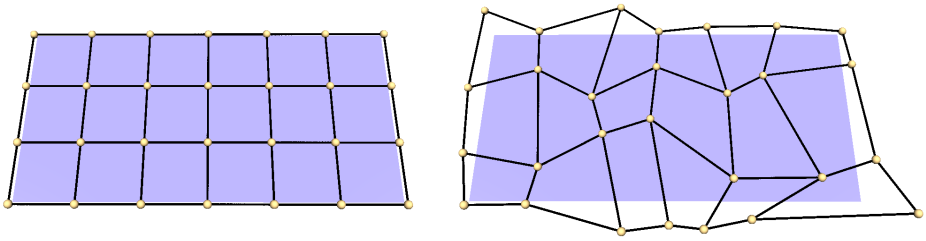

Figure 5.1:
Left: A regular 7×4–grid
Right: The same grid after a random distortion to generate a testcase.

To achieve that we generated a gaussian distributed number with $\mu = 0, \sigma = 0.25$ and clamped it to the range $[-0.25,0.25]$. We chose such an $r \in [-0.25,0.25]$ per dimension and moved the control–points by that factor towards their respective neighbours[1].

In other words we set

$$p_i = \begin{cases} p_i + (p_i - p_{i-1}) \cdot r, & \text{if } r \text{ negative} \\ p_i + (p_{i+1} - p_i) \cdot r, & \text{if } r \text{ positive} \end{cases}$$

in each dimension separately.

An Example of such a testcase can be seen for a 7×4–grid in figure 5.1.

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

DRAFT

5.2 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 $1.05\times$ of the optimal solution.

We used different regular grids that we manipulated as explained in Section 5.1 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.

5.2.1 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 box plot in figure 5.2

DRAFT

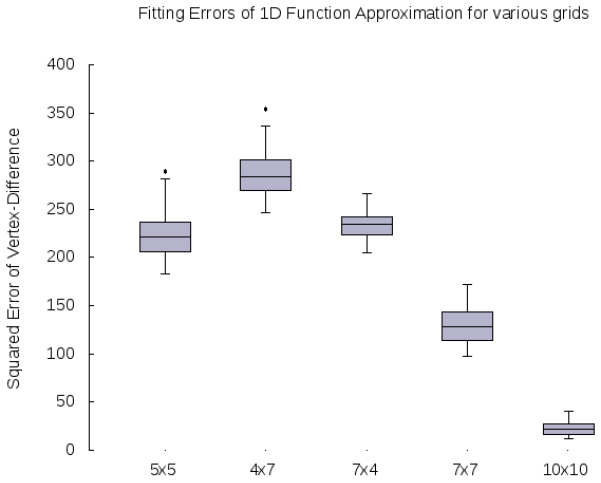

Figure 5.2: The squared error for the various grids we examined.
Note that 7×4 and 4×7 have the same number of control–points.

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 with a mean$\pm$sigma of 233.09 ± 12.32 for the former and 286.32 ± 22.36 for the latter, although they have the same number of control–points. This is an indication of an 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.

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.

DRAFT

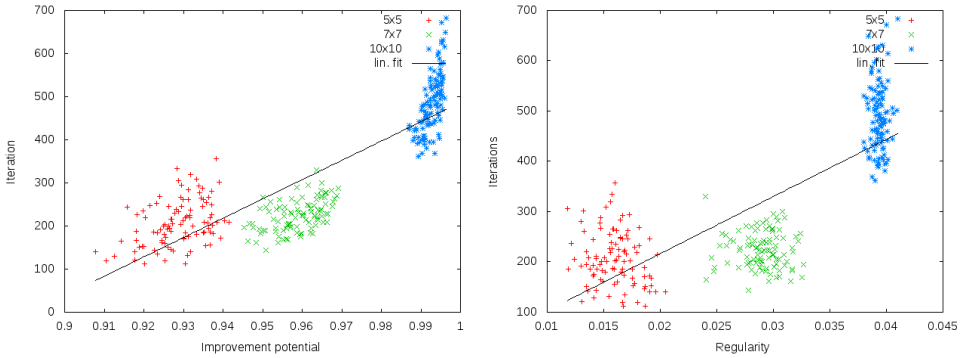

Figure 5.3:
Left: Improvement potential against steps until convergence
Right: Regularity against steps until convergence
Coloured by their grid–resolution, both with a linear fit over the whole dataset.

5.2.2 Regularity

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

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.
Note: Not significant results are marked in red.

To explain this discrepancy we took a closer look at what caused these high number of iterations. In figure 5.3 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 5.2 and 5.4) offsetting any gain the regularity–measurement could achieve.

5.2.3 Improvement Potential

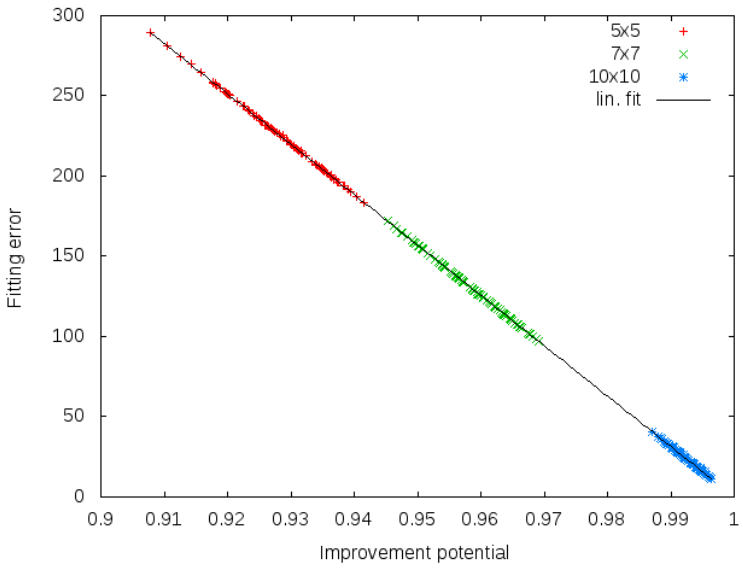

Figure 5.4: Improvement potential plotted against the error yielded by the evolutionary optimization for different grid–resolutions

The improvement potential should correlate to the quality of the fitting–result. We plotted the results for the tested grid-sizes 5×5, 7×7 and 10×10 in figure 5.4. We tested the 4×7 and 7×4 grids as well, but omitted them from the plot.

Additionally we tested the results for a distorted gradient described in 5.1 with a μ–value of 0.25, 0.5, $0{,}75$, and 1.0 for the 5×5 grid and with a μ–value of 0.5 for all other cases.

All results show the identical *very strong* and *significant* correlation with a Spearman–coefficient of $-r_S = 1.0$ and p–value of 0.

These results indicate, that $\|\mathbb{1} - \mathbf{U}\mathbf{U}^{+}\|_F$ is close to 0, reducing the impacts of any kind of gradient. Nevertheless, the improvement potential seems to be suited to make estimated guesses about the quality of a fit, even lacking an exact gradient.

5.3 Procedure: 3D Function Approximation

As explained in section 4.2 in detail, we do not know the analytical solution to the global optimum. Additionally we have the problem of finding the right correspondences between the original sphere–model and the target–model, as they consist of $10\,807$ and $12\,024$ vertices respectively, so we cannot make a one–to–one–correspondence between them as we did in the one–dimensional case.

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 $\mathbf{c_S}(\dots)$.

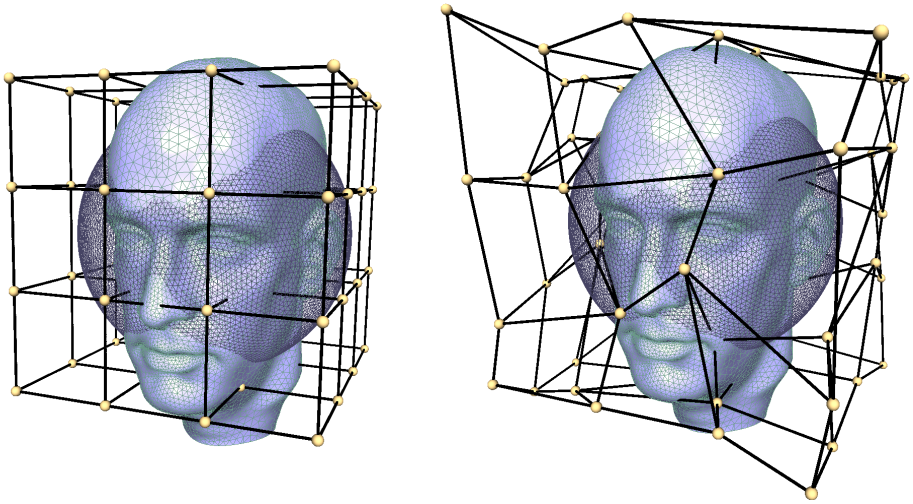

Figure 5.5:
Left: The 3D–setup with a $4 \times 4 \times 4$–grid.
Right: The same grid after added noise to the control–points.

For the next step we then halve the regularization–impact λ (starting at 1) of our *fitness–function* (4.3) and calculate the next incremental solution $\mathbf{P}^* = \mathbf{U}^+\mathbf{T}$ with the updated correspondences (again, mapping each vertex to its closest neighbour in the respective other model) to get our next target–error. We repeat this process as long as the target–error keeps decreasing and use the number of these iterations as measure of the convergence speed. As the resulting evolutional error without regularization is in the numeric range of ≈ 100, whereas the regularization is numerically ≈ 7000 we need at least 10 to 15 iterations until the regularization–effect wears off.

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.

In figure 5.5 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 in the exact same random manner between their neighbours as described in section 5.1, but in three instead of two dimensions[2].

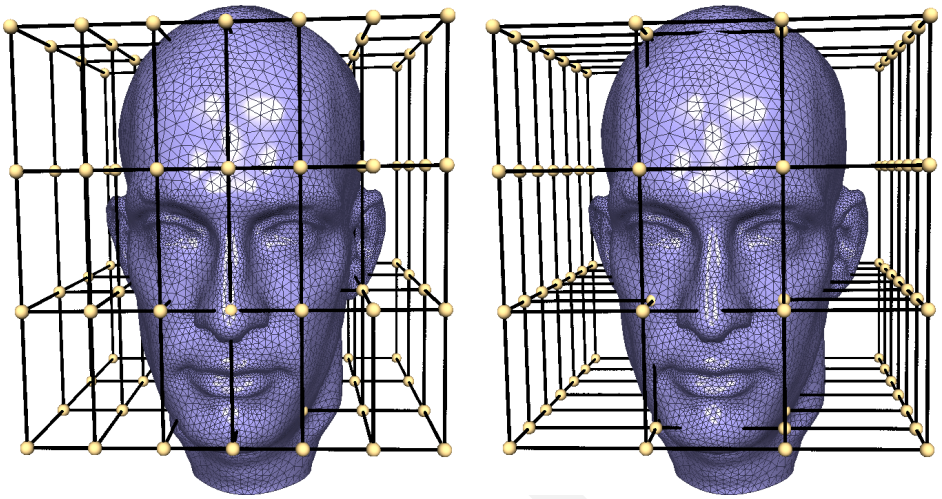

Figure 5.6:
Left: A $7 \times 4 \times 4$ grid suited to better deform into facial features.
Right: A $4 \times 4 \times 7$ grid that we expect to perform worse.

As is clearly visible from figure 5.6, 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.

5.4 Results of 3D Function Approximation

In the 3D–Approximation we tried to evaluate further on the impact of the grid–layout to the overall criteria. As the target–model has many vertices in concentrated in the facial area we start from a $4 \times 4 \times 4$ grid and only increase the number of

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

control points in one dimension, yielding a resolution of $7 \times 4 \times 4$ and $4 \times 4 \times 7$ respectively. We visualized those two grids in figure 5.6.

To evaluate the performance of the evolvability–criteria we also tested a more neutral resolution of $4 \times 4 \times 4$, $5 \times 5 \times 5$, and $6 \times 6 \times 6$ — similar to the 1D–setup.

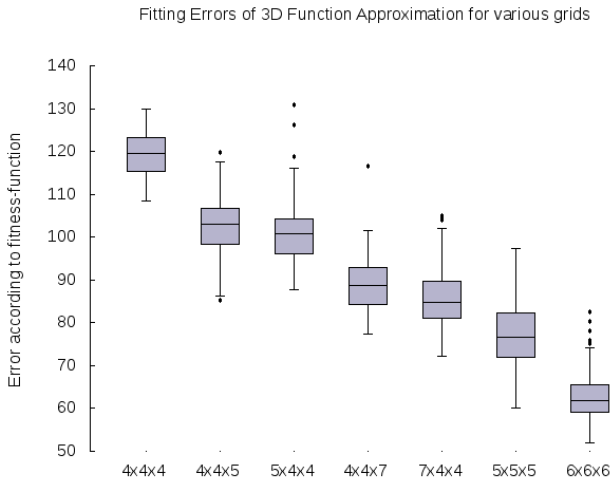

Figure 5.7: The fitting error for the various grids we examined.
Note that the number of control–points is a product of the resolution, so $X \times 4 \times 4$ and $4 \times 4 \times X$ have the same number of control–points.

5.4.1 Variability

$4 \times 4 \times$ X	X $\times 4 \times 4$	Y $\times$ Y $\times$ Y	all
0.89 (0)	0.9 (0)	0.91 (0)	0.94 (0)

Table 5.2: Correlation between variability and fitting error for the 3D fitting scenario.
Displayed are the negated Spearman coefficients with the corresponding p-values in brackets for three cases of increasing variability (X $\in$ [4,5,7], Y $\in$ [4,5,6]).
Note: Not significant results are marked in red.

Similar to the 1D case all our tested matrices had a constant rank (being $m = x \cdot y \cdot z$

for a $x \times y \times z$ grid), so we again have merely plotted the errors in the box plot in figure 5.7.

As expected the $\mathrm{X} \times 4 \times 4$ grids performed slightly better than their $4 \times 4 \times \mathrm{X}$ counterparts with a mean$\pm$sigma of 101.25 ± 7.45 to 102.89 ± 6.74 for $\mathrm{X} = 5$ and 85.37 ± 7.12 to 89.22 ± 6.49 for $\mathrm{X} = 7$.

Interestingly both variants end up closer in terms of fitting error than we anticipated, which shows that the evolutionary algorithm we employed is capable of correcting a purposefully created „bad“ grid. Also this confirms, that in our cases the number of control–points is more important for quality than their placement, which is captured by the variability via the rank of the deformation–matrix.

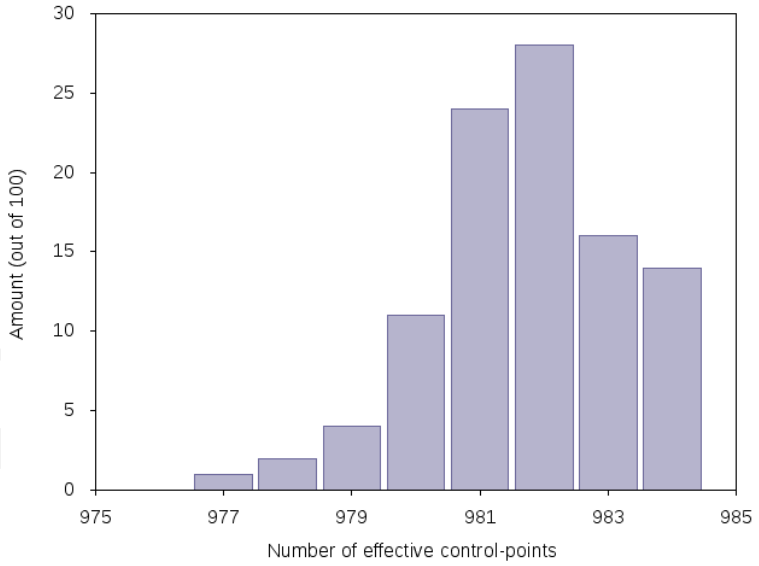

Figure 5.8: Histogram of ranks of various $10 \times 10 \times 10$ grids with 1000 control–points each showing in this case how many control points are actually used in the calculations.

Overall the correlation between variability and fitness–error were *significant* and showed a *very strong* correlation in all our tests. The detailed correlation–coefficients

are given in table 5.2 alongside their p–values.

As introduces in section 3.3 and visualized in figure 3.1, we know, that not all control points have to necessarily contribute to the parametrization of our 3D–model. Because we are starting from a sphere, some control-points are too far away from the surface to contribute to the deformation at all.

One can already see in 2D in figure 3.1, that this effect starts with a regular 9×9 grid on a perfect circle. To make sure we observe this, we evaluated the variability for 100 randomly moved $10 \times 10 \times 10$ grids on the sphere we start out with.

As the variability is defined by $\frac{\text{rank}(\mathbf{U})}{n}$ we can easily recover the rank of the deformation–matrix $\mathbf{U}$. The results are shown in the histogram in figure 5.8. Especially in the centre of the sphere and in the corners of our grid we effectively loose control–points for our parametrization.

This of course yields a worse error as when those control–points would be put to use and one should expect a loss in quality evident by a higher reconstruction–error opposed to a grid where they are used. Sadly we could not run a in–depth test on this due to computational limitations.

Nevertheless this hints at the notion, that variability is a good measure for the overall quality of a fit.

5.4.2 Regularity

Opposed to the predictions of variability our test on regularity gave a mixed result — similar to the 1D–case.

DRAFT

	$5 \times 4 \times 4$	$7 \times 4 \times 4$	$X \times 4 \times 4$
	0.15 (0.147)	0.09 (0.37)	0.46 (0)
$4 \times 4 \times 4$	$4 \times 4 \times 5$	$4 \times 4 \times 7$	$4 \times 4 \times X$
0.38 (0)	0.17 (0.09)	0.40 (0)	0.46 (0)
	$5 \times 5 \times 5$	$6 \times 6 \times 6$	$Y \times Y \times Y$
	-0.18 (0.0775)	-0.13 (0.1715)	-0.25 (0)

all: 0.15 (0)

Table 5.3: Correlation between regularity and number of iterations for the 3D fitting scenario. Displayed are the negated Spearman coefficients with the corresponding p–values in brackets for various given grids ($X \in [4,5,7]$, $Y \in [4,5,6]$). Note: Not significant results are marked in red.

In roughly half of the scenarios we have a *significant*, but *weak* to *moderate* correlation between regularity and number of iterations. On the other hand in the scenarios where we increased the number of control–points, namely 125 for the $5 \times 5 \times 5$ grid and 216 for the $6 \times 6 \times 6$ grid we found a *significant*, but *weak* **anti**–correlation when taking all three tests into account[3], which seem to contradict the findings/trends for the sets with 64, 80, and 112 control–points (first two rows of table 5.3).

Taking all results together we only find a *very weak*, but *significant* link between regularity and the number of iterations needed for the algorithm to converge.

As can be seen from figure 5.9, we can observe that increasing the number of control–points helps the convergence–speeds. The regularity–criterion first behaves as we would like to, but then switches to behave exactly opposite to our expectations, as can be seen in the first three plots. While the number of control–points increases from red to green to blue and the number of iterations decreases, the regu-

[3] Displayed as $Y \times Y \times Y$

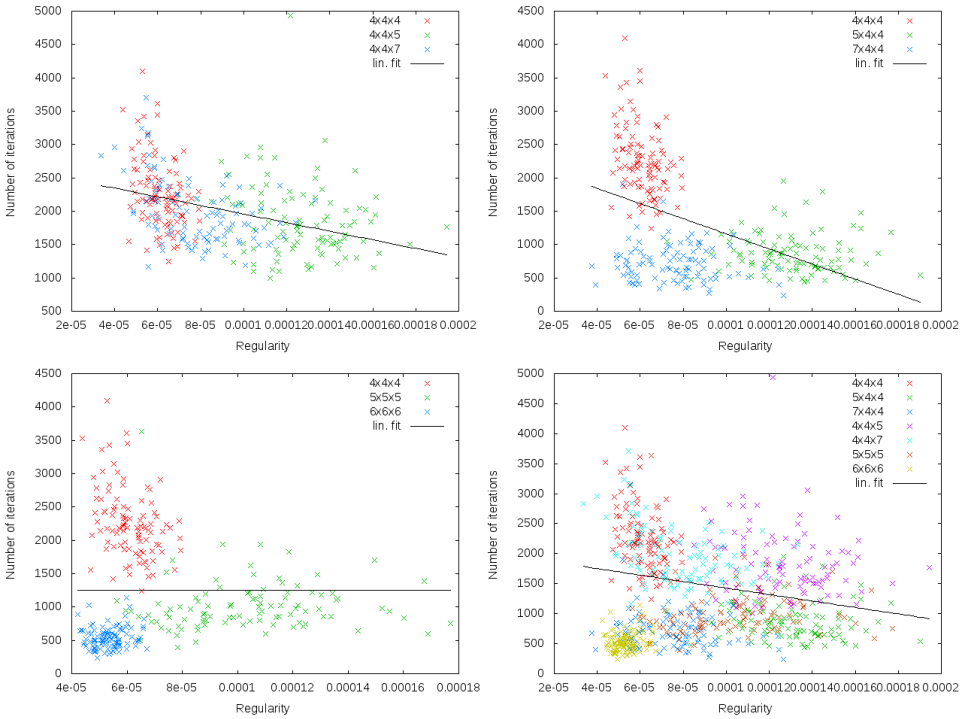

Figure 5.9: Plots of regularity against number of iterations for various scenarios together with a linear fit to indicate trends.

larity seems to increase at first, but then decreases again on higher grid–resolutions.

This can be an artefact of the definition of regularity, as it is defined by the inverse condition–number of the deformation–matrix $\mathbf{U}$, being the fraction $\frac{\sigma_{\min}}{\sigma_{\max}}$ between the least and greatest right singular value.

As we observed in the previous section, we cannot guarantee that each control–point has an effect (see figure 5.8) and so a small minimal right singular value occurring on higher grid–resolutions seems likely the problem.

Adding to this we also noted, that in the case of the $10 \times 10 \times 10$–grid the regularity was always 0, as a non–contributing control-point yields a 0–column in the

deformation–matrix, thus letting $\sigma_{\min} = 0$. A better definition for regularity (i.e. using the smallest non–zero right singular value) could solve this particular issue, but not fix the trend we noticed above.

5.4.3 Improvement Potential

	$5 \times 4 \times 4$	$7 \times 4 \times 4$	$X \times 4 \times 4$
	0.3 (0.0023)	0.23 (0.0233)	0.89 (0)
$4 \times 4 \times 4$	$4 \times 4 \times 5$	$4 \times 4 \times 7$	$4 \times 4 \times X$
0.5 (0)	0.38 (0)	0.32 (0.0012)	0.9 (0)
	$5 \times 5 \times 5$	$6 \times 6 \times 6$	$Y \times Y \times Y$
	0.47 (0)	-0.01 (0.8803)	0.89 (0)
			all: 0.95 (0)

Table 5.4: Correlation between improvement–potential and fitting–error for the 3D fitting scenario. Displayed are the negated Spearman coefficients with the corresponding p–values in brackets for various given grids ($X \in [4,5,7]$, $Y \in [4,5,6]$). Note: Not significant results are marked in red.

Comparing to the 1D–scenario, we do not know the optimal solution to the given problem and for the calculation we only use the initial gradient produced by the initial correlation between both objects. This gradient changes with every iteration and will be off our first guess very quickly. This is the reason we are not trying to create artificially bad gradients, as we have a broad range in quality of such gradients anyway.

We plotted our findings on the improvement potential in a similar way as we did before with the regularity. In figure 5.10 one can clearly see the correlation and

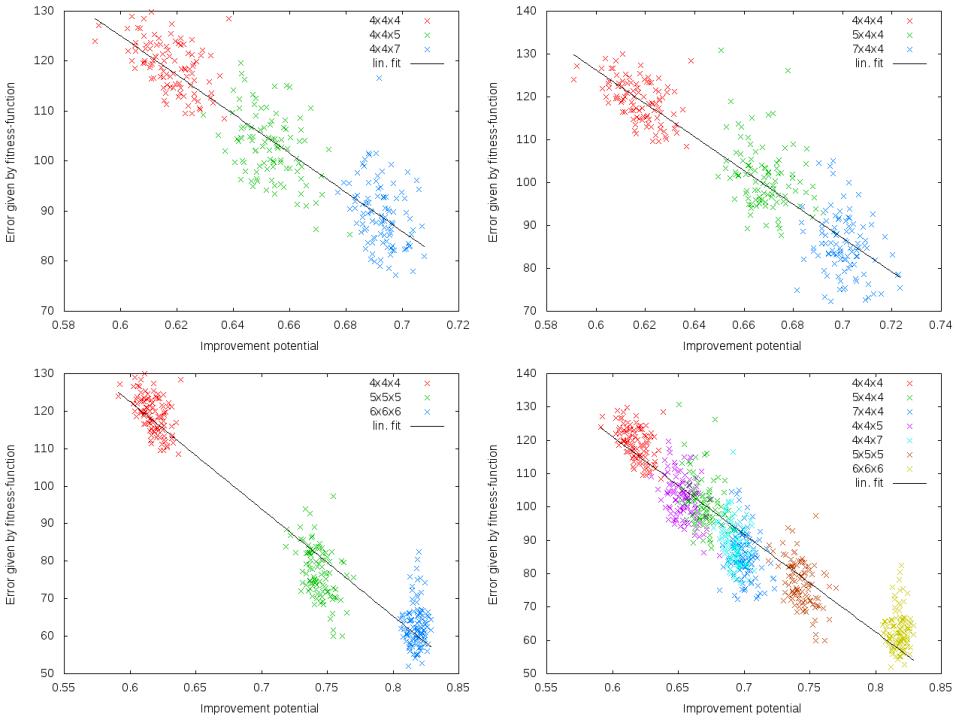

Figure 5.10: Plots of improvement potential against error given by our fitness–function after convergence together with a linear fit of each of the plotted data to indicate trends.

the spread within each setup and the behaviour when we increase the number of control–points.

Along with this we also give the Spearman–coefficients along with their p–values in table 5.4. Within one scenario we only find a *weak* to *moderate* correlation between the improvement potential and the fitting error, but all findings (except for $7 \times 4 \times 4$ and $6 \times 6 \times 6$) are significant.

If we take multiple datasets into account the correlation is *very strong* and *significant*, which is good, as this functions as a litmus–test, because the quality is naturally tied to the number of control–points.

All in all the improvement potential seems to be a good and sensible measure of quality, even given gradients of varying quality.

Lastly, a small note on the behaviour of improvement potential and convergence speed, as we used this in the 1D case to argue, why the *regularity* defied our expectations. As a contrast we wanted to show, that improvement potential cannot serve for good predictions of the convergence speed. In figure 5.11 we show improvement potential against number of iterations for both scenarios. As one can see, in the 1D scenario we have a *strong* and *significant* correlation (with $-r_S = -0.72$, $p = 0$), whereas in the 3D scenario we have the opposite *significant* and *strong* effect (with $-r_S = 0.69$, $p = 0$), so these correlations clearly seem to be dependent on the scenario and are not suited for generalization.

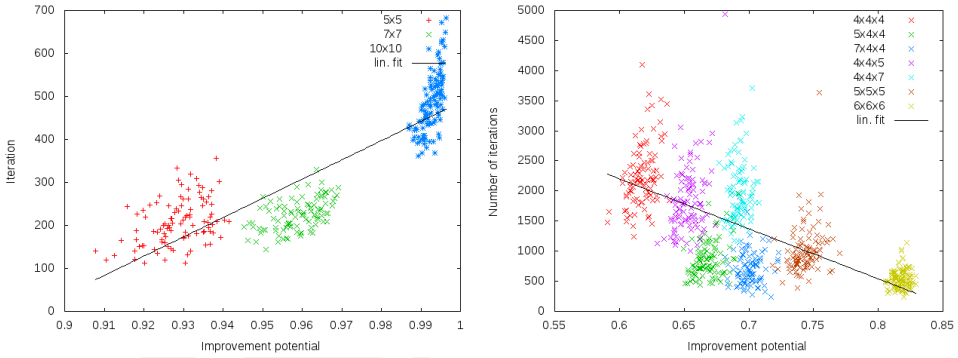

Figure 5.11:
Left: Improvement potential against convergence speed for the 1D–scenario
Right: Improvement potential against convergence speed for the 3D–scnario

6 Discussion and outlook

In this thesis we took a look at the different criteria for evolvability as introduced by Richter et al.[1], namely *variability*, *regularity* and *improvement potential* under different setup–conditions. Where Richter et al. used Radial Basis Function (RBF), we employed Freeform–Deformation (FFD) to set up a low–complexity parametrization of a more complex vertex–mesh.

In our findings we could show in the 1D–scenario, that there were statistically significant very strong correlations between *variability and fitting error* (0.94) and *improvement–potential and fitting error* (1.0) with comparable results than Richter et al. (with 0.31 to 0.88 for the former and 0.75 to 0.99 for the latter), whereas we found only weak correlations for *regularity and convergence–speed* (0.28) opposed to Richter et al. with 0.39 to 0.91.[1]

For the 3D–scenario our results show a very strong, significant correlation between *variability and fitting error* with 0.89 to 0.94, which are pretty much in line with the findings of Richter et al. (0.65 to 0.95). The correlation between *improvement*

[1] We only took statistically *significant* results into consideration when compiling these numbers. Details are given in the respective chapters.

DRAFT

potential and fitting error behave similar, with our findings having a significant coefficient of 0.3 to 0.95 depending on the grid–resolution compared to the 0.61 to 0.93 from Richter et al. In the case of the correlation of *regularity and convergence speed* we found very different (and often not significant) correlations and anti–correlations ranging from -0.25 to 0.46, whereas Richter et al. reported correlations between 0.34 to 0.87.

Taking these results into consideration, one can say, that *variability* and *improvement potential* are very good estimates for the quality of a fit using Freeform–Deformation (FFD) as a deformation function, while we could not reproduce similar compelling results as Richter et al. for *regularity and convergence speed*.

One reason for the bad or erratic behaviour of the *regularity*–criterion could be that in an FFD–setting we have a likelihood of having control–points that are only contributing to the whole parametrization in negligible amounts, resulting in very small right singular values of the deformation–matrix $\mathbf{U}$ that influence the condition–number and thus the *regularity* in a significant way. Further research is needed to refine *regularity* so that these problems get addressed, like taking all singular values into account when capturing the notion of *regularity*.

Richter et al. also compared the behaviour of direct and indirect manipulation in [1], whereas we merely used an indirect FFD–approach. As direct manipulations tend to perform better than indirect manipulations, the usage of Direct Manipulation Freeform–Deformation (DM–FFD) could also work better with the criteria we examined. This can also solve the problem of bad singular values for the *regularity* as the incorporation of the parametrization of the points on the surface, which are the essential part of a direct–manipulation, could cancel out a bad control–grid as the bad control–points are never or negligibly used to parametrize those surface–points.

DRAFT

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

DRAFT

DRAFT

Appendix

DRAFT

DRAFT

A Bibliography

[1] 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

[2] 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

[3] 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

[4] *Kapitel* 2. In: ROTHLAUF, Franz: *Representations for Genetic and Evolu-*

DRAFT

tionary Algorithms. Berlin, Heidelberg : Springer Berlin Heidelberg, 2006. – ISBN 978–3–540–32444–7, 9–32

[5] 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

[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] EIBEN, Ágoston E ; HINTERDING, Robert ; MICHALEWICZ, Zbigniew: Parameter control in evolutionary algorithms. In: *IEEE Transactions on evolutionary computation* 3 (1999), Nr. 2, 124–141. https://www.researchgate.net/profile/Marc_Schoenauer/publication/223460374_Parameter_Control_in_Evolutionary_Algorithms/links/545766440cf26d5090a9b951.pdf

[13] RECHENBERG, Ingo: Evolutionsstrategie Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. (1973)

[14] WEISE, Thomas ; CHIONG, Raymond ; TANG, Ke: Evolutionary Optimization: Pitfalls and Booby Traps. In: *J. Comput. Sci. & Technol* 27 (2012), Nr. 5. http://jcst.ict.ac.cn:8080/jcst/EN/article/downloadArticleFile.do?attachType=PDF&id=9543

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

[16] 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

DRAFT

[17] 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

[18] 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

[19] 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

[20] 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

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

[22] 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

DRAFT

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

DRAFT

DRAFT

B Abbreviations

CMA–ES Covariance Matrix Adaption Evolution Strategy

DM–FFD Direct Manipulation Freeform–Deformation

FFD Freeform–Deformation

RBF Radial Basis Function

DRAFT

DRAFT

C List of Figures

1.1 Example of the use of evolutionary algorithms in automotive design (from [1]). 3
1.2 Example of RBF–based deformation and FFD targeting the same mesh. 5

2.1 Example of B-Splines 8
2.2 B–spline–basis–function as partition of unity 10
2.3 Figure 7 from [8]. 12
2.4 Fig. 3. taken from [14] 16

3.1 Example of a high resolution control–grid 27

4.1 The 1D–target–shape 30
4.2 3D source and target meshes 32

5.1 Example of a 1D–grid 37
5.2 1D Fitting Errors for various grids 39
5.3 Improvement potential and regularity vs. steps 40
5.4 Correlation 1D Improvement vs. Error 41

DRAFT

5.5 Example of a 3D–grid 43
5.6 Different resolution of 3D grids 44
5.7 3D Fitting Errors for various grids 45
5.8 Histogram of ranks of high–resolution deformation–matrices 46
5.9 Regularity for different 3D–grids 49
5.10 Improvement potential for different 3D–grids 51
5.11 Improvement potential and convergence speed for 1D and 3D–scenarios 52

DRAFT

Todo list

Improvement: Bibliotheksverzeichnis links anpassen. DOI überschreibt Direktlinks des Autors. 54

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.

Bielefeld, October 29, 2017 .....

Stefan Dresselhaus

DRAFT