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 14, 2017

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

Contents

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

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 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 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].

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.

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 $\tau_i \in [0,1[$ 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:

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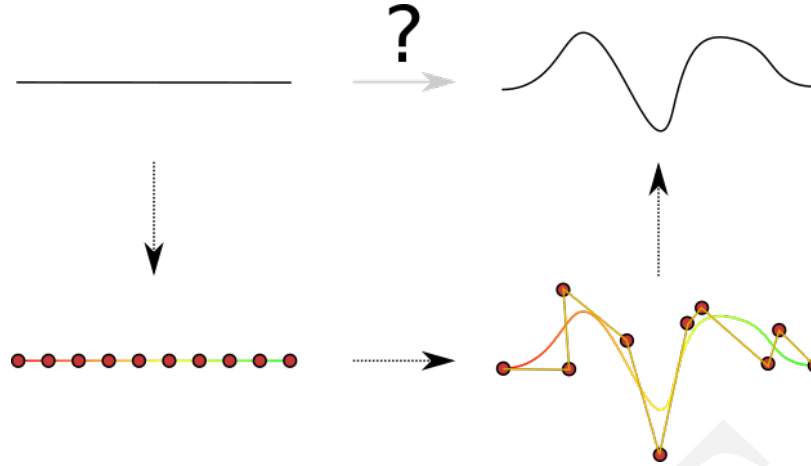
and

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

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

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]:

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

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.

---

[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

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.

> **Improvement:** Bilder von Basisfunktionen zur Visualisierung.

For a given number of points $v_1, \ldots, v_n$ 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:

$$\mathbf{v} = \mathbf{Np}$$

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

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[7, chapter 3.2][8].

This disadvantages led to the formulation of Direct Manipulation Freeform–Deformation (DM–FFD)[7, 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[9], which we will use later on.
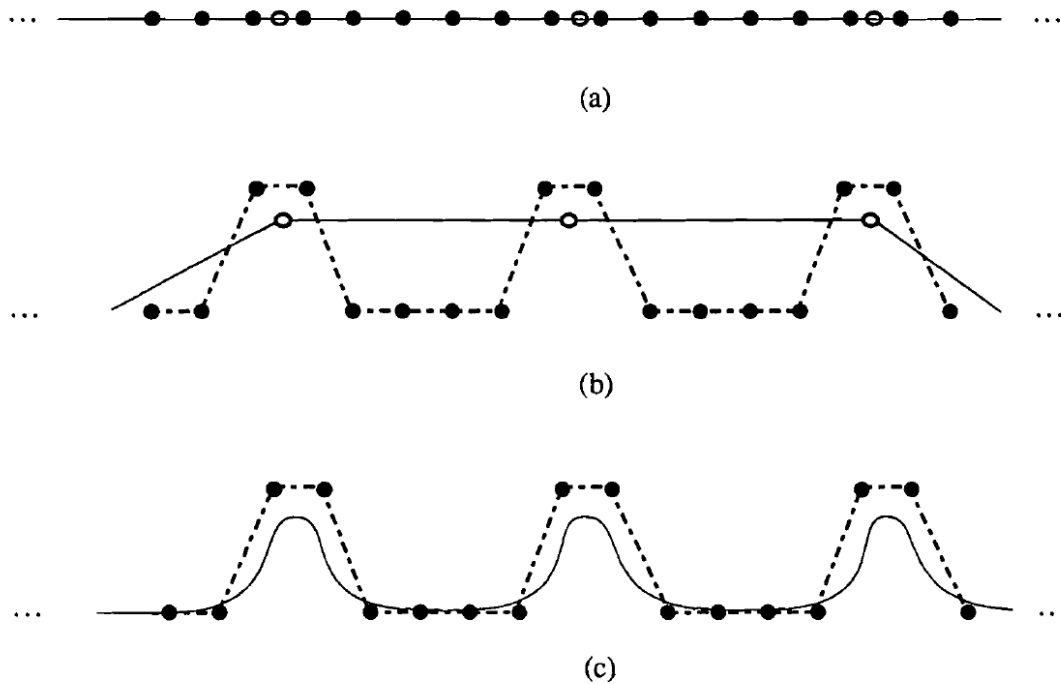

(a)

(b)

(c)

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 2.2: Figure 7 from [7].

But this approach also has downsides as can be seen in figure 2.2, 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 [10]) 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.

  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[11]. An overview of the typical problems are shown in figure 2.3.
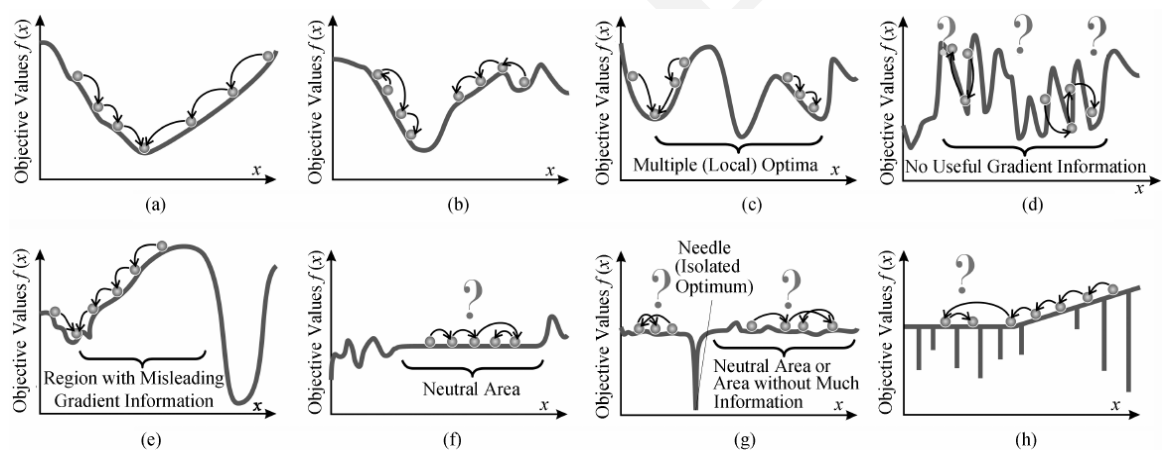

Fig.3. Examples of different possible scenarios in the fitness landscape (under minimization). (a) Best case. (b) Multi-modal with low total variation. (c) Multi-modal with higher total variation. (d) Rugged (multi-modal + high total variation). (e) Deceptive. (f) Neutral. (g) Needle-in-a-haystack. (h)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

Figure 2.3: Fig. 3. taken from [11]

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 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.

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.

2.4.1   Variability

In [5] *variability* is defined as

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

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

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

13

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 $m \ll n$.

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.

This criterion should be characteristic for numeric stability on the on hand[12, chapter 2.7] and for convergence speed of evolutionary algorithms on the other hand[5] as it is tied to the notion of locality[11, 13].

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]:

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

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

denotes the Frobenius–Norm.

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.

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:

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

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

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

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[14] 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.

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:

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

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

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

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

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–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.
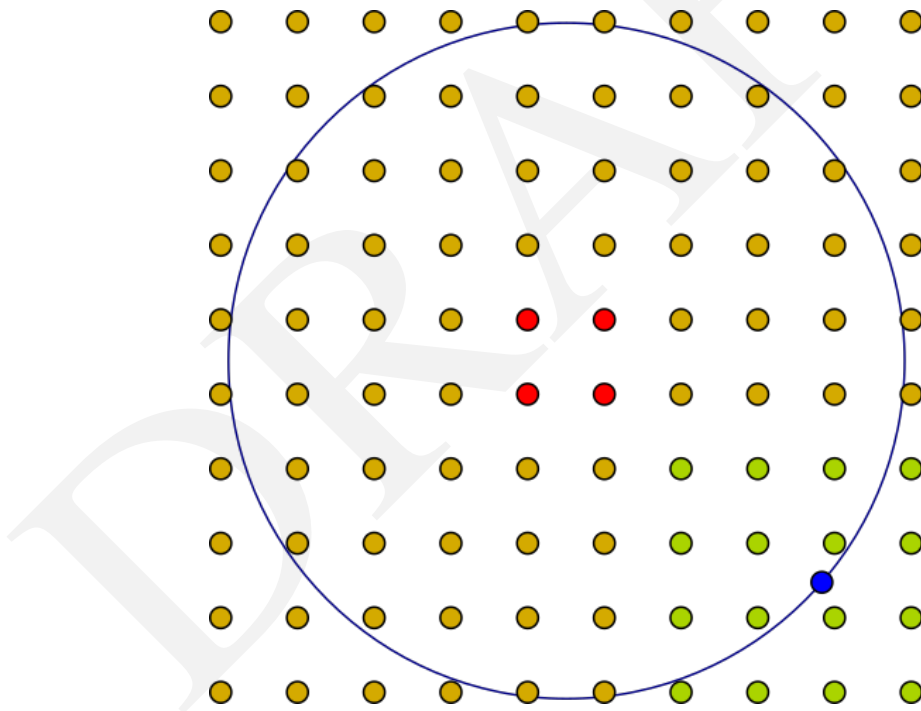

Figure 3.1: A high resolution (10×10) of control–points over a circle. Yellow/green points contribute to the parametrization, red points don't.
An Example–point (blue) is solely determined by the position of the green control–points.

---

[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

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 gaussian noise onto each control-point[2] to simulate different starting-conditions.

Unsure: verweis auf DM–FFD?

---

[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.

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

4.1 Test Scenario: 1D Function Approximation

4.1.1 Optimierungsszenario

- Ebene -> Template–Fit

4.1.2 Matching in 1D

- Trivial

4.1.3 Besonderheiten der Auswertung

- Analytische Lösung einzig beste
- Ergebnis auch bei Rauschen konstant?
- normierter 1–Vektor auf den Gradienten addieren

  – Kegel entsteht

4.2 Test Scenario: 3D Function Approximation

4.2.1 Optimierungsszenario

- Ball zu Mario

4.2.2 Matching in 3D

- alternierende Optimierung

4.2.3 Besonderheiten der Optimierung

- Analytische Lösung nur bis zur Optimierung der ersten Punkte gültig
- Kriterien trotzdem gut

5 | Evaluation of Scenarios

5.1   Spearman/Pearson–Metriken

- Was ist das?
- Wieso sollte uns das interessieren?
- Wieso reicht Monotonie?
- Haben wir das gezeigt?
- Statistik, Bilder, blah!

5.2   Results of 1D Function Approximation
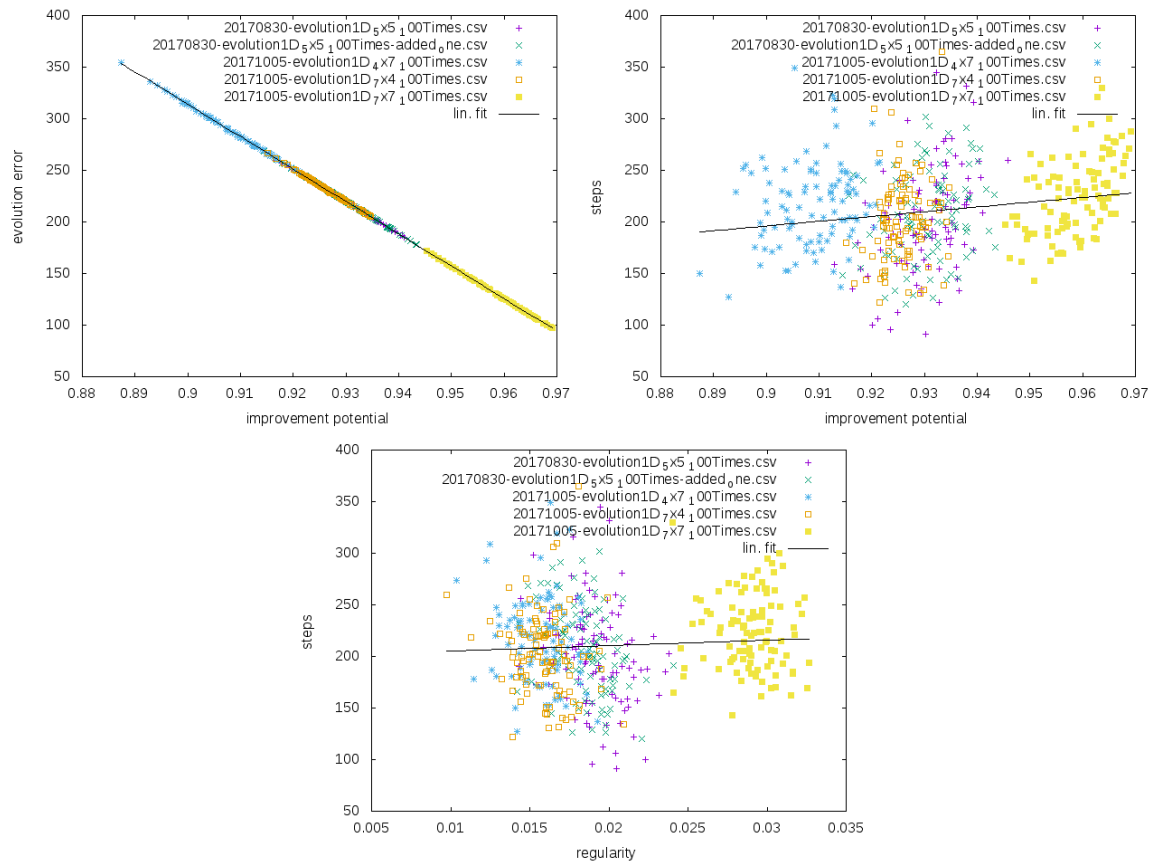
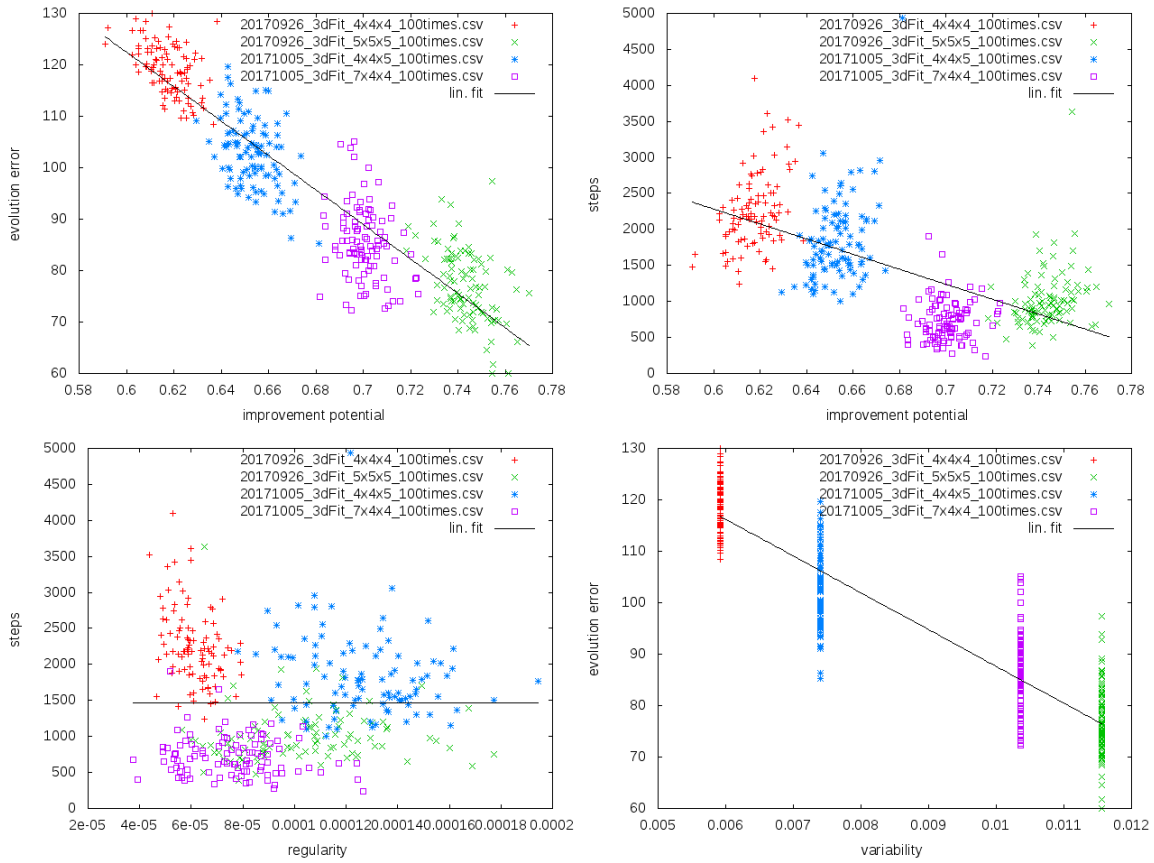## 5.3   Results of 3D Function Approximation

Figure 5.1: Results 1D

Figure 5.2: Results 3D

6 | Schluss

HAHA .. als ob -.-

Appendix

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

[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] 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`

[8] 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`

[9] 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

[10] 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`

[11] 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`

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

[13] 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

[14] 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

B | Abbreviations

FFD Freeform–Deformation

DM–FFD Direct Manipulation Freeform–Deformation

RBF Radial Basis Function

C | List of Figures

x

Todo list

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 14, 2017 . . . .

Stefan Dresselhaus