National Technical University of Athens
School of Electrical and Computer Engineering
Division of Signals, Control and Robotics

# *Implicit Neural Sculpting*

Diploma Thesis

*of*

***Petros Tzathas***

**Supervisor :** Petros Maragos
Professor NTUA
**Co-supervisor :** Anastasios Roussos
Principal Researcher FORTH

Laboratory of Computer Vision, Speech Communication and Signal Processing
Athens, October 2022

National Technical University of Athens
School of Electrical and Computer Engineering
Division of Signals, Control and Robotics
Laboratory of Computer Vision, Speech Communication and Signal Processing

# *Implicit Neural Sculpting*

Diploma Thesis

*of*

***Petros Tzathas***

**Supervisor :** Petros Maragos
Professor NTUA

**Co-supervisor :** Anastasios Roussos
Principal Researcher FORTH

Approved by the Examining Committee on 31st October, 2022:

| | | |
|---|---|---|
| ........................ | ........................ | ........................ |
| Petros Maragos | Gerasimos Potamianos | Athanasios Rontogiannis |
| Professor NTUA | Associate Professor UTH | Associate Professor NTUA |

Laboratory of Computer Vision, Speech Communication and Signal Processing
Athens, October 2022

.....................................................

**Petros Tzathas**
Electrical and Computer Engineer

# Περίληψη

Τα τελευταία χρόνια, οι πεπλεγμένες αναπαραστάσεις επιφανειών μέσω νευρωνικών δικτύων το οποία κωδικοποιούν την προσημασμένη απόσταση έχουν γίνει δημοφιλείς και έχουν επιτύχει αποτελέσματα αιχμής σε διάφορα προβλήματα (π.χ. αναπαράσταση σχήματος, ανακατασκευή σχήματος). Ωστόσο, σε αντίθεση με τις συμβατικές αναπαραστάσεις, όπως τα πολυγωνικά meshes, η επεξεργασία των πεπλεγμένων αναπαραστάσεων δεν εύκολη να και οι υπάρχουσες εργασίες που προσπαθούν να αντιμετωπίσουν αυτό το πρόβλημα είναι εξαιρετικά περιορισμένες. Στην παρούσα διατριβή, προτείνουμε την πρώτη μέθοδο για αποτελεσματική και διαδραστική επεξεργασία συναρτήσεων προσημασμένης απόστασης που εκφράζονται μέσω νευρωνικών δικτύων, επιτρέποντας ελευθερία από τον χρήστη. Εμπνευσμένοι από το λογισμικό τρισδιάστατης γλυπτικής για meshes, χρησιμοποιούμε μία θεώρηση που βασίζεται σε πινέλα η οποία είναι διαισθητική και μπορεί, στο μέλλον, να χρησιμοποιηθεί σε λογισμικό ψηφιακής τέχνης και επιστημονικές εφαρμογές. Για να περιορίσουμε την επιρροή των επιθυμητών παραμορφώσεων της επιφάνειας, ρυθμίζουμε το δίκτυο χρησιμοποιώντας ένα αντίγραφό του για να δειγματίσουμε την επιφάνεια που εκφραζόταν προηγουμένως. Εισάγουμε ένα νέο πλαίσιο για την επεξεργασιών επιφανειών σε στυλ γλυπτικής με αποτελεσματική προσαρμογή των βαρών του δικτύου, σε συνδυασμό με έναν αλγόριθμο για ομοιόμορφη δειγματοληψία επιφανειών. Αξιολογούμε ποιοτικά και ποσοτικά τη μέθοδό μας σε διάφορα διαφορετικά τρισδιάστατα αντικείμενα και κάτω από πολλές διαφορετικές επεξεργασίες. Τα αναφερόμενα αποτελέσματα δείχνουν ξεκάθαρα ότι η μέθοδός μας αποδίδει υψηλή ακρίβεια, όσον αφορά την επίτευξη των επιθυμητών επεξεργασιών, ενώ ταυτόχρονα διατηρεί τη γεωμετρία εκτός των περιοχών αλληλεπίδρασης. Ο κώδικας είναι διαθέσιμος στην συνοδευτική ιστοσελίδα https://pettza.github.io/3DNS/.

**Λέξεις Κλειδιά** — Αναπαράσταση Επιφάνειας, Πεπλεγμένες Αναπαραστάσεις, Ψηφιακή Γλυπτική, Δειγματοληψία Επιφάνειας, Δειγματοληψία με Μαρκοβιανές Αλυσίδες, Νευρωνικά Δίκτυα, Μηχανική Μάθηση

# Abstract

In recent years, implicit surface representations through neural networks that encode the signed distance have gained popularity and have achieved state-of-the-art results in various tasks (e.g. shape representation, shape reconstruction, and learning shape priors). However, in contrast to conventional shape representations such as polygon meshes, the implicit representations cannot be easily edited and existing works that attempt to address this problem are extremely limited. In the present thesis, we propose the first method for efficient interactive editing of signed distance functions expressed through neural networks, allowing free-form editing. Inspired by 3D sculpting software for meshes, we use a brush-based framework that is intuitive and can, in the future, be used in digital art software and scientific applications. In order to localize the desired surface deformations, we regulate the network by using a copy of it to sample the previously expressed surface. We introduce a novel framework for simulating sculpting-style surface edits with efficient adaptation of network weights, in conjunction with an algorithm for uniform surface sampling. We qualitatively and quantitatively evaluate our method on various different 3D objects and under many different edits. The reported results clearly show that our method yields high accuracy, in terms of achieving the desired edits, while at the same time preserving the geometry outside the interaction areas. Code is provided on the accompanying project website https://pettza.github.io/3DNS/.

**Keywords** — Surface Representation, Implicit Representations, Digital Sculpting, Surface Sampling, Markov Chain Sampling, Neural Networks, Machine Learning

# Ευχαριστίες

Καταρχάς θα ήθελα να αναφερθώ στον καθηγητή Πέτρο Μαραγκό. Ο κ. Μαραγκός ήταν ο διδάσκων των μαθημάτων που μου κέντρισαν το ενδιαφέρον και με έστρεψαν προς την ερευνητική κατεύθυνση που ακολουθώ. Ενίοτε είχαμε και μερικές ενδιαφέρουσες ανταλλαγές ιδεών (θέλω να πιστεύω ότι αυτή η ανταλλαγή ήταν όντως αμοιβαία). Πέραν αυτού υπήρξε ο επιβλέπων της διπλωματικής μου και θέλω να τον ευχαριστήσω που με έφερε σε επαφή με τον Δρ. Αναστάσιο Ρούσο, τον οποίο και θέλω να ευχαριστήσω στη συνέχεια. Ως συνεπιβλέπων της διπλωματικής μου ο Τάσος με στήριξε καθόλη τη διάρκειά της νοητικά αλλά και υλικά, καθώς μου παρείχε πόρους δίχως τους οποίος δεν θα ήταν δυνατή η εκπόνηση της εργασίας. Τον ευχαριστώ, λοιπόν, για τον αμέριστο ενθουσιασμό του, την συνεχή καθοδήγηση του και τις πολλές καρποφόρες συζητήσεις μας. Είθε η συνεργασία μας να μην τελειώσει εδώ. Τέλος, τους ευχαριστώ και τους δύο για την βοήθεια και συμμετοχή τους στην έκδοση της εργασίας.

Η κατάθεση αυτής της διπλωματικής σημάνει παράλληλα και την αποφοίτητση μου από το Πολυτεχνείο. Σε αυτό το σημείο θέλω να εκφράσω τις ευχαριστίες και την αγάπη μου στα άτομα τα οποία υπήρξαν ένα αναπόσπαστο κομμάτι της ζωής μου και με έκαναν τον άνθρωπο που είμαι. Είναι μάλλον αδύνατο να κατεφέρω να εκφράσω την σημασία τους για μένα εντός ολίγων γραμμών κειμένου αλλά επιθυμώ τουλάχιστον να τα αναγνωρίσω. Ευχαριστώ την οικογένεια μου για την αγάπη τους και τη στήριξή τους. Ευχαριστώ, επίσης, την κοπέλα μου και του φίλους μου, παιδικούς και φοιτητικούς. Ιδιαίτερα, αναφέρω τους συμφοιτητές μου στο «Παραδοσιακό», με τους οποίους μοιραστήκαμε αξέχαστες στιγμές· υπάρχει ένα ειδικό μέρος της καρδιάς μου αφιερωμένο σε αυτούς. Τέλος, θα θεωρούσα παρλάλειψη να μην αναφέρω τον κ. Διονύσιο «Σάκη» Αλεξόπουλο· εκτιμώ αυτόν και την πίστη του σε εμένα.

Πέτρος «Πετροσστός» Τζάθας

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# Chapter 1

# Ελληνική Περίληψη

## Contents

## 1.1 Εισαγωγή

Από τα φτερά μια πεταλούδας που πρόκειται να χρησιμοποιηθεί σε μια ταινία κινουμένων σχεδίων μέχρι και τα φτερά ενός αεροπλάνου για ένα μηχανολογικό σχέδιο, η ανάγκη για αναπαράσταση επιφανειών σε υπολογιστή εμφανίζεται σε μια πληθώρα εφαρμογών. Είναι λογικό, λοιπόν, πως οι τρόποι αναπαράστασης που έχουν επινοηθεί ανά τα χρόνια είναι όσο πολλαπλές και ποικίλες είναι και οι σχετικές εφαρμογές. Τα πολυγωνικά meshes, οι καμπύλες Bézier και οι επιφάνειες υποδιαίρεσης είναι μόνο μερικές από τις επιλογές, με τα meshes να αποτελούν την πλέον συνήθη αναπαράσταση.

Στο πλαίσιο της μηχανικής μάθησης η ανάγκη για αναπαράσταση επιφανειών εμφανίζεται κυρίως σε προβλήματα γεωμετρικού χαρακτήρα, όπως η ανακατασκευή της γεωμετρίας ενός αντικειμένου δοθείσας μερικής πληροφορίας περί αυτού (για παράδειγμα φωτογραφίες του ή ένα νέφος σημείων). Τα meshes έχουν αξιοποιηθεί εξαρχής και σε αυτήν την περίπτωση, με παλαιότερες εργασίες να μοντελοποιούν τις θέσεις τον κόμβων χρησιμοποιώντας ανάλυση πρωτευουσών συνιστωσών [9, 11, 24].

Παρότι τα meshes αποτελούν χρήσιμες και αποδοτικές αναπαραστάσεις, δεν είναι κατάλληλα για την μοντελοποίηση επιφανειών των οποίων οι τοπολογίες μπορεί να διαφέρουν αναμεταξύ τους. Άλλες αναπαραστάσεις έχουν προταθεί προκειμένου να ξεπεραστεί η συγκεκριμένη δυσκολία. Αυτές περιλαμβάνουν τα κανονικά πλέγματα, τα οκταδικά δένδρα και τις πεπλεγμένες αναπαραστάσεις. Λόγω της πλεγματικής δομής των πρώτων δύο, για την επεξεργασία τους έχουν χρησιμοποιηθεί συνελικτικά δίκτυα [19, 33, 70]. Πάρα την ευελιξία τους, τα κανονικά πλέγματα δεν μπορούν να επιτύχουν υψηλές αναλύσεις λόγω της κυβικής πολυπλοκότητας στη μνήμη που χρειάζονται και ενώ τα οκταδικά δένδρα αντιμετωπίζουν αυτό το ζήτημα, και αυτά καταλήγουν να έχουν την χαρακτηριστική "οδοντωτή" όψη.

Όσον αφορά τις πεπλεγμένες αναπαραστάσεις, με τον όρο αυτό εννοείται ότι η επιφάνεια αναπαρίσταται ως το ισοϋψές σύνολο στο 0 μιας τρισδιάστατης συνάρτησης. Οι πεπλεγμένες αναπαραστάσεις έχουν και αυτές μακρά ιστορία στον χώρο της επιστήμης υπολογιστών. Έχουν υπάρξει προσπάθειες χρήσης τους ανά τους καιρούς. Στη μηχανική μάθηση αυτές έχουν κατά κόρον βασιστεί σε οκταδικά δέντρα [13]. Παράλληλα, για την παραγωγή οπτικών εφέ και προγραμματιστικής τέχνης (procedural art) [54], πολύ συχνά χρησιμοποιούνται αναλυτικές εκφράσεις.

Πρόσφατα προτάθηκε από παράλληλες εργασίες [16, 47, 50] το πάντρεμα νευρωνικών δικτύων και πεπλεγμένων αναπαραστάσεων. Σε αυτές τις εργασίες χρησιμοποιείται ένα νευρωνικό δίκτυο ως η προαναφερθείσα τρισδιάστατη συνάρτηση. Στις περισσότερες από τις επακόλουθες εργασίες το νευρωνικό εκπαιδεύεται ώστε να προσεγγίζει την συνάρτηση προσημασμένης απόστασης της επιθυμητής επιφάνειας. Λόγω της ευελιξίας στην τοπολογίας και της ομαλότητας της τρισδιάστατης συνάρτησης, αυτή η προσέγγιση αντιμετωπίζει τα περιορισμούς που αναφέραμε παραπάνω και έχει οδηγήσει σε εντυπωσιακά και υποσχόμενα αποτελέσματα. Παρ᾽ όλα αυτά, υπάρχει έλλειψη από εργαλεία επεξεργασίας αυτών των καινοτόμων πεπλεγμένων νευρωνικών αναπαραστάσεων παρόμοια με αυτά που είναι διαθέσιμα για προϋπάρχουσες αναπαραστάσεις και η έρευνα πάνω σε αυτά είναι περιορισμένη.

Στην παρούσα εργασία επιχειρούμε να καλύψουμε αυτό το κενό προτείνοντας μία μέθοδο που επιτρέπει την κατά βούληση και διαδραστική τοπική επεξεργασία πεπλεγμένων νευρωνικών συναρτήσεων προσημασμένης απόστασης. Παράλληλα, αναπτύσσουμε ένα αλγόριθμο δειγματοληψίας της αναπαριστούμενης επιφάνειας. Μέσω των πειραμάτων μας αποδεικνύουμε την υπεροχή της μεθόδου μας, όσον αφορά την λεπτομέρεια της γεωμετρίας έναντι αντίστοιχων εργαλείων για τριγωνικά meshes, καθώς και την ικανότητα του αλγορίθμου μας να δειγματοληπτεί την επιφάνεια πιο ομοιόμορφα έναντι προηγουμένων.

## 1.2 Μαθηματικά Θεμέλια

Στον πυρήνα της αναπαράστασης με την οποία ασχολούμαστε βρίσκεται η *συνάρτηση προσημασμένης απόστασης*. Σε αυτήν την ενότητα θα ορίσουμε μαθηματικά την έννοια της και θα παρουσιάσουμε ορισμένες ιδιότητες που είναι σημαντικές για την ενσωμάτωση με τα νευρωνικά (για αποδείξεις βλ. κεφάλαιο 3).

Ξεκινάμε ορίζοντας τι είναι η συνάρτηση (απρόσημης) απόστασης. Έστω ένα σύνολο $A$ του $\mathbb{R}^3$. Για κάθε σημείο του τρισδιάστατου χώρου μπορούμε να ορίσουμε την απόστασή του από το σύνολο ως την ελάχιστη απόσταση από κάποιο του στοιχείο. Τυπικά έχουμε τον ακόλουθο ορισμό:

**Ορισμός 1.1: Συνάρτηση Απόστασης (ΣΑ)**

Έστω $A \subseteq \mathbb{R}^3$ και $x \in \mathbb{R}^3$. Η συνάρτηση απόστασης από το $A$ είναι:

$$d(x, A) = \inf_{y \in A} \{d(x, y)\}$$

όπου το $d$ εντός το *infimum* είναι η ευκλείδεια απόσταση μεταξύ δύο σημείων.

Προκειμένου να προχωρήσουμε στον ορισμό της προσημασμένης έκδοσης της παραπάνω συνάρτησης, χρειάζεται ακόμα να καθορίσουμε τι εννούμε με τον όρο επιφάνεια.

**Ορισμός 1.2: Επιφάνεια**

Επιφάνεια είναι ένα συμπαγές, τοπολογικά δισδιάστατο και ομαλό υποσύνολο του $\mathbb{R}^3$.

Συγκεκριμένα, μας ενδιαφέρουν οι κλειστές απλές επιφάνειες, δηλαδή αυτές που χωρίζουν το χώρο σε δύο χωρία, εκ των οποίων το ένα είναι φραγμένο και αναφέρεται ως το *εσωτερικό* της επιφάνειας (τα σημεία του βρίσκονται εντός της επιφάνειας), και το άλλο, μη φραγμένο, ως το *εξωτερικό* της επιφάνειας (τα σημεία του βρίσκονται εκτός της επιφάνειας). Η ανάλυση θα μπορούσε να ξεκινήσει από το εσωτερικό της επιφάνειας, δηλαδή θεωρώντας ότι το αντικείμενο ενδιαφέροντος είναι αυτό το σύνολο. Ο λόγος είναι ότι πολλές φορές μας ενδιαφέρει η ογκομετρία ενός αντικειμένου. Έξαλλου τα φυσικά αντικείμενα έχουν όγκο ακόμα και αν τα μοντελοποιούμε χρησιμοποιώντας την επιφάνεια τους. Σε αυτήν την περίπτωση η επιφάνεια είναι το σύνορο αυτού του συνόλου και η απαίτηση να είναι κλειστή προκύπτει αβίαστα. Λόγω του παραπάνω, το εσωτερικό της επιφάνειας αναφέρεται και ως *περικλειόμενος όγκος*.

Η συνάρτηση προσημασμένης απόστασης ορίζεται για κλειστές επιφάνειες ως εξής:

**Ορισμός 1.3: Συνάρτηση Προσημασμένης Απόστασης (ΣΠΑ)**

Έστω $S \subset \mathbb{R}^3$, μία κλειστή επιφάνεια και $x \in \mathbb{R}^3$. Η συνάρτηση προσημασμένης απόστασης από την $S$ είναι:

$$d_s(x,S) = \begin{cases} d(x,S) & \text{αν το } x \text{ είναι εκτός της } S \\ -d(x,S) & \text{διαφορετικά} \end{cases}$$

Σε αυτό το σημείο είναι εύλογο να αναρωτηθεί κανείς γιατί να προτιμάται η ΣΠΑ έναντι της ΣΑ, εφόσον η πρώτη μπορεί να οριστεί για πιο περιορισμένα σύνολα. Αφενός, όπως προαναφέραμε πολλές φορές μας ενδιαφέρουν (ή προκύπτει φυσικά) ο όγκος των αντικειμένων. Από την άλλη είναι η ΣΠΑ έχει και ένα πλεονέκτημα το οποίο θα γίνει φανερό λίαν συντόμως.

Έχοντας ορίσει τις σχετικές έννοιες μπορούμε να προχωρήσουμε στις ιδιότητες.

**Θεώρημα 1.4**

Η ΣΑ και η ΣΠΑ μια επιφάνειας είναι παραγωγίσιμες σχεδόν παντού.

Η παραπάνω ιδιότητα είναι σημαντική διότι τα νευρωνικά είναι συνεχείς και παραγωγίσιμες συναρτήσεις, είναι αναμενόμενο να προσεγγίζουν μια παραγωγίσιμη συνάρτηση καλύτερα απ' ότι μία μη-παραγωγίσιμη.

Πιο αναλυτικά, για την κλίση τους ισχύει το παρακάτω θεώρημα:

**Θεώρημα 1.5**

Έστω ένα σημείο $x \in \mathbb{R}^3$ στο οποίο η ΣΑ ή η ΣΠΑ είναι παραγωγίσιμες, τότε:

$$\begin{aligned} &\|\nabla d(x,S)\| = 1 \\ &\text{και} \\ &\|\nabla d_s(x,S)\| = 1 \end{aligned}$$

Συγκεκριμένα, αν $y \in S$ είναι το σημείο της επιφάνειας που βρίσκεται κοντύτερα στο

$x$, ισχύουν τα εξής:

$$\nabla d(x,S) = \frac{x-y}{\|x-y\|}$$

και

$$\nabla d_s(x,S) = \mathrm{sign}(d_s(x,S))\frac{x-y}{\|x-y\|}$$

Από το παραπάνω είναι φανερό ότι η κλίση μιας ΣΑ σε ένα σημείο έχει διεύθυνση την ευθεία που περνάει από αυτό σημείο και το κοντινότερο του στην επιφάνεια και κατεύθυνση μακρυά από αυτό. Για μία ΣΜΑ ισχύει το ίδιο εκτός της επιφάνειας, ενώ εντός της επιφάνειας η κλίση έχει αντίθετη κατεύθυνση από την αντίστοιχη ΣΑ. Εδώ αποκαλύπτεται το πλεονέκτημα της προσημασμένης απόστασης. Σε αντίθεση με τη ΣΑ, η ΣΠΑ είναι παραγωγίσιμη και στα σημεία της επιφάνειας και η κλίση σε αυτά είναι το κάθετο διάνυσμα στην επιφάνεια που δείχνει προς το εξωτερικό της.

Πέραν αυτού, από το παραπάνω θεώρημα προκύπτει ότι η ΣΑ και η ΣΠΑ αποτελούν λύσεις της ακόλουθης μορφής της εικονικής διαφορικής εξίσωσης:

$$\begin{aligned} &f(x) = 0,\, x \in S \\ &\|\nabla f(x)\| = 1 \end{aligned} \tag{1.1}$$

Αν απαιτήσουμε επιπλέον η λύση της εξίσωσης να είναι παραγωγίσιμη πάνω στην επιφάνεια τότε, εκ των δύο, μόνο η ΣΠΑ αποτελεί λύση.

## 1.3 Σχετική Δουλειά

Στην εισαγωγή αναφερθήκαμε σε πεπλεγμένες νευρωνικές αναπαραστάσεις. Σε αυτήν την ενότητα θα εξηγήσουμε πιο αναλυτικά τη σημασία αυτού του όρου και θα εξετάσουμε τις σημαντικότερες συμβολές σε αυτό το πεδίο.

Ως πεπλεγμένη επιφάνεια στα μαθηματικά αναφέρεται αυτή της οποίας τα σημεία δεν δίνονται άμεσα, αλλά μέσω μιας εξίσωσης την οποία ικανοποιούν. Δηλαδή ένα σημείο $x$ ανήκει στην επιφάνεια αν και μόνο αν:

$$F(x) = 0 \tag{1.2}$$

Η χρήση πεπλεγμένων αναπαραστάσεων δεν είναι κάτι πρωτοφανές στην χώρο της μηχανικής μάθησης και των γραφικών. Παλαιότερες εργασίες [13] έχουν χρησιμοποιήσει οκταδικά δένδρα για το χειρισμό τους. Παράλληλα, μέσω αναλυτικών εκφράσεων έχει καταστεί δυνατή η παραγωγή περίπλοκων οπτικών εφέ και προγραμματιστικής τέχνης, όπως μορφοκλάσματα.

Αυτό που είναι καινοτόμο είναι η συγχώνευση πεπλεγμένων αναπαραστάσεων και νευρωνικών. Πρόσφατα, υπήρξαν τρεις παράλληλες εργασίες που πρωτοστάτησαν σε αυτές τις νευρωνικές πεπλεγμένες αναπαραστάσεις. Αυτές των Park et al. [50], Chen and Zhang [16], και Mescheder et al. [47]. Εξ αυτών στην πρώτη η συνάρτηση $F$ της εξίσωσης 1.2 είναι η συνάρτηση προσημασμένης απόστασης, ενώ στις άλλες δύο είναι η δείκτρια συνάρτηση του εσωτερικού της επιφάνειας. Στην παρούσα εργασία ενδιαφερόμαστε για την πρώτη περίπτωση η οποία είναι και η πιο δημοφιλής προσέγγιση, οπότε παρακάτω θα παρουσιάσουμε συνοπτικά την εργασία των Park et al. και στη συνέχεια κάποιες από τις πιο σημαντικές συμβολές που ακολούθησαν.

Προκειμένου να αναπαραστήσουν την επιφάνεια, οι Park et al. χρησιμοποιούν ένα feedforward νευρωνικό το οποίο παίρνει ως είσοδο την χωρική συντεταγμένη και προσεγγίζει την ΣΠΑ της επιφάνειας. Το πρόβλημα μοντελοποιείται ως πρόβλημα παλινδρόμησης. Τα δεδομένα είναι ένα mesh της επιφάνειας και για την εκπαίδευση χρησιμοποιούνται ζεύγη που αποτελούνται από ένα σημείο $x$ και την πραγματική τιμή $s$ τής ΣΜΑ για αυτό. Τα σημεία $x$ παράγονται με δειγματοληψία πάνω στο mesh, κοντά στο mesh και εντός ενός παραλληλεπίπεδο που περιέχει το mesh. Η συνάρτηση κόστους είναι η εξής:

$$L(y, s) = |clamp(y, \delta) - clamp(s, \delta)| \tag{1.3}$$

όπου $y$ είναι έξοδος του νευρωνικού για την είσοδο $x$, και $\delta$ είναι μία σταθερά. Ο λόγος που χρησιμοποιείται το clamp είναι ώστε το νευρωνικό να προσεγγίζει την συνάρτηση καλύτερα κοντά στην επιφάνεια. Εκτός του να αναπαριστά ένα σχήμα, το νευρωνικό μπορεί να δέχεται ένα διανυσματικό κωδικό ως είσοδο πέραν της χωρικής συντεταγμένης και να αναπαριστά έτσι μια ολόκληρη κλάση σχημάτων.

Μετέπειτα εργασίες πρότειναν διαφορετικές συναρτήσεις και αρχιτεκτονικές προκειμένου να πετύχουν καλύτερα αποτελέσματα. Όσον αφορά τις συναρτήσεις κόστους, διάφορες εργασίες έχουν προσθέσει όρους που μπορούν να αξιοποιήσουν πληροφορία για τα κάθετα διανύσματα της επιφάνειας [3, 4]. Τα κάθετα διανύσματα που "προβλέπει" το νευρωνικό είναι η κλίση του όπως προείπαμε. Ιδιαίτερη πρόοδο έφερε η εργασία των Gropp et al. [31] στην οποία αντί να μοντελοποιήσουν το πρόβλημα ως παλινδρόμηση, θεωρούν, στο πνεύμα των Sirignano and Spiliopoulos [63], ότι το νευρωνικό λύνει μία διαφορική εξίσωση, συγκεκριμένα τη μορφή της διαφορικής εξίσωσης που δείξαμε στην εξίσωση 1.1. Η συνάρτηση κόστους αποτελείται από δύο βασικούς όρους. Ο πρώτος ωθεί το νευρωνικό να πάρει την τιμή μηδέν στα σημεία τα οποία ανήκουν στην επιφάνεια και ο δεύτερος ωθεί το νευρωνικό να έχει μοναδιαία κλίση παντού.

Οι Sitzmann et al. [64] χρησιμοποιούν την παραπάνω συνάρτηση κόστους και την επεκτείνουν προσθέτοντας ακόμα έναν όρο με τον οποίο αποφεύγεται η δημιουργία ανεπιθύμητων επιφανειών σε σημεία που δεν θα έπρεπε να υπάρχουν, το οποίο συμβαίνει διότι με την παραπάνω συνάρτηση δεν προσδιορίζεται η τιμή που πρέπει να πάρει το νευρωνικό πέραν από τα σημεία της επιφάνειας. Ο όρος αυτός λειτουργεί "τιμωρώντας" μικρές τιμές της συνάρτησης του νευρωνικού σε σημεία που δεν ανήκουν στην επιφάνεια. Στην παρούσα εργασία χρησιμοποιούμε αυτήν την επέκταση και η τελική μορφή της συνάρτησης κόστους είναι η ακόλουθη:

$$L(\theta) = L_S(\theta) + L_{eik}(\theta) + L_{es}(\theta) \tag{1.4}$$
$$L_S(\theta) = \mathbb{E}_{p_S}\{\lambda_1\,|f_\theta(x)| + \lambda_2\, g\,(\nabla_x f_\theta(x), n_x)\} \tag{1.5}$$
$$L_{eik}(\theta) = \lambda_3\,\mathbb{E}_q\big\{\big|\|\nabla_x f_\theta(x)\| - 1\big|\big\} \tag{1.6}$$
$$L_{es}(\theta) = \lambda_4\,\mathbb{E}_q\{e^{-\alpha|f_\theta(x)|}\} \tag{1.7}$$
$$g(a,b) = 1 - \frac{a\cdot b}{\|a\|\|b\|} \tag{1.8}$$

όπου τα $\lambda_1, \lambda_2, \lambda_3$ and $\lambda_4$ είναι ρυθμιστικές σταθερές, $S$ είναι η επιφάνεια, $p_S$ είναι μια κατανομή πάνω στην επιφάνεια, $q$ είναι η ομοιόμορφη κατανομή εντός ενός παραλληλεπιπέδου που περιέχει την επιφάνεια $\theta$ είναι το διάνυσμα με τα βάρη του νευρωνικού, $f_\theta$ είναι η συνάρτηση του νευρωνικού, $g$ είναι η ομοιότητα συνημιτόνου, $n_x$ είναι το κάθετο διάνυσμα στο $x$ και $\alpha$ είναι ένας μεγάλος θετικός αριθμός. Ο όρος της εξίσωσης 1.5 περιλαμβάνει το όρο που ωθεί τη συνάρτηση του νευρωνικού να μηδενίζεται στη σημεία της επιφάνειας καθώς και έναν όρα για τα κάθετα διανύσματα. Ο όρος της εξίσωσης 1.6 ωθεί την κλίση να έχει μοναδιαίο μέτρο και γι' αυτό αποκαλείται και εικονικός όρος, Τέλος, ο όρος της εξίσωση 1.7 είναι αυτός που περιγράψαμε τελευταίο.

Εξίσου ενδιαφέρουσες δουλειές έχουν ασχοληθεί με την αρχιτεκτονική του νευρωνικού δικτύου. Το SIREN [64] χρησιμοποιώντας ημίτονα αντί των συνήθων ReLUs έδειξε θετικά αποτελέσματα σε διάφορα προβλήματα. Στα πειράματά μας χρησιμοποιούμε τέτοια δίκτυα.

Οι αρχιτεκτονικές που έχουν καταφέρει να αποδώσουν μεγαλύτερη λεπτομέρεια στα σχήματα που μοντελοποιούν είναι υβριδικές [14, 49, 67]. Με αυτό εννοούμε ότι χρησιμοποιούν μια χωρική δομή δεδομένων που αποθηκεύει εκπαιδεύσιμα βάρη τα οποία χρησιμοποιούνται ως είσοδοι σε κάποιο νευρωνικό.

## 1.4 Μέθοδος

Ο στόχος μας σε αυτήν την εργασία είναι να προτείνουμε μια μέθοδο για την επεξεργασία της επιφάνειας που εκφράζεται από μια νευρωνική συνάρτηση προσημασμένης απόστασης. Η έρευνα που υπάρχει πάνω σε αυτό είναι περιορισμένη [21, 34, 48] και οι τεχνικές ουσιαστικά επιτρέπουν την διαδραστική εξερεύνηση ενός χώρου σχημάτων που μαθαίνεται κατά την εκπαίδευση χωρίς τη δυνατότητα αλλαγής πέραν αυτού. Αντ' αυτού, εμείς επιθυμούμε να καταστήσουμε δυνατή την κατά το δοκούν αλλαγή της γεωμετρίας της επιφάνειας. Εμπνεόμαστε, λοιπόν, από τα 3D sculpting λογισμικά που υπάρχουν για meshes τα οποία χρησιμοποιούν εργαλεία που λέγονται πινέλα προκειμένου να αλλάξουν τοπικά την επιφάνεια γύρω από ένα σημείο που επιλέγεται από το χρήστη.

Αναφερόμαστε σε αυτό το σημείο ως *σημείο αλληλεπίδρασης*, στην περιοχή που αλλάζει γύρω από το σημείο ως *περιοχή αλληλεπίδρασης* και στην διαδικασία γενικά ως *αλληλεπίδραση*. Το κυριότερο πρόβλημα που συναντάμε είναι το πώς να επιβάλλουμε την τοπικότητα. Για να αλλάξουμε την επιφάνεια που αναπαρίσταται πρέπει να αλλάξουμε τις παραμέτρους του νευρωνικού, αλλά, εν γένει, μια αλλαγή στις παραμέτρους επηρεάζει την έξοδο της συνάρτησης του νευρωνικού σε ένα μη φραγμένο χωρίο του χώρου εισόδου. Επομένως, αν εκπαιδεύσουμε αφελώς το δίκτυο χρησιμοποιώντας δείγματα μόνο από την περιοχή αλληλεπίδρασης η επιφάνεια θα παραμορφωθεί και στο υπόλοιπό της. Προκειμένου να ελαττώσουμε αυτό το παράπλευρο αποτέλεσμα προτείνουμε συνδυασμό δειγμάτων που βρίσκονται εκτός της περιοχής αλληλεπίδρασης και εκφράζουν την υπάρχουσα γεωμετρία και δειγμάτων από την επιθυμητή αλλαγή σύμφωνα με το πινέλο. Τα πρώτα τα αποκαλούμε *δείγματα επιφάνειας* και τα δεύτερα *δείγματα αλληλεπίδρασης*.

### 1.4.1 Δείγματα Επιφάνειας

Δύο υπάρχουσες δουλειές [5, 18] έχουν προτείνει παρόμοιους αλγορίθμους για τη δειγματοληψία του ισοϋψούς συνόλου μια νευρωνικής συνάρτησης. Ξεκινάνε δειγματοληπτώντας σημεία ομοιόμορφα κατανεμημένα εντός ένος παραλληλεπιπέδου που περιέχει το σύνολο και έπειτα τα προβάλουν στην επιφάνεια με επαναλήψεις γενικευμένου Newton-Raphson:

$$x_{n+1} = x_n - f(x_n)\frac{\nabla f(x_n)}{\|\nabla f(x_n)\|} \tag{1.9}$$

Για μία πραγματική ΣΠΑ το $f(x_n)$ είναι η απόσταση του $x_n$ από το κοντινότερο σημείο στην επιφάνεια και η κλίση της $f$ είναι η κατεύθυνση αντίθετη από αυτό, οπότε μία επανάληψη είναι αρκετή. Οι νευρωνικές ΣΠΑ όμως είναι προσεγγίσεις, συνεπώς χρειάζονται παραπάνω, αλλά όχι πολλές. Στην πράξη χρησιμοποιούμε 7.

Ακολουθούμε συγκεκριμένα την εργασία των Chibane et al. [18] στην οποία ο αρχικός αριθμός δειγμάτων είναι μικρότερος από τον τελικό. Έχοντας βρει κάποια δείγματα πάνω στην επιφάνεια με τον παραπάνω τρόπο, επεκτείνουν το σύνολο αυτό δημιουργώντας καινούργια από τα υπάρχοντα. Συγκεκριμένα διαλέγουν τυχαία σημεία από αυτό το σύνολο, τα μετατοπίζουν με γκαουσιανό θόρυβο και τα ξαναπροβάλουν στην επιφάνεια.

Στην εργασία μας χρησιμοποιήσουμε παρόμοια διαδικασία με την παραπάνω προκειμένου να παράξουμε τα δείγματα της επόμενης επανάληψης της εκπαίδευσης. Έστω ότι έχουμε ένα σύνολο δειγμάτων επιφάνειας. Τότε τα μετατοπίζουμε εφαπτομενικά στην επιφάνεια και τα ξαναπροβάλουμε. Πιο αναλυτικά, μετατοπίζουμε κάθε σημείο ομοιόμορφα στον δίσκο (η ακτίνα του είναι σταθερή για όλα τα σημεία) που εφάπτεται της επιφάνειας σε αυτό το σημείο. Διεξάγουμε πειράματα που αποδεικνύουν ότι αυτός ο τρόπος δειγματοληψίας οδηγεί σε πιο ομοιόμορφες κατανομές από τον αφελή τρόπο του να κατασκευάζουμε το σύνολο δειγμάτων από την αρχή σε κάθε επανάληψη εκπαίδευσης.

### 1.4.2 Πινέλα

Μία από τις δημοφιλέστερες μεθόδους για κατασκευή και επεξεργασία 3D μοντέλων είναι η 3D γλυπτική. Η ονομασία προέρχεται από την ομοιότητα με την γλυπτική πηλού, όπου ο γλύπτης χρησιμοποιεί διάφορα εργαλεία προκειμένου να προσθέσει/αφαιρέσει υλικό και να δώσει σχήμα στην επιφάνεια. Τα εργαλεία στην ψηφιακή γλυπτική είναι τα πινέλα. Παρακάτω παρουσιάζουμε τον δικό μας φορμαλισμό για τα πινέλα και τον τρόπο που επηρεάζουν την επιφάνεια στην περιοχή αλληλεπίδρασης.

#### Πρότυπα και Οικογένειες Πινέλων

Ορίζουμε ως πρότυπο πινέλου μία $C^1$ (ή ομαλότερη) δισδιάστατη συνάρτηση ορισμένη πάνω στον μοναδιαίο δίσκο με κέντρο την αρχή των αξόνων, η οποία έχει μέγιστη τιμή το 1 και μηδενίζεται στο μοναδιαίο κύκλο (ιδανικά και οι παράγωγοί της το ίδιο). Έστω ότι $b_T(x)$ είναι ένα πρότυπο πινέλου, τότε οι παραπάνω ιδιότητες συνοψίζονται ως εξής:

$$b_T : \{x \in \mathbb{R}^2 \mid \|x\| \leq 1\} \to \mathbb{R} \tag{1.10}$$

$$\max\{b_T(x)\} = 1 \tag{1.11}$$

$$\|x\| = 1 \Rightarrow b_T(x) = 0 \ (\wedge \nabla_x b_T(x) = 0) \tag{1.12}$$

Μπορούμε, τώρα, να ορίσουμε μία οικογένεια $B_{r,s}$ παραμετροποιημένη με την ακτίνα $r$ και την ένταση $s$ ως:

$$B_{r,s}(x) = s\, b_T\left(\frac{x}{r}\right),\ r \in \mathbb{R}^+, s \in \mathbb{R} \tag{1.13}$$

Η ακτίνα επηρεάζει το μέγεθος της περιοχής αλληλεπίδρασης, ενώ η ένταση ελέγχει το πόσο θα αλλοιωθεί η επιφάνεια. Όπως θα γίνει φανερό, μία θετική τιμή για την ένταση δημιουργεί εξογκώματα, ενώ μία αρνητική τιμή δημιουργεί βαθουλώματα.

Στα πειράματά μας χρησιμοποιούμε το ακόλουθο πρότυπο πινέλου, το οποίο είναι συμμετρικό ως προς την αρχή των αξόνων:

$$b_T(x) = P(1 - \|x\|) \tag{1.14}$$

$$P(x) = 6x^5 - 15x^4 + 10x^3 \tag{1.15}$$

### 1.4.3 Δράση Πινέλων

Για να χρησιμοποιήσουμε ένα πινέλο πάνω σε ένα σημείο της επιφάνειας θεωρούμε ότι η συνάρτησή του ορίζεται στο εφαπτόμενο επίπεδο σε αυτό το σημείο με τον άξονα $z$ να ταυτίζεται με το κάθετο διάνυσμα. Χρησιμοποιώντας το θεώρημα πεπλεγμένης συνάρτησης [44] μπορούμε να εκφράσουμε την επιφάνεια, η οποία είναι το ισοϋψές σύνολο τιμής 0 της συνάρτησης του νευρωνικού, ως το γράφο μίας δισδιάστατης συνάρτησης ορισμένης πάνω στο ίδιο επίπεδο. Η αλλοιωμένη επιφάνεια δίνεται από το γράφο του αθροίσματος της πεπλεγμένης συνάρτησης και της συνάρτησης πινέλου. Επομένως η δράση του πινέλου σε ένα σημείο της επιφάνειας είναι η μετατόπισή του κατά την κατεύθυνση το κάθετου διανύσματος του σημείου αλληλεπίδρασης απόσταση ίση με την τιμή του πινέλου στο αρχικό.

### 1.4.4 Δείγματα Αλληλεπίδρασης

Έχοντας περιγράψει το πώς δρα ένα πινέλο πάνω στην επιφάνεια, προχωράμε στο πώς παράγουμε τα δείγματα αλληλεπίδρασης. Αρχικά τοποθετούμε σημεία ομοιόμορφα στον εφαπτόμενο δίσκο στο σημείο αλληλεπίδρασης με ακτίνα ίση με την ακτίνα του πινέλου. Έπειτα προβάλουμε τα σημεία αυτά πάνω στην επιφάνεια και τα μετατοπίζουμε όπως περιγράψαμε προηγουμένως. Μπορούμε να χρησιμοποιήσουμε το θεώρημα πεπλεγμένης συνάρτησης για να υπολογίσουμε και τα αλλοιωμένα κάθετα διανύσματα επίσης.

### 1.4.5 Συνδυασμός Δειγμάτων Επιφάνειας και Αλληλεπίδρασης

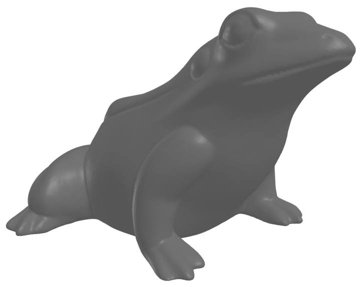
Όπως αναφέραμε στην αρχή της ενότητας, η μέθοδος που προτείνουμε λειτουργεί συνδυάζοντας δείγματα από την επιφάνεια που εκφράζεται από το νευρωνικό πριν την αλληλεπίδραση και δείγματα από την επιθυμητή αλλαγή. Για το σκοπό αυτό, αφού δημιουργήσουμε τα δείγματα επιφάνειας, αφαιρούμε από το σύνολο αυτά που βρίσκονται εντός ακτίνας ίσης με αυτήν του πινέλου από το σημείο αλληλεπίδρασης. Έστω ότι $N$ είναι ο αρχικός αριθμός δειγμάτων και $M$ ο αριθμός όσων αφαιρέθηκαν.

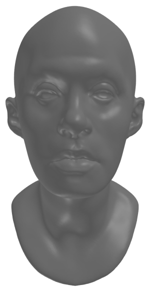
Το επόμενο που χρειάζεται να προσδιορίσουμε είναι ο αριθμός των δειγμάτων αλληλεπίδρασης. Η εξισορρόπηση μεταξύ των δύο δειγμάτων δεν γίνεται ρητά με βάρη, οπότε η σχέση μεταξύ αυτού του αριθμού και του αριθμού δειγμάτων επιφάνειας που απέμειναν $(N - M)$ παίζει αυτό το ρόλο. Πολύ λίγα δείγματα αλληλεπίδρασης δεν θα είναι αρκετά για να αλλάξει όπως θέλουμε η επιφάνεια. Πολλά, από την άλλη, μπορεί να οδηγήσουν σε έντονη παραμόρφωση εκτός της περιοχής αλληλεπίδρασης λόγω ελλιπούς εκπροσώπησης των δειγμάτων επιφάνειας. Εκτός αυτού, θέλουμε ο αριθμός των δειγμάτων αλληλεπίδρασης να είναι ανάλογος του εμβαδού της περιοχής αλληλεπίδρασης. Παρότι δεν μπορούμε να υπολογίσουμε αυτό το εμβαδόν αναλυτικά, λόγω της σχετικής ομοιομορφίας της κατανομής που παράγει ο αλγόριθμος δειγματοληψίας μας, ο λόγος $M/N$ το προσεγγίζει πιθανοτικά. Συνεπώς, μπορούμε να χρησιμοποιήσουμε το $M$ για να κλιμακώσουμε τον αριθμό τον δειγμάτων αλληλεπίδρασης. Επιπλέον, πολλαπλασιάζουμε και με έναν παράγοντα (επιλέγουμε το 10) ώσταν να αυξήσουμε την σημασία τους στην συνάρτηση κόστους εφόσον αυτά ευθύνονται για την αλλαγή.

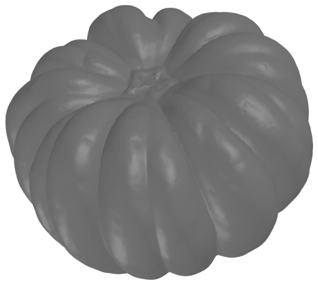
## 1.5 Πειράματα

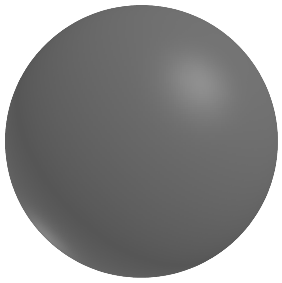
Για την ποιοτική και ποσοτική αξιολόγηση της προτεινόμενης μεθόδου καθώς και του αλγορίθμου δειγματοληψίας διεξάγουμε μία σειρά πειραμάτων, συμπεριλαμβανομένης μιας

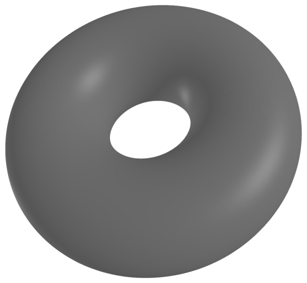
(a) Bunny


(b) Frog


(c) Bust


(d) Pumpkin

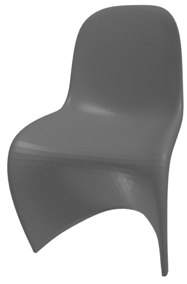

(e) Sphere


(f) Torus

Σχήμα 1.1: Το σύνολο σχημάτων που συλλέξαμε για τα πειράματά μας.

(a) Dining chair

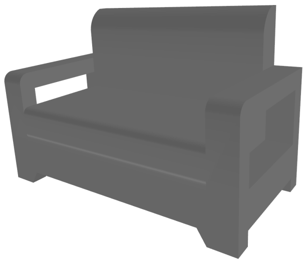
*4dd46b9657c0e998b4d5420f7c27d2df*


(b) Chair

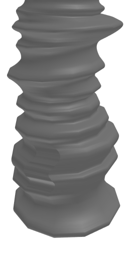
*02e76cb4f1039c482eb499cc8fbcd*

(c) Armchair

*c5d880efc887f6f4f9111ef49c078dbe*


(d) Sofa

*bcff6c5cb4127aa15e0ae65e074d3ee1*


(e) Vase

*13375f8fce3142e6597d391ab6fcc1*

Σχήμα 1.2: Τα επιπλέον σχήματα από το ShapeNet τα οποία χρησιμοποιούμε για τη σύγκριση με την επεξεργασία mesh. Κάτω από τη λεζάντα κάθε εικόνας φαίνεται το ShapeNet ID του αντίστοιχου μοντέλου σε πλάγια γράμματα.

σύγκρισης με επεξεργασία mesh. Για την υλοποίηση μας χρησιμοποιούμε το PyTorch [51].

Για αυτά τα πειράματα συλλέξαμε ένα σύνολο σχημάτων. Συγκεκριμένα χρησιμοποιούμε τέσσερα meshes, το Standord Bunny [72] και τρία από το TurboSquid [71], καθώς και δύο σχήματα που περιγράφονται αναλυτικά, μία σφαίρα και ένα τόρο. Τα παραπάνω φαίνονται στο σχήμα 1.1. Για τη σύγκριση με την επεξεργασία mesh επεκτείνουμε αυτή τη συλλογή με μερικά μοντέλα από το ShapeNet [15] τα οποία φαίνονται στο σχήμα 1.2.

Η αρχιτεκτονική που χρησιμοποιούμε είναι SIREN με 2 κρυφά επίπεδα με 128 νευρώνες το καθένα. Επιπλέον εφαρμόσουμε κανονικοποίηση βαρών [60] σε κάθε επίπεδο. Για κάθε σχήμα εκπαιδεύουμε ένα ξεχωριστό νευρωνικό. Τέλος, χρησιμοποιούμε την απόσταση Chamfer για τις αριθμητικές συγκρίσεις.

### 1.5.1 Αξιολόγηση Αλγόριθμου Δειγματοληψίας

Θέλουμε να μελετήσουμε την κατανομή που παράγει ο αλγόριθμος δειγματοληψίας μας. Για αυτό το σκοπό διαμερίζουμε την επιφάνεια σε περιοχές και προσεγγίζουμε την μέση τιμή της συνάρτησης πυκνότητας πιθανότητας σε καθεμία από αυτές. Συγκεκριμένα, εκτελούμε $N$ $(= 100)$ επαναλήψεις του αλγορίθμου, με $N$ $(= 10000)$ δείγματα η καθεμία, και για κάθε περιοχή $D$ μετράμε τον συνολικό αριθμό των δειγμάτων $c_D$ εντός της. Η εκτιμήτρια της μέσης τιμής της συνάρτησης πυκνότητας πιθανότητας σε μία περιοχή είναι, λοιπόν:

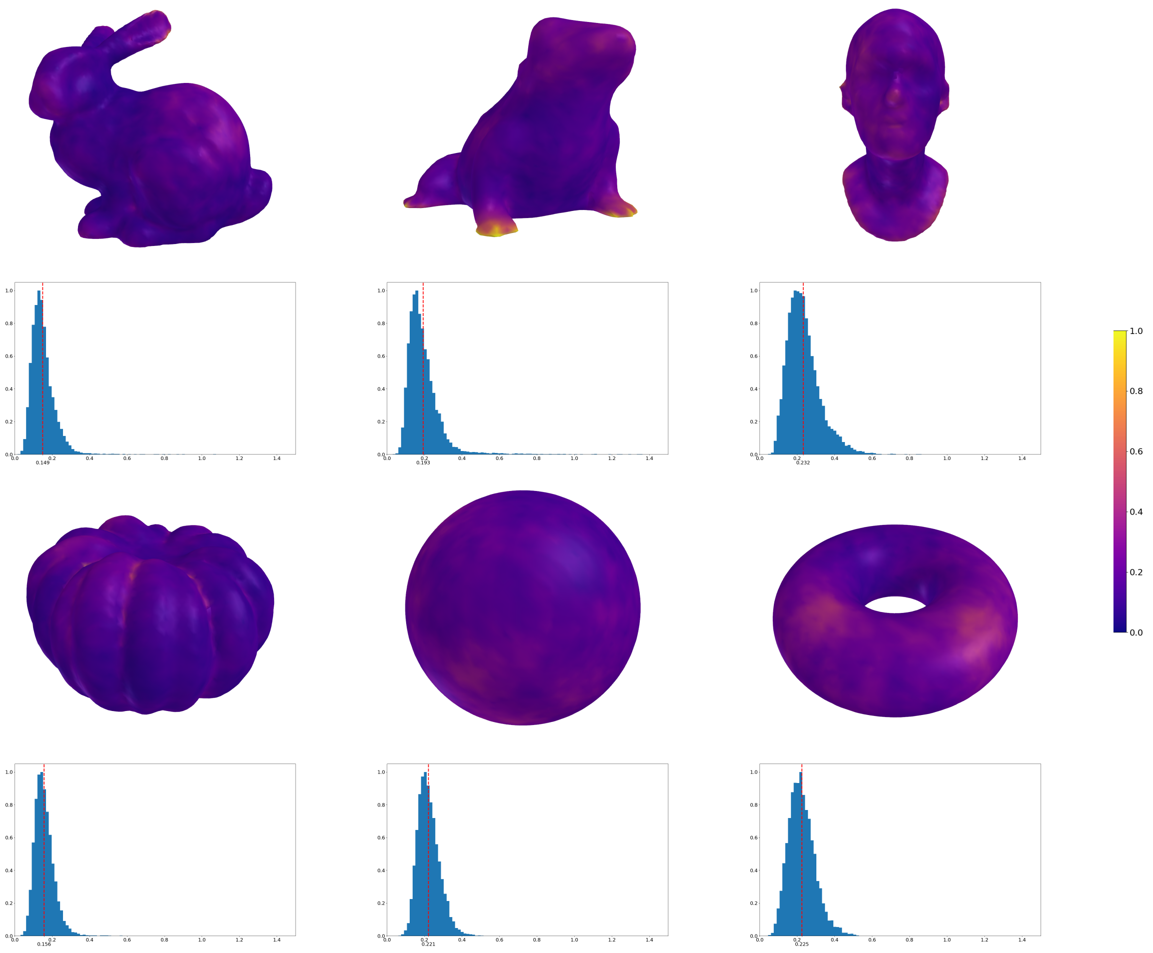
$$pdf = \frac{c_D}{N \cdot M \cdot A_D} \tag{1.16}$$

όπου $A_D$ το εμβαδόν της επιφάνειας. Επειδή η επιφάνεια και κατά συνέπεια οι περιοχές του διαμερισμού δεν περιγράφονται αναλυτικά χρησιμοποιούμε αντ' αυτού ένα mesh (με τα τρίγωνά του ως τις περιοχές) που κατασκευάζεται μέσω Marching Cubes [43]. Θεωρούμε ότι ένα σημείο ανήκει στο τρίγωνο στο οποίο βρίσκεται πιο κοντά.

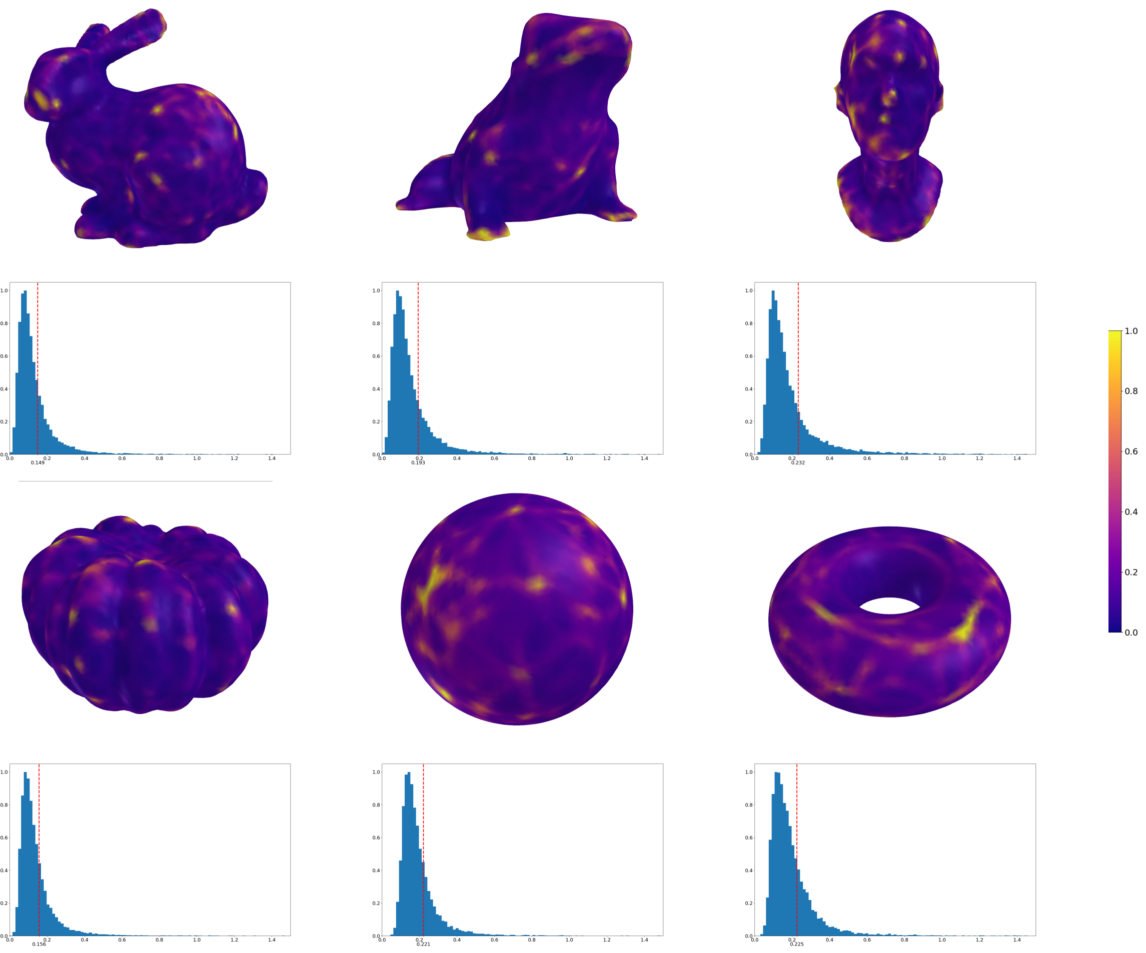
Ακολουθούμε την ίδια διαδικασία και για την περίπτωση όπου σε κάθε επανάληψη το σύνολο δειγμάτων κατασκευάζεται εξ αρχής. Τα αποτελέσματα φαίνονται στα σχήματα 1.3 και 1.4 αντίστοιχα, όπου η εκτιμώμενη τιμή της συνάρτησης πυκνότητας πιθανότητας οπτικοποιείται με τα χρώματα των τριγώνων. Στα σχήματα αυτά παραθέτουμε επίσης και ιστογράμματα των εκτιμώμενων τιμών, στα οποία με διακεκομμένη κόκκινη γραμμή φαίνεται η τιμή της πραγματικής ομοιόμορφης κατανομής. Παρατηρούμε ότι για τον αλγόριθμό μας τα χρώματα είναι πιο ομοιόμορφα και ότι τα ιστογράμματα είναι κεντραρισμένα γύρω από την τιμή της ομοιόμορφης κατανομής.

### 1.5.2 Παράμετροι Πινέλου

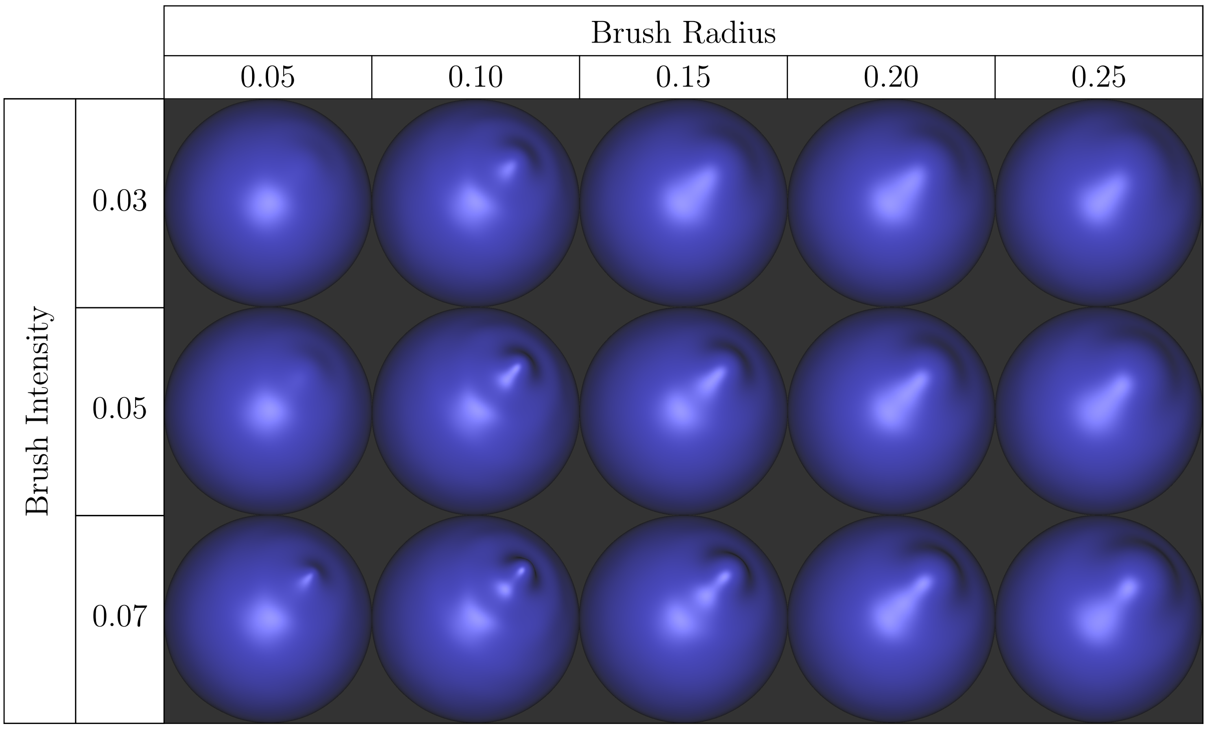
Στα σχήματα 1.5 και 1.6 παρουσιάζουμε τη δράση του πινέλου στο ίδιο σημείο επιφάνεια μίας σφαίρας για διαφορετικές τιμές της ακτίνας και της έντασης. Για το πρώτο σχήμα χρησιμοποιούμε θετικές τιμές της έντασης, οπότε το πινέλο δημιουργεί εξογκώματα, ενώ



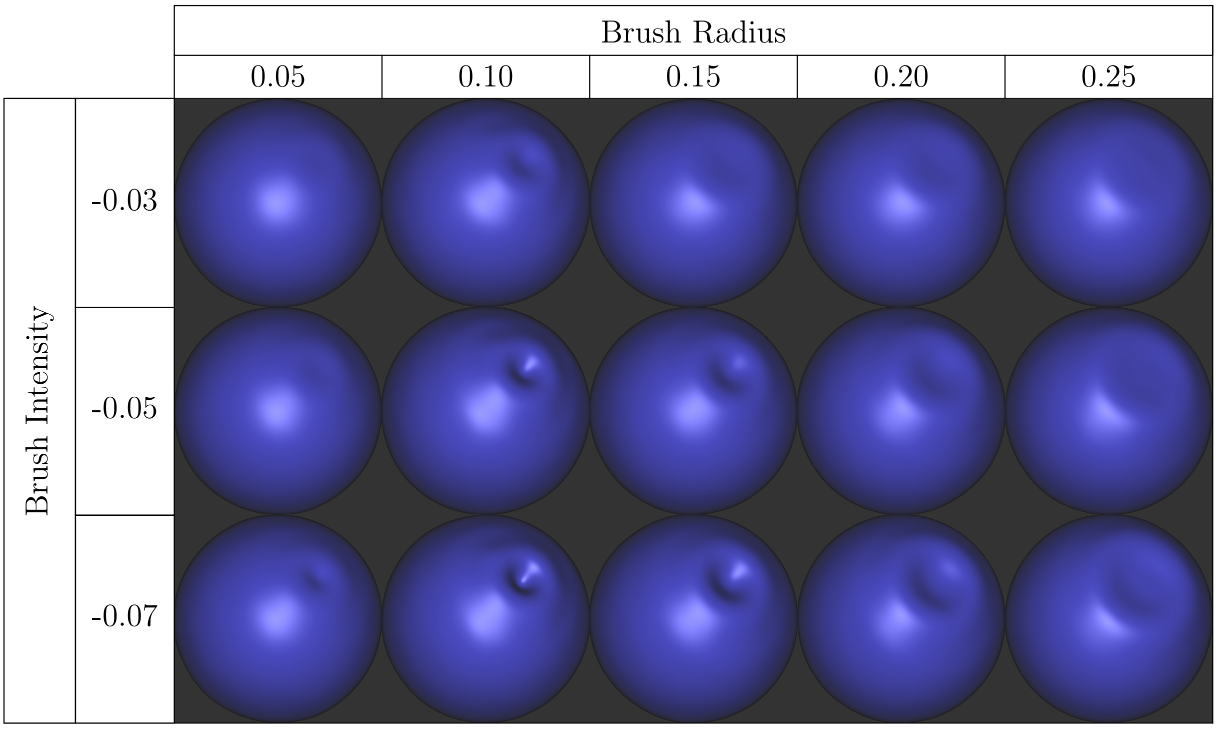
Σχήμα 1.3: Τα αποτελέσματα της εκτίμησης συνάρτησης πυκνότητας πιθανότητας για τον αλγόριθμο δειγματοληψίας μας.



Σχήμα 1.4: Τα αποτελέσματα της εκτίμησης συνάρτησης πυκνότητας πιθανότητας για την περίπτωση όπου το σύνολο δειγμάτων αρχικοποιείται σε κάθε επανάληψη.



Σχήμα 1.5: Το αποτέλεσμα της δράσης του πινέλου στο ίδιο σημείο με διαφορετικές τιμές για την ακτίνα και την ένταση. Οι τιμές τις έντασης είναι θετικές και έτσι το πινέλο δημιουργεί εξογκώματα.



Σχήμα 1.6: Όπως και στο σχήμα 1.5, αλλά με αρνητικές τιμές για την ένταση, οπότε το πινέλο δημιουργεί βαθουλώματα.

| Shape | Mean Chamfer Distance $\times 10^3$ ($\downarrow$) | | | | | |
|---|---|---|---|---|---|---|
| | Over whole surface | | | Inside interaction area | | |
| | Ours | Naive | Simple Mesh | Ours | Naive | Simple Mesh |
| Bunny | **9.407** | 14.106 | 11.127 | **5.527** | 12.919 | 17.707 |
| Frog | **8.172** | 12.865 | 8.756 | **4.750** | 9.805 | 17.051 |
| Bust | **7.279** | 11.486 | 7.901 | **3.818** | 8.779 | 14.926 |
| Pumpkin | **8.774** | 13.558 | 11.489 | **4.315** | 5.910 | 20.693 |
| Sphere | **7.209** | 12.550 | 7.399 | **3.555** | 6.117 | 12.982 |
| Torus | **7.142** | 13.516 | 7.415 | **3.574** | 5.980 | 13.402 |
| Dining chair | **7.256** | 20.545 | 8.260 | **8.843** | 11.288 | 10.703 |
| Chair | **8.498** | 30.620 | 8.650 | **8.741** | 19.072 | 15.344 |
| Armchair | **14.152** | 19.433 | 14.311 | **5.085** | 10.942 | 23.654 |
| Sofa | **11.160** | 18.268 | 11.899 | **4.477** | 8.623 | 23.940 |
| Vase | **9.063** | 15.565 | 10.345 | **10.477** | 15.713 | 16.725 |
| *Average* | **8.919** | 16.592 | 9.778 | **5.742** | 10.468 | 17.012 |

Πίνακας 1.1: Σύγκριση της μεθόδου μας με και χωρίς δείγματα επιφάνειας (Ours και Naive, αντίστοιχα) και επεξεργασία ενός mesh ισοδύναμου μεγέθους με το νευρωνικό.

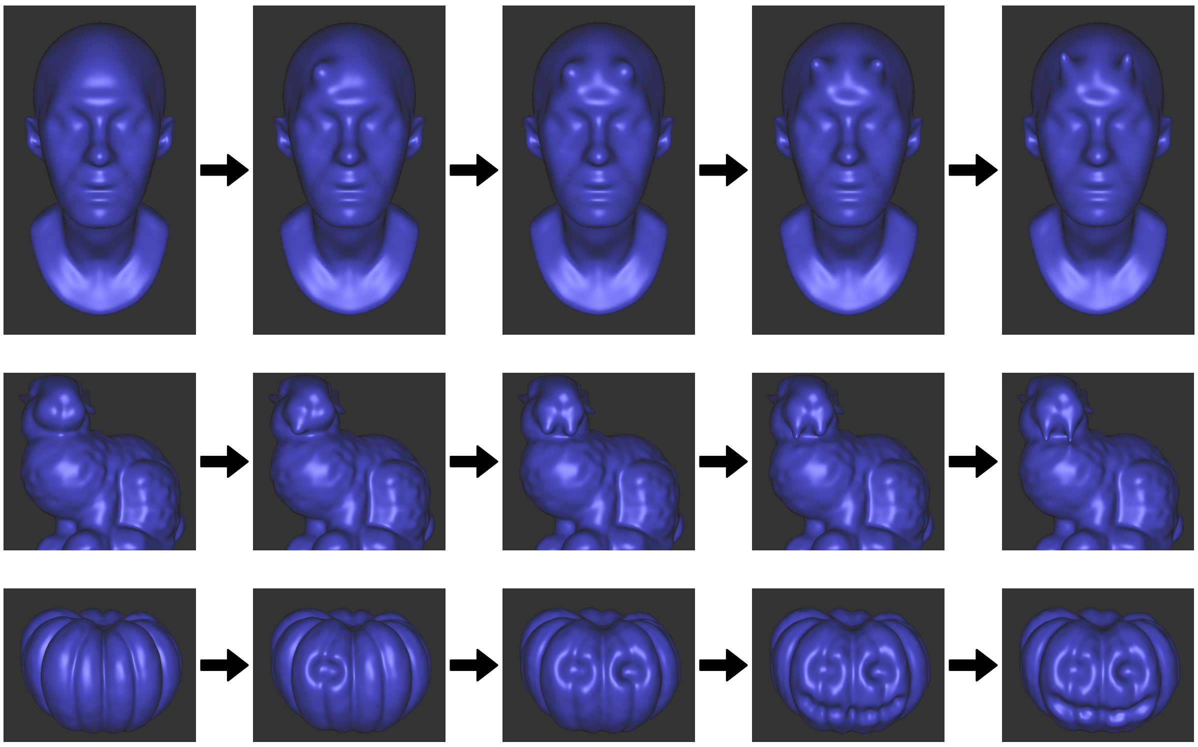
για το δεύτερο αρνητικές, οπότε το πινέλο δημιουργεί βαθουλώματα. Και στα δύο σχήμτα η ακτίνα παραμένει σταθερή στην κάθε γραμμή και η ένταση στην κάθε στήλη.

### 1.5.3 Σύγκριση με Επεξεργασία Mesh

Η πιο δημοφιλής αναπαράσταση για 3D γλυπτική είναι τα meshes. Επομένως μία σύγκριση με αυτά είναι αναγκαία για την αξιολόγηση της προτεινόμενης μεθόδου. Παρ' όλα αυτά η άμεση ποσοτική σύγκριση με εμπορικά προγράμματα επεξεργασίας mesh δεν είναι δυνατή, διότι τα πινέλα σε αυτά λειτουργούν διφορετικά από την δική μας μοντελοποίηση. Υλοποιούμε, συνεπώς, μία επεξεργασία mesh η οποία μιμείται τον τρόπο που δρα το πινέλο στη νευρωνική ΣΠΑ.

Κατά την κατασκευή των δειγμάτων αλληλεπίδρασης προβάλλουμε σημεία του εφαπτόμενου δίσκου πάνω στην επιφάνεια. Για το mesh χρειάζεται να ακολουθήσουμε την αντίθετη διαδικασία εφόσον οι κόμβοι του, τους οποίους θέλουμε να μετακινήσουμε προκειμένου να αλλάξουμε την επιφάνεια, βρίσκονται εξ αρχής πάνω στην επιφάνεια την οποία ορίζει, αλλά δεν γνωρίσουμε την τιμή της συνάρτησης πινέλου για αυτούς. Για κάθε κόμβο εντός της περιοχής αλληλεπίδρασης ακολουθούμε την ακτίνα που ξεκινάει από αυτόν με κατεύθυνση το αντίστοιχο κάθετο διάνυσμα και βρίσκουμε την τομή της με τον εφαπτόμενο δίσκο στο σημείο αλληλεπίδρασης. Χρησιμοποιούμε αυτήν την τομή για να υπολογίσουμε την τιμή του πινέλου και μετατοπίζουμε τον κόμβο αναλόγως.

Πέραν του παραπάνω, δεν είναι ξεκάθαρο πώς οι δύο αυτές αναπαραστάσεις μπορούν να συγκριθούν ουτώς ή άλλως. Επιλέγουμε να συγκρίνουμε την γεωμετρική εκφραστικότητά τους δοθέντος ενός σταθερού προϋπολογισμού μνήμης. Χρησιμοποιούμε ένα mesh υψηλής ανάλυσης ως το ground truth και ένα χαμηλής ανάλυσης ως αναφορά. Τα τέσσερα meshes



Σχήμα 1.7: Παραδείγματα πολλαπλών αλληλεπιδράσεων

του σχήματος 1.1 έχουν αρκετή ανάλυση για να χρησιμοποιηθούν ως το ground truth. Τα σχήματα του ShapeNet, όχι όμως, οπότε για αυτά, καθώς και για τη σφαίρα και τον τόρο, κατασκευάζουμε το ground truth μέσω Marching Cubes [43]. Ακολουθώντας το NGLoD [67] χρησιμοποιούμε τετραγωνικό αποδεκατισμό προκειμένου να κατασκευάσουμε το mesh αναφοράς, ώστε το μέγεθος του να είναι περίπου ίσο με αυτό του νευρωνικού.

Έχοντας κατασκευάσει το ground truth mesh και το mesh αναφοράς, εκτελούμε τις ίδιες 10 ανεξάρτητες αλληλεπιδράσεις σε αυτά, καθώς και στο νευρωνικό, χρησιμοποιώντας την μέθοδό μας αλλά και την αφέλη προσέγγιση όπου χρησιμοποιούνται μόνο δείγματα αλληλεπίδρασης. Το πινέλο έχει ακτίνα 0.08 και ένταση 0.06 για κάθε αλληλεπίδραση. Για την αριθμητική σύγκριση υπολογίζουμε την μέση Chamfer απόσταση (με 10000 δείγματα) μεταξύ του ground truth mesh και των υπολοίπων, υπολογισμένη σε όλη την επιφάνεια, καθώς επίσης μόνο μέσα στην περιοχή αλληλεπίδρασης. Τα αποτελέσματα συνοψίζονται στον πίνακα 1.1, όπου φαίνεται ότι η μέθοδός μας προσφέρει καλύτερα αποτελέσματα.

### 1.5.4 Παραδείγματα Πολλαπλών Αλληλεπιδράσεων

Στο σχήμα 1.7 φαίνονται τα αποτελέσματα πολλών αλληλεπιδράσεων σε τρία σχήματα. Χρησιμοποιήθηκαν διάφορες τιμές για τις παραμέτρους τους πινέλου.

## 1.6 Κατακλείδα

Στην παρούσα εργασία παρουσιάσαμε την πρώτη, απ' όσο γνωρίζουμε, μέθοδο για διαδραστική επεξεργασία νευρωνικών ΣΠΑ, καθώς και ένα αλγόριθμο ομοιόμορφης δειγματοληψίας τους. Δείξαμε μέσω πειραμάτων τα οφέλη που προσφέρει έναντι των πλέον διαδεδομένων τεχνικών. Ελπίζουμε ότι με αυτόν τον τρόπο η χρήση τους θα διαδοθεί περισσότερο σε εμπορικές αλλά και επιστημονικές εφαρμογές. Υπάρχουν ήδη εργασίες ρομποτικής οι οποίες χρησιμοποιούν τέτοιες αναπαραστάσεις. Βέβαια, προκειμένου να υπάρξει η υιοθέτηση αυτών των καινοτόμων τεχνικών χρειάζεται περισσότερη έρευνα. Υπάρχει μεγάλο περιθώριο βελτίωσης και πειραματισμού. Νεώτερες αρχιτεκτονικές οι οποίες χρησιμοποιούν χωρικές δομές μπορούν να αναπαραστήσουν περισσότερες λεπτομέρειες και είναι ενδεχομένως καταλληλότερες. Επίσης η μοντελοποίηση των πινέλων μπορεί να επεκταθεί επιτρέποντας πολυπλοκότερη επεξεργασία. Η εξερεύνηση αυτών των μονοπατιών είναι ο μόνο δρόμος προς την ευρεία υιοθέτηση των ΣΠΑ.

# Chapter 2

# Introduction

*Formless protoplasm able to mock and reflect all forms and organs and processes—viscous agglutinations of bubbling cells—rubbery fifteen-foot spheroids infinitely plastic and ductile—slaves of suggestion, builders of cities—more and more sullen, more and more intelligent, more and more amphibious, more and more imitative—Great God! What madness made even those blasphemous Old Ones willing to use and to carve such things?*

H. P. Lovecraft, *At the Mountains of Madness*

The wings of a butterfly, the petals of a flower, a human visage, and the antlers of a deer, as well as, the chassis of car, a mechanical gear, and the walls of a building are but a meagre set of examples of natural and human-made objects that occupy the interest of people working on a wide range of fields. An artist might want to create a butterfly which is going to fly across a scene in an animated film. Depictions of plants are useful to botanologists and have been included since ever in related books. A mechanical engineer, on the other hand, will want to design the chassis of a car and test its aerodynamic properties, and an architect will want to design a building. The question, then, is how to represent these objects, since, the purpose is not to procure the object itself, or, even if it is, its design will undergo various iterations?

In the days of yore, before the digital era and the incorporation of computers to the workflow of virtually any task, the answer to the above question would be pen and paper and/or physical construction (usually in smaller scale). Indeed, even today, this keeps being the answer sometimes. Maquettes, for example, are used to test aircraft designs. However, computers have provided more sophisticated approaches and capabilities that were, simply, impossible by traditional means. Computer animation, computer games, computer aided design (CAD), architectural visualization, etc., are improved constantly and influencing each other producing along the way ever more impressive results. In these digital environments, the objects of interest are, in the vast majority, represented by their surface, and it is the problem of *surface representation* for computers that lies at the core of the present thesis.

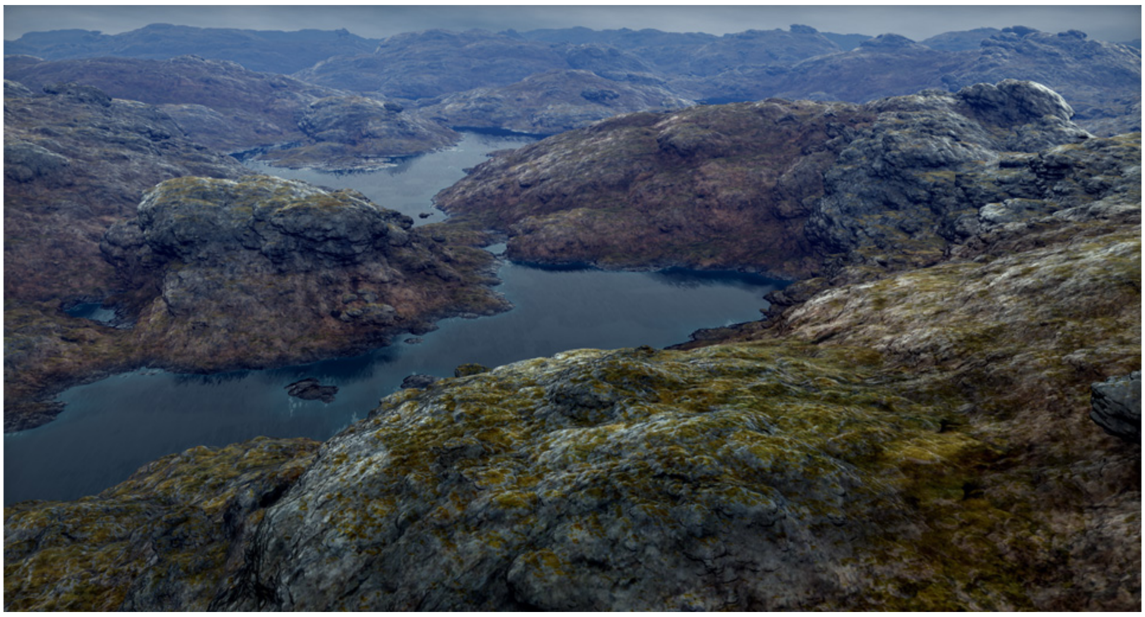


Figure 2.1: The terrain in the rendering above is described through its signed distance function. Taken from [54].

As already discussed, surface representation is a problem encountered in many and diverse applications. It is only natural, therefore, that the representations devised over the years are as many and diverse as the applications, each with their respective pros and cons. Bézier patches, B-splines and octrees are only some of the choices with the most ubiquitous being meshes.

Relatively recently, given the ever rising popularity of artificial neural networks, a new class of surface representations has been proposed. In this approach, the surface, which is frequently required to be closed, is represented *implicitly* as the level set of the function of a neural network with one output. Several papers have presented very interesting and promising results using such representations. In most of these works, the network tries to learn either the signed distance function or the occupancy function. The signed distance function measures, for every point, its distance to the surface, negating it if the point lies inside the volume the surface encloses. Accordingly, the occupancy function takes the value 0 outside the surface and the value 1 inside the surface (or vice versa), which can be thought as thresholding the signed distance function. In this thesis we focus on the former. Besides representing the surface of one shape, these networks can also represent an entire class of shapes by taking a class code along with the spatial coordinates as input.

Representing a surface using its SDF is not a new idea. Prior works have used octrees that hold the SDF's value at the positions of their nodes. Also, shader artists, who use GPU shader programs to create art, have used analytic SDF expressions to render complex scenes (see figures 2.1 and 2.2). Coupling SDFs with neural networks, however, has proved more versatile.

Nonetheless, being able to represent a shape, usually, does not constitute a full solution to the problem. Even a representation able to represent the most minute details efficiently

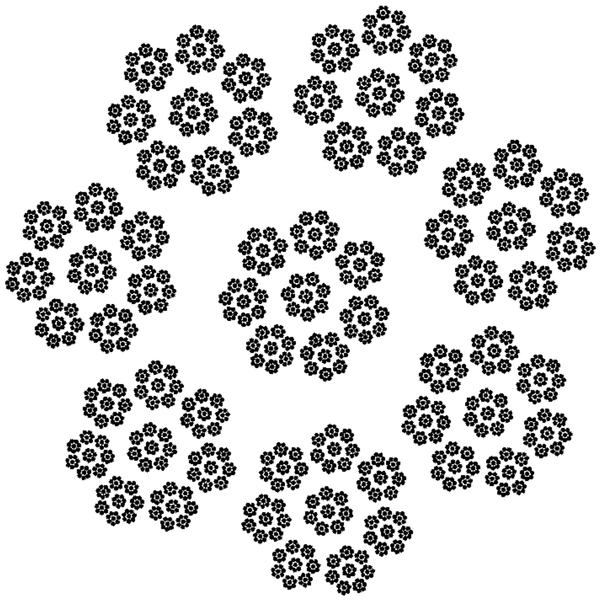


Figure 2.2: A fractal set rendered using its signed distance function.

would be totally useless for a specific application if it would not provide the features that the application requires. The different features or properties that a representation offers relative to others is exactly why different representations are suitable for different tasks. Most notably, it should be possible to render the represented surface, in order to examine it visually. Since we are dealing with 3D shapes, inspecting them visually is crucial to just about any application. Another feature that is also required by most applications is the ability to edit or create the represented surface manually with interactive software. The type of editing can be quite different, though. For some tasks, for example precise editing of the individual mesh vertices is more appropriate, while for others, tools that specify the end-effect, treating the representation opaquely and disregarding the internals details, are more useful. It is the latter manner of editing that concerns us in this thesis.

Despite the success of neural SDFs, their editability has not been particularly studied. There exist some works that allow the user to control a number of parameters in order to change the shape. However, the outcome belongs in a learned space of shapes. Essentially, these works allow interactive exploration of this space, without the possibility of modifying the shape beyond its confines. For example, if the network has been trained to represent airplanes it will not be possible change the shape into a car. Providing methods that allow ad hoc editing of neural SDFs are critical in making these representations more widespread and in this thesis we attempt to do just that. Specifically, we implement editing capabilities for neural SDFs inspired from 3D sculpting. This way of editing uses tools called brushes to deform the surface locally. Due to its nature, it is directed mainly towards artistic applications. We partly aim at making neural SDFs a viable representation for these applications. This does not preclude the possibility of the use of our method in scientific applications as well. After all, neural SDFs have found use in robotics already and our proposed method can provide local corrections. Summarily, our contributions are:

- A self-contained analysis of the basic properties of distance functions.
- An algorithm for quasi-uniform sampling of a surface represented with a neural SDF.
- Mathematical analysis and experimental examination of the distribution produced by the algorithm.
- A method for editing locally the surface of a neural SDF based on 3D sculpting.
- A quantitative comparison with the corresponding mesh editing.

These results have also been compiled into a paper [73]. The rest of this thesis is organized as follows:

- In Chapter 3, we present some mathematical foundation along with a brief discussion of neural networks.
- In Chapter 4, we review prior work on surface representation, focusing on research on neural SDFs.
- In Chapter 5, we describe our proposed method in detail.
- In Chapter 6, we perform experiments to evaluate our method quantitatively and qualitatively.

# Chapter 3

# Preliminaries

## Contents

In this chapter, we aim to provide the theoretical background that is required for the comprehension of the following chapters. We will focus on machine learning and some mathematical ideas, definitions, and procedures pertinent to our work. More specifically, in Section 3.1 we discuss distance functions and their properties, afterwards, in Section 3.2, we deal with sampling algorithms, and, finally, we summarily present machine learning with a focus on neural networks, in Section 3.3. We try to keep our discussion self-contained. This is a difficult task when handling such diverse topics, however, so we assume a basic knowledge of linear algebra, vector calculus and probability theory and we provide relevant citations when necessary to fill in the gaps.

## 3.1 Distance Functions

We begin by defining a notion that is central to this work and permeates almost every aspect. The notion of a distance function to a surface. Firstly, we shall start quite abstractly with metric spaces in general and, then, move on to a more concrete setting.

**Definition 3.1: Metric Space**

A metric space $(M, d)$ is a set $M$ along with a binary real function $d : M \times M \to \mathbb{R}$ which has the following properties:

(i) $d(x, y) = d(y, x), \forall x, y \in M$

(ii) $d(x, y) = 0 \iff x = y, \forall x, y \in M$ (Symmetry)

(iii) $d(x, y) + d(y, z) \geq d(x, z), \forall x, y, z \in M$ (Triangle inequality)

A function $d$ satisfying these properties is called a metric.

**Lemma 3.2**

Let $(M, d)$ be a metric space, then $\forall x, y \in M$, then:

$$d(x, y) \geq 0$$

*Proof.*

$$\begin{aligned}
&d(x, y) + d(y, x) \geq d(x, x)\\
&d(x, y) + d(y, x) \geq 0\\
&2d(x, y) \geq 0\\
&d(x, y) \geq 0
\end{aligned}$$

□

**Lemma 3.3: Continuity of the Metric**

Let $x_i, y_i \in \mathbb{R}^3, i \in \mathbb{N}$ be two sequences converging to $x_l, y_l$, respectively, then:

$$\lim d(x_i, y_i) = d(x_l, y_l)$$

that is, the metric is a continuous function on $M \times M$

*Proof.* Let $\varepsilon > 0$. There exists $j \in \mathbb{N}$ such that $d(x_i, x_l) < \frac{\varepsilon}{2}$ and $d(y_i, y_l) < \frac{\varepsilon}{2}, \forall i > j$

$$\begin{aligned} d(x_i, y_i) &< d(x_i, x_l) + d(x_l, y_l) + d(y_l, y_i) \\ &< \frac{\varepsilon}{2} + d(x_l, y_l) + \frac{\varepsilon}{2} \\ &< d(x_l, y_l) + \varepsilon \end{aligned}$$

Hence, $d(x_i, y_i) - d(x_l, y_l) < \varepsilon$
By reasoning, symmetrically for $d(x_l, y_l)$, we get $d(x_l, y_l) - d(x_i, y_i) < \varepsilon$ □

A metric space is a general concept that can model any set where a distance between its elements can be defined. Examples of metric spaces are Euclidean spaces and $L_p$ function spaces.

Next, we define the distance of a point (that is an element of the space) to a subset A of the space.

**Definition 3.4: Distance to a subset**

Let $(M, d)$ be a metric space, $x \in M$ and $A \subset M$, then we define the distance of of $x$ to $A$ as follows:

$$d(x, A) = \inf_{y \in M} \{d(x, y)\}$$

*Remark* (1). Note that we use $d$ to represent both the distance between points and that between a point and a set. Since the arguments differ, the specific distance should be clear from the context without any ambiguity arising.

*Remark* (2). It is clear from Lemma 3.2 that $d(x, A) \geq 0$.

*Remark* (3). If $A$ is closed then the min can be used in place of inf.

**Lemma 3.5**

Let $(M, d)$ be a metric space, then $\forall x \in M, A \subset M$, with $A$ closed:

$$d(x, A) = 0 \iff x \in A$$

*Proof.*

(i) $d(x, A) = 0 \Rightarrow x \in A$

$$\begin{aligned}&d(x, A) = 0\\&\exists\, y \in A : d(x, y) = 0\\&x = y\\&x \in A\end{aligned}$$

(ii) $x \in A \Rightarrow d(x, A) = 0$

$$\begin{aligned}&x \in A\\&d(x, A) \leq d(x, x) = 0\\&d(x, A) = 0\end{aligned}$$

□

For a closed set $A$, the minimum distance can be obtained with more than one point of $A$, that is, the points of $A$ that are closest to a point $x$ might contain more than one point. It is useful to characterize the elements of the space by the cardinality of the set of closest surface points. More specifically, we have the following definitions:

**Definition 3.6**

Let $(M, d)$ be a metric space, $x \in M$ and $A \subset M$, with $A$ closed. The set of points of $A$ closest to $x$ is defined as follows:

$$\mathcal{C}(x, A) = \{y \in A \,|\, d(x, y) = d(x, A)\} = \underset{y \in A}{argmin}\{d(x, y)\}$$

*Remark.* $|\mathcal{C}(x, A)| \geq 1$

**Definition 3.7: Regular and Singular points**

Let $(M, d)$ be a metric space and $A \subset M$, closed. A point $x \in M$ shall be called:

- *regular*, if $|\mathcal{C}(x, A)| = 1$
- *singular*, otherwise.

Having set the general basis of our tools we will move on to our space of interest which is $\mathbb{R}^3$ endowed with the Euclidean metric. Henceforth, when we use $d$ for the distance between points we shall take it to mean the Euclidean metric unless explicitly mentioned otherwise.

In this space we shall define our objects of interest, that is surfaces, as follows (for a discussion of dimension refer to [25, 26]):

**Definition 3.8: Surface**

A surface is a compact 2-dimensional subset of $\mathbb{R}^3$

The above is a very general definition that seeks to encompass any set that intuitively would be characterized as surface. We state it thusly to comply with the existing bibliography, however, we need to limit our attention by adding two further restrictions: closedness (not to be confused with topological closedness) and connectedness. A closed surface is one that separates the space into two regions, one of which is bounded, whose interface is the surface. We call the bounded region the interior of the surface and the other one exterior, accordingly. A connected surface is one for which there exists a path from every point on it to every other, and the path is on the surface as well. For example, a hemisphere, see figure 3.1a, is an open surface, while a sphere is closed. Also, if we were to consider two spheres as one surface, as in figure 3.1b, this would be a non-connected one. Henceforth, when we use the term *surface* we will take it to mean closed and connected unless explicitly stated otherwise.

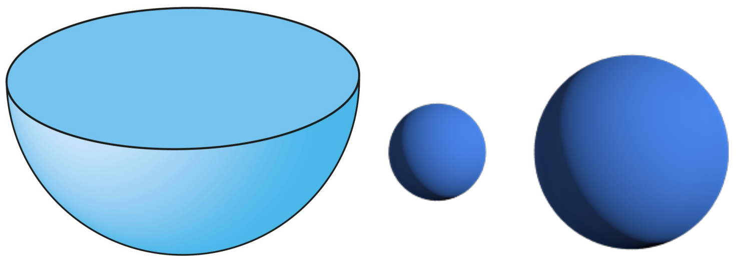

(a) Open surface (b) Non-connected surface

Figure 3.1: Examples of open and non-connected surfaces

Since a surface is a set, definition 3.4 for distance from a point to a set applies to it as well. For surfaces (closed specifically), however, we can define a signed version of the distance function as follows:

**Definition 3.9: Signed Distance Function**

Let $S \subset \mathbb{R}^3$ be a surface and $x \in \mathbb{R}^3$, then the distance from $x$ to $S$ is:

$$d_s(x, S) = \begin{cases} d(x, S) & \text{if x outside } S \\ -d(x, S) & \text{otherwise} \end{cases}$$

*Remark.* $d(x, S) = |d_s(x, S)|$

We will be using the acronyms UDF (Unsigned Distance Function) and SDF (Signed Distance Functions) for functions of definitions 3.4 and 3.9, respectively, and DF (distance function) for the both of them.

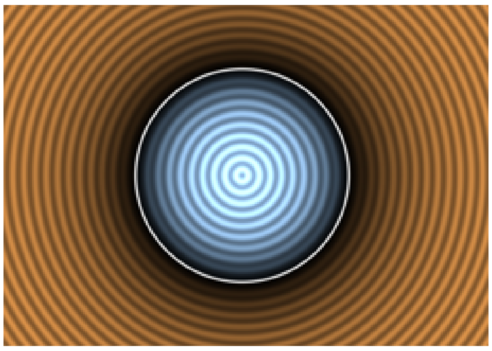


Figure 3.2: 2D profile of a sphere's signed distance function.

A 2D profile of the signed distance function to a sphere is depicted in figure 3.2. Negative values of the function are shown in shades of blue and positive in shades of orange.

### 3.1.1 DF Properties

We are going to investigate a number of properties that DFs satisfy and will be useful further on in this thesis. In summary, what we will show is that DFs are differentiable almost everywhere with unit norm gradients that point away from the closest surface point and towards it, where the SDF takes negative values. More specifically the set of points where a DF is differentiable is a subset of the regular points. However, proving these facts is quite involved, so we are going to need some theorems and lemmas, before tackling them.

**Theorem 3.10: SDFs are Lipschitz**

The SDF of a surface $S$ is a Lipschitz function with Lipschitz constant of 1, that is, $\forall x, y \in \mathbb{R}^3$:

$$|d_s(x,S) - d_s(y,S)| \leq d(x,y)$$

*Proof.* We are going to split the proof in two cases:

(i) $d_s(x,S), d_s(y,S)$ have the same sign

Then $|d_s(x,S) - d_s(y,S)| = |d(x,S) - d(y,S)|$

Without loss of generality we assume that $|d(x,S) - d(y,S)| = d(x,S) - d(y,S)$

Let $z \in \mathcal{C}(y,S)$, which means that $d(y,z) = d(y,S)$, then by the triangle inequality:

$$d(x,z) \leq d(x,y) + d(y,z) = d(x,y) + d(y,S)$$
$$d(x,z) - d(y,S) \leq d(x,y)$$

But $d(x,S) \leq d(x,z)$

(ii) $d_s(x,S), d_s(y,S)$ have different signs

Then $|d_s(x,S) - d_s(y,S)| = d(x,S) + d(y,S)$

Since the signed distances have different signs then one of $x, y$ is inside the surface and one outside of it. The line that passes them intersects the surface at some point $z \in S$. Since $z$ lies between $x$ and $y$, $d(x,z) + d(z,y) = d(x,y)$. But, also, $d(x,S) \leq d(x,z)$ and $d(y,S) \leq d(y,z) = d(z,y)$

$\square$

**Corollary 3.11**

Let $S$ be a surface and $x \in \mathbb{R}^3$ where the SDF is differentiable, then:

$$\|\nabla d_s(x, S)\| \leq 1$$

*Proof.* The norm of the gradient is equal to the directional derivative along the direction of the gradient, that is:

$$\|\nabla d_s(x, S)\| = \nabla_v \, d_s(x, S), \text{ where } v = \frac{\nabla d_s(x, S)}{\|\nabla d_s(x, S)\|}$$

But, we have the following for the directional derivative:

$$\begin{aligned} \nabla_v \, d_s(x, S) &= \lim_{t \to 0} \frac{d_s(x + tv, S) - d_s(x, S)}{t} \\ &= \lim_{t \to 0} \frac{d_s(x + tv, S) - d_s(x, S)}{d(x + tv, x)} \\ &\leq 1 && \text{(by the Lipschitz property)} \end{aligned}$$

□

*Remark.* We will eventually prove that $\|\nabla d_s(x, S)\|$ is exactly equal to 1 almost everywhere. The above result provides a bound and will be used to that purpose.

The proof of the Lipschitz property for a UDF is essentially the first case of the proof above and the corollary follows in the same way. By Rademacher's Theorem [27], in conjunction with the one we just proved, DFs are differentiable almost everywhere. We now move a step further, in order to characterize the sets of points where an DF is or is not differentiable.

We begin by proving the following interesting lemma and, then, some more consequential results:

**Lemma 3.12**

Let $S$ be a surface and $x_i \in \mathbb{R}^3, i \in \mathbb{N}$ be a sequence converging to $x_l$, then there exists an increasing sequence $k_i \in \mathbb{N}$ and a sequence $y_{k_i} \in \mathcal{C}(x_{k_i}, S)$ converging to $y_l \in \mathcal{C}(x_l, S)$

*Proof.* By the axiom of choice, there exists a sequence $y_i \in \mathcal{C}(x_i, S)$. This sequence is bounded because $S$ is a compact set.

Now, by the Bolzano-Weierstrass Theorem, $y_i$ has a convergent subsequence $y_{k_i}$. Let $y_l = \lim y_{k_i}$. We need to prove that $y_l \in \mathcal{C}(x_l, S)$. We will prove this by contradiction.

Assume that $y_l \notin \mathcal{C}(x_l, S)$, then, $d(x_l, y_l) - d(x_l, y_c) = \varepsilon > 0$, with $y_c \in \mathcal{C}(x_l, S)$

By the convergence of $x_i$ and the continuity of the metric, there exists $j \in \mathbb{N}$ such that:

$$d(x_{k_j}, x_l) < \frac{\varepsilon}{2} \text{ and } d(x_l, y_l) < d(x_{k_j}, y_{k_j}) + \frac{\varepsilon}{2}$$

We, then, have:

$$\begin{aligned} d(x_{k_i}, y_c) &\leq d(x_{k_j}, x_l) + d(x_l, y_c) \\ &= d(x_{k_j}, x_l) + d(x_l, y_l) - \varepsilon \\ &< \frac{\varepsilon}{2} + d(x_{k_j}, y_{k_j}) + \frac{\varepsilon}{2} - \varepsilon \\ &= d(x_{k_j}, y_{k_j}) \end{aligned}$$

It follows that $y_{k_j} \notin \mathcal{C}(x_{k_i}, S)$, which is a contradiction.

Therefore, $y_l \in \mathcal{C}(x_l, S)$. □

**Theorem 3.13**

Let $S$ be a surface, $x \in \mathbb{R}^3$ and $y \in \mathcal{C}(x, S)$, then $y \in \mathcal{C}(z, S)$ for every $z$ on the line segment connecting $x$ and $y$. Furthermore, if $x$ is regular, then every point on the line segment is regular as well.

*Proof.* Assume that $y \notin \mathcal{C}(z, S)$, then there exists a $w \in \mathcal{C}(z, S)$ such that $d(z, S) = d(z, w) < d(z, y)$

$$\begin{aligned} d(x, w) &\leq d(x, z) + d(z, w) \\ &< d(x, z) + d(z, y) \\ &= d(x, y) \end{aligned}$$

It follows that $y \notin \mathcal{C}(x, S)$, which is a contradiction.

Now, assume that $x$ is regular and that $z$ is not. This entails that there exists a $w \in \mathcal{C}(z, S)$ such that $d(z, S) = d(z, w) = d(z, y)$ and $w \neq y$. By the same reasoning as above, we, once again, arrive at a contradiction. □

**Corollary 3.14**

Let $S$ be a surface, $x \in \mathbb{R}^3$ and $y \in \mathcal{C}(x, S)$, and $v = \frac{y-x}{\|y-x\|}$ then:

$$\lim_{t \to 0^+} \frac{d(x + tv, S) - d(x, S)}{t} = -1$$

*Proof.* By the above theorem, $y \in \mathcal{C}(x + tv, S)$ and thus:

$$d(x, S) = d(x, y) \text{ and } d(x + tv, S) = d(x + tv, y)$$

Hence:

$$\begin{aligned}\lim_{t\to 0^+} \frac{d(x+tv,S)-d(x,S)}{t} &= \lim_{t\to 0^+} \frac{d(x+tv,y)-d(x,y)}{t} \\ &= \lim_{t\to 0^+} \frac{-t}{t} \\ &= -1\end{aligned}$$

□

**Corollary 3.15**

Let $S$ be a surface, $x \in \mathbb{R}^3$ and $y \in \mathcal{C}(x,S)$ be a singular point of its UDF, then the UDF is not differentiable at $x$.

*Proof.* Since $x$ is a singular point there exist two distinct $y, z \in \mathcal{C}(x,S)$. Even if $x, y, z$ are colinear $y$ and $z$ cannot lie on the same side of $x$, because one of them would be further then the other. So, we have two separate directions $v_y = \frac{y-x}{\|y-x\|}$ and $v_z = \frac{z-x}{\|z-x\|}$. By applying the corollary above, we get:

$$\lim_{t\to 0^+} \frac{d(x+tv_y,S)-d(x,S)}{t} = -1 \text{ and } \lim_{t\to 0^+} \frac{d(x+v_z,S)-d(x,S)}{t} = -1$$

As we have shown in corollary 3.11, -1 is the minimum value that this limit, which is the directional derivative, can take. However, this cannot happen at two different directions, and therefore, the UDF is not differentiable at $x$. □

*Remark.* Since, as discussed above, the UDF is differentiable almost everywhere, it follows that the set of singular points has measure 0.

Finally, we are in position to prove the following result.

**Theorem 3.16**

Let $S$ be a surface, $x \in \mathbb{R}^3$ where its UDF is differentiable and $y$ be the only element of $\mathcal{C}(x,S)$, then:

$$\nabla d(x,S) = \frac{x-y}{\|x-y\|}$$

*Proof.* Let $v = \frac{y-x}{\|y-x\|}$. By corollary 3.14, $\nabla_v d(x,S) = -1$ and by corollary 3.11, this is the minimum value the directional derivative can take. Therefore, the gradient has norm equal to 1 and direction opposite to $v$. □

It is straightforward to see that the analogous result for the SDF is:

$$\nabla d_s(x,S) = \text{sign}(d_s(x,S))\frac{x-y}{\|x-y\|}$$

This means that the gradient of the SDF at a point $x$ points away from the closest point of $x$, when $x$ is outside the surface, and towards it when $x$ is inside the surface.

The properties we have proved in this section also show that the UDF and the SDF of a surface are solutions to the following form of eikonal equation:

$$\begin{aligned} &f(x) = 0, \, x \in S \\ &\|\nabla f(x)\| = 1 \end{aligned} \tag{3.1}$$

If we further require that the solution to the equation is differentiable on S, then only the SDF is a solution.

We conclude this section by making an interesting remark. Even though, throughout this section we worked in $\mathbb{R}^3$, the definitions and proofs we have given are valid in any Euclidean space and for sets that are more general than what we defined as a surface.

## 3.2 Sampling

In this section, we shift our focus to sampling. Sampling, in our context, refers to the process of generating values, be they real numbers, points on some multidimensional space or something more complex, usually with some restriction on the way they are distributed. Sampling can be split in two big categories:

- Deterministic
- Random

Deterministic sampling techniques use predefined patterns to position the samples. For example, taking uniformly spaced numbers on the real line. More elaborate techniques use low discrepancy sequences, like Halton sequences. Examples of such sequences applied to rendering can be found in [52]. Random sampling, on the other hand, uses stochasticity to produce the samples according to some probability distribution.

In this section we are going to deal with random sampling and various algorithms pertaining to that. More specifically, we will present some general techniques and then apply them to sampling various shapes.

### 3.2.1 General Techniques

#### Rejection Sampling

A well known way of getting samples that lie inside of the unit disk centered at the origin, is to sample two values $x, y$ from a uniform distribution over $[-1, 1]$ until $\sqrt{x^2 + y^2} \leq 1$. Due to this process involving rejecting samples until a condition is met, the technique is called *rejection sampling*. A simple version of this technique that distributes samples uniformly inside a 2D shape bounded by $[-1,1]^2$, is given by the following algorithm:

**Algorithm 3.1:** Simple Rejection Sampling

```
Data: $S$ : 2D shape
1 do
2     $x \leftarrow$ sample from $Unif([-1,1])$
3     $y \leftarrow$ sample from $Unif([-1,1])$
4 while $(x,y)$ outside $S$
```

Rejection sampling can, however, be formulated more generally. So generally, in fact, that it allows us to simulate any distribution if we are able to get samples from any other, with only a few restrictions on the relation these two must have. There is a downside, nevertheless, and that is that this sampling technique might be inefficient, because multiple samples might be rejected before one is accepted. This can be demonstrated easily, if we extend the aforementioned algorithm for sampling the unit disk, to sample unit hyperballs. As is well known, the ratio of the volume of the unit $n$-dimensional hyperball to the volume of the box $[-1,1]^n$ goes to 0 as $n$ goes to infinity. Hence, for larger $n$, more samples will be rejected, on average, before accepting one. The general form of rejection sampling is summarized in the following theorem:

**Theorem 3.17: Rejection Sampling**

Let $\{X_i\}_1^\infty$ be *i.i.d.* according to $f(x)$ and $\{Y_i\}_1^\infty$ be *i.i.d.* uniformly over $[0,1]$. Also, let $g(x) \leq Mf(x)$ for some density $g(x)$ and $M > 0$. If $Z$ is equal to the first $X_i$, for which $Y_i \leq \frac{g(X_i)}{Mf(X_i)}$, then $Z \sim g$

*Proof.* Firstly, we compute the acceptance probability:

$$\begin{aligned}
\mathbb{P}\left[Y_i \leq \frac{g(X_i)}{Mf(X_i)}\right] &= \int_\Omega \int_0^{\frac{g(x)}{Mf(x)}} f(x)\,dy\,dx \\
&= \int_\Omega \int_0^{\frac{g(x)}{Mf(x)}} dy f(x)\,dx \\
&= \int_\Omega \frac{g(x)}{Mf(x)} f(x)\,dx \\
&= \int_\Omega \frac{g(x)}{M}\,dx \\
&= \frac{1}{M}\int_\Omega g(x)\,dx \\
&= \frac{1}{M}
\end{aligned}$$

Let $T$ be the index of the first $X_i$ that is accepted. Then, $T$ follows a geometric distribution with expected value $M$.

We can, now, find the joint distribution of $Z$ and $T$. If $E$ is a measurable set, then:

$$\begin{aligned}
\mathbb{P}\left[Z \in E \vee T = t\right] &= \mathbb{P}\left[Z \in E \,|\, T = t\right]\mathbb{P}\left[T = t\right] \\
&= \mathbb{P}\left[X_t \in E \,|\, T = t\right]\mathbb{P}\left[T = t\right] && (Z = X_t) \\
&= \mathbb{P}\left[X_t \in E \,\middle|\, Y_t \leq \frac{g(X_t)}{Mf(X_t)}\right]\mathbb{P}\left[T = t\right] && (X_t \text{ independent of prev. } X_i, Y_i) \\
&= \frac{\mathbb{P}\left[Y_t \leq \frac{g(X_t)}{Mf(X_t)} \vee X_t \in E\right]}{\mathbb{P}\left[Y_t \leq \frac{g(X_t)}{Mf(X_t)}\right]}\mathbb{P}\left[T = t\right] \\
&= \frac{\int_E \int_0^{\frac{g(x)}{Mf(x)}} f(x)\,dy\,dx}{\frac{1}{M}}\mathbb{P}\left[T = t\right] \\
&= \frac{\frac{1}{M}\int_E g(x)\,dx}{\frac{1}{M}}\mathbb{P}\left[T = t\right] \\
&= \int_E g(x)\,dx\,\mathbb{P}\left[T = t\right]
\end{aligned}$$

Ergo, $Z$ and $T$ are independent, and $Z$ is distributed according to $g(x)$ □

*Remark.* The distribution $f$ is referred to as the proposal distribution and $g$ as the target distribution.

In the process of proving the correctness of rejection sampling, we have also proven the following corollary:

**Corollary 3.18**

The number of samples required for rejection sampling follows a geometric distribution with expected value $M$.

Intuitively, the above corollary tells us that, on average, we need to get $M$ samples before finding a usable one. Hence, it is beneficial to use the smallest $M$ possible. What we need, is:

$$\frac{g(x)}{Mf(x)} \leq 1 \iff \frac{g(x)}{f(x)} \leq M \tag{3.2}$$

which implies $M$ is an upper bound of $\frac{g(x)}{f(x)}$. Naturally, the optimal $M$ is the least upper bound:

$$M_{opt} = \sup \frac{g(x)}{f(x)} \tag{3.3}$$

Equation 3.2 indirectly implies, also, that $f$ dominates $g$, that is, the support of $f$ includes the support of $g$. This is quite understandable, since, the proposal distribution must produce samples anywhere that is possible, according to the target distribution. Now, we will rely on what was just discussed to show that the supremum of equation 3.3

must be at least 1.

**Theorem 3.19**

Let $f$, $g$ be two probability distributions, with $f$ dominating $g$, then:

$$\sup \frac{g(x)}{f(x)} \geq 1$$

*Proof.* Let $D$ and $E$ be the support of $f$ and $g$, respectively, then, since $E \subseteq D$:

$$\int_E f(x)\,dx \leq \int_D f(x)\,dx = \int_E g(x)\,dx = 1$$

If $\sup \frac{g(x)}{f(x)} = s < 1$, then:

$$g(x) \leq sf(x) < f(x), \forall x \in E$$

But, then:

$$\int_E g(x)\,dx < \int_E f(x)\,dx$$

which is a contradiction. □

At this point, it is quite reasonable to wonder how the general form of rejection sampling, that we have proved in Theorem 3.17, relates to the algorithm we presented at the beginning for uniformly sampling the unit disk centered at the origin. After all, no uniform samples were used there, but rather a condition on the samples from the proposal distribution. Let's examine this case again through the prism of this theorem.

We sample points uniformly inside the square $[-1, 1]^2$, so:

$$f(x, y) = \begin{cases} \frac{1}{4} & \text{if } -1 \leq x, y \leq 1 \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad g(x, y) = \begin{cases} \frac{1}{\pi} & \text{if } \sqrt{x^2 + y^2} \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

By equation, 3.3 $M_{opt} = \frac{4}{\pi}$ and hence:

$$\frac{g(x, y)}{M_{opt} f(x, y)} = \begin{cases} 1 & \text{if } \sqrt{x^2 + y^2} \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

It is clear, therefore, why, we do not need the uniform samples $\{Y_i\}_1^\infty$ of Theorem 3.17. Samples outside the disk are accepted with probability 0, that is never, and samples inside the disk are accepted with probability 1, which is always.

Despite rejection sampling's generality, it is not usually used, because, most often, there exist more efficient methods. Nevertheless, we will, later, use rejection sampling to sample a torus.

### Inverse Transform Sampling

We are, now, going to present a method that allows us to sample from any 1-dimensional distribution. This method works by taking a sample from a uniform distribution over $[0, 1]$ and transforming it using the inverse of the target cumulative distribution function. In practice, for this method to be applied we have to be able to find a closed-form expression expression for the inverse of the cdf, which in many cases is impossible. We present, below, a proof of correctness:

**Theorem 3.20: Inverse Transform Sampling**

Le $F$ be a cumulative distribution function and $U$ be a random variable distributed uniformly over $[0, 1]$, then the random variable $F^{-1}(U)$ is distributed according to $F$, where $F^{-1}(u) = \inf\{x \,|\, F(x) \leq u\}$.

*Proof.* We will show that the cumulative distribution function of $F^{-1}(U)$ is $F$

$$\mathbb{P}\left[F^{-1}(U) \leq x\right] = \mathbb{P}\left[U \leq F(x)\right] = F(x)$$

$\square$

*Remark.* Usually, the values 0 and 1 are excluded from the range of $U$. This is because $F^{-1}(0) = -\infty$ and for many distributions $F^{-1}(1) = \infty$. For continuous distributions, excluding two values is negligible, so that does not change the theorem.

**Example 3.21: Sampling from an exponential distribution**

Let's say we want to sample the exponential distribution $f(x) = e^{-x}$. The CDF is:

$$F(x) = \int_{-\infty}^{x} f(t)\,dt = \begin{cases} 1 - e^{-x} & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

And, hence, the inverse is:

$$F^{-1}(u) = -\log(1 - u)$$

Even though Theorem 3.20 applies to 1-dimensional distributions, we can use it to sample from multidimensional ones, as well. Let's assume we have a 2-dimensional pdf $f_{X,Y}$, then:

$$f_{X,Y} = f_{Y|X} f_X \tag{3.4}$$

Now, we can use inverse transform sampling to sample from $X$'s cdf $F_X$ and then, given the sampled $x$, we can sample $Y$ from the conditional cdf $F_{Y|X}$. The trick that was just described can be naturally extended to sample higher dimensional distributions. For example a 3-dimensional pdf can be factored as follows:

$$f_{X,Y,Z} = f_{Z|X,Y} f_{Y|X} f_X \tag{3.5}$$

and be sampled by sampling first $X$, then $Y$, then $Z$. The order with which we sample the variables can, of course, be switched to better fit our needs. For example, $f_{X|Y}$ might have a closed-form inverse while $f_{Y|X}$ does not.

Whenever applicable, inverse transform sampling is, usually, preferred over other methods, because it requires exactly as many uniform random samples as the variables we want to sample. Compare that with rejection sampling for which the number of random samples is not even deterministic. Yet, the need for a closed-form inverse prevents its usage for sampling more complex distributions.

### Markov Chain Sampling

The methods we have seen so far sample exactly from a specified distribution. Next, we are going to present a method that approximates a distribution. This method uses a Markov Chain, so first of all, we will begin by defining it.

**Definition 3.22: Markov Chain**

Let $S$ be a measurble space. A Markov chain is a discrete-time $S$-valued stochastic process $X_n$ that satisfies the following property:

$$\mathbb{P}\left[X_n = x_n \,|\, X_{n-1} = x_{n-1}, X_{n-2} = x_{n-2}, \ldots\right] = \mathbb{P}\left[X_n = x_i \,|\, X_{n-1} = x_{n-1}\right]$$

*Remark* (1). $S$ is referred to as the *state space.*

*Remark* (2). This property is called Markovian.

*Remark* (3). Informally, the Markovian property means that the next value in a sequence depends only on the present one.

The most common cases for the state space is for it to be:

- Finite
- Continuous

We will first take a look at finite state spaces, because they are simpler. Then we will extend our discussion to continuous state spaces which are naturally more pertinent to our subject.

Let's say that $|S| = N$. We can arrange the probabilities $p_{ij} = \mathbb{P}\left[X_n = x_j \,|\, X_{n-1} = x_i\right]$ in a $N \times N$ matrix $P$. This matrix is called the *transition matrix* for obvious reasons. It is also a stochastic matrix, specifically row-stochastic, because its elements are non-negative and its rows sum up to 1. We can also represent a distribution over the state space as a row vector. If $\mu_n$ is the probability vector at some time $n$, then it is easy to see that, by the law of total probability:

$$\mu_{n+1} = \mu_n P \tag{3.6}$$

This equation makes the name transition matrix even clearer. We can, also, find the how the distribution changes after $k$ steps by using the matrix $P^k$. If a distribution $\pi$ remains unchanged after being multiplied by $P$, then it is called a stationary distribution. We have, thus, the following definition:

**Definition 3.23: Stationary Distribution**

If $P$ is a transition matrix of a Markov chain, then a distribution $\pi$ is called stationary if:

$$\pi = \pi P$$

*Remark.* The stationary distribution is, also, called *steady state* distribution.

From the above definition it follows that the stationary distribution is an eigenvector of $P$ with eigenvalue 1. Under some mild assumptions, it can be proven that the stationary distribution is unique and that as we step along the chain, any initial distribution will converge to the stationary. More formally:

**Theorem 3.24**

Let $P$ be a $n \times n$ transition matrix for a Markov chain. If $P_{ij}^k > 0, \forall 1 \leq i, j \leq n$ for some $k\mathbb{N}$, then there exists a unique stationary distribution $\pi$. Furthermore:

$$\lim_{n \to \infty} \mu P^n = \pi, \text{ for all distributions } \mu$$

*Proof.* Refer to [29]. □

The above theorem is key to sampling distributions with Markov chains. There are ways of constructing a Markov chain, whose stationary distribution is the desired one. Then we can simulate the chain for enough steps in order to converge to the stationary distribution. The Metropolis-Hastings [57] algorithm is the most famous example of such a process.

We opted to present the main results on Markov chains for the finite case, because the tools for analyzing a continuous state space are more complicated and unintuitive. Having described them, however, we are now in position to undertake this labour. Since, the state space is not finite anymore we cannot arrange the transition probabilities in a matrix. The analogous concept is a transition kernel. We give the relevant definitions below:

**Definition 3.25: Transition Kernel**

Let $S$ be a state space, then $P$ is called a transition kernel if for every $x \in S$, $P(x, \cdot)$ is a probability measure and for every measurable set $A$, $P(\cdot, A)$ is a measurable function.

*Remark.* We use $P$ for both the transition matrix and the transition kernel, the meaning should be deducible form the context.

**Definition 3.26: Transition Kernel Density**

Let $S$ be a state space and $P$ be a transition kernel, then $p$ is called a transition kernel density if for every $x \in S$ and every measurable set $A \subset S$ it satisfies the following:

$$P(x, A) = \int_A p(x, y)\, dy$$

Finding $\mu_{n+1}$ by applying the kernel to a distribution $\mu$ can, then, be done as follows:

$$\begin{gathered} \mu_{n+1}(A) = \int_S P(x, A)\, d\mu_n(x) \\ \text{or} \\ \mu_{n+1}(y) = \int_S p(x, y)\mu_n(x)\, dx \end{gathered} \tag{3.7}$$

The first is also written $\mu_{n+1} = \mu_n P$. A stationary distribution $\pi$ in this case satisfies the following property:

$$\pi = \pi P \quad \text{or} \quad \pi(y) = \int_S p(x, y)\pi(x)\, dx \tag{3.8}$$

Finally, we have the following theorem, which corresponds to theorem 3.24:

**Theorem 3.27**

Let $S$ be a state space and $P$ be a transition kernel. If there exist a distribution $\mu$, $k \in \mathbb{N}$ and $\varepsilon \leq \mu$ for which $P^k(x, A) \geq \varepsilon\mu$, for every $x \in S$, then there exists a unique stationary distribution $\pi$. Furthermore:

$$\lim_{n\to\infty} \|\phi P^n - \pi\| = 0 \text{ for all distributions } \phi$$

where $\|\cdot\|$ denotes total variation.

*Proof.* Refer to [23] □

### 3.2.2 Sampling Primitive Shapes

The term *primitive* is widely used in graphics to refer to shapes that are used either as starting points or basic components for something more complex. We will, now, discuss methods for sampling such shapes. In particular, we will show how spheres, balls, disks (which is a 2-dimensional ball), tori, triangles and meshes can be sampled uniformly. We note that we include meshes, despite them not being characterized as primitives typically. Also, the reason why we are interested in uniformity will become clearer in 5, for now, we take it as a requirement.

Before starting, though, we will present some tools that will be needed. The shapes we want to sample can be represented parametrically, that is as the image of a function from a parameter space to the space where the shape resides. For example the following are the parametric equations for the unit circle centered at the origin:

$$\begin{cases} x = \cos\varphi \\ y = \sin\varphi \end{cases} \qquad \varphi \in [0, 2\pi) \tag{3.9}$$

When sampling a parametric representation, we may have the following clash. The desired distribution is specified in the space of the function image, but the sampling takes place in the parameter space. We will show how these two relate to each other and how we can find the distribution in the parameter space given the distribution in the shape space. Let us assume that $\Phi : A \to B$ is the mapping from the parameters to the shape. We will examine three cases:

- $A \subset \mathbb{R}^2$ and $B = \mathbb{R}^2$
- $A \subset \mathbb{R}^3$ and $B = \mathbb{R}^3$
- $A \subset \mathbb{R}^2$ and $B = \mathbb{R}^3$

In each case, we are going to show what the differential generalized volume element is at a point $b = \Phi(a) \in B$, with $a \in A$. Having found that, we can transform a distribution in $B$ to a distribution in the parameter space via the following theorem:

**Theorem 3.28: Distribution in Parameter Space**

Let $\Phi : A \to B$ and $g(a)da$ be the differential generalized volume element at $b = \Phi(a)$. If $f(b)$ is a distribution in $B$, then $f(\Phi(a))g(a)$ is the distribution in the $A$.

*Proof.* Let $D \subset B$ be a measurable set, then:

$$\begin{aligned} \mathbb{P}[b \in D] &= \int_D f(b)\, db \\ &= \int_{\Phi^{-1}(D)} f(\Phi(a))g(a)da \end{aligned}$$

We also have:

$$\mathbb{P}\left[b \in D\right] = \mathbb{P}\left[a \in \Phi^{-1}(D)\right]$$

Hence, $f(\Phi(a))g(a)$ is the distribution in $A$. □

Let us now see what $g(a)$ is in each of the three cases. These results should be available in any introductory book on vector calculus, e.g. [44]. In the first two cases, $g(a)$ is the absolute of the jacobian determinant of $\Phi$. In the third case, $g(a)$ is the length of the cross product of the partial derivatives with respect to the parameters. More formally:

- $A \subset \mathbb{R}^2$ and $B = \mathbb{R}^2$

$$\begin{aligned} g(a) = g(u,v) &= \left|\det(J(\Phi(u,v)))\right| \\ &= \left|\begin{vmatrix} \frac{\partial \Phi_x}{\partial u} & \frac{\partial \Phi_x}{\partial v} \\ \frac{\partial \Phi_y}{\partial u} & \frac{\partial \Phi_y}{\partial v} \end{vmatrix}\right| \end{aligned} \tag{3.10}$$

- $A \subset \mathbb{R}^3$ and $B = \mathbb{R}^3$

$$\begin{aligned} g(a) = g(u,v,w) &= \left|\det(J(\Phi(u,v,w)))\right| \\ &= \left|\begin{vmatrix} \frac{\partial \Phi_x}{\partial u} & \frac{\partial \Phi_x}{\partial v} & \frac{\partial \Phi_x}{\partial w} \\ \frac{\partial \Phi_y}{\partial u} & \frac{\partial \Phi_y}{\partial v} & \frac{\partial \Phi_y}{\partial w} \\ \frac{\partial \Phi_z}{\partial u} & \frac{\partial \Phi_z}{\partial v} & \frac{\partial \Phi_z}{\partial w} \end{vmatrix}\right| \end{aligned} \tag{3.11}$$

- $A \subset \mathbb{R}^2$ and $B = \mathbb{R}^3$

$$\begin{aligned} g(a) = g(u,v) &= \left\| \frac{\partial \Phi}{\partial u} \times \frac{\partial \Phi}{\partial v} \right\| \\ &= \left\| \left( \frac{\partial \Phi_x}{\partial u}, \frac{\partial \Phi_y}{\partial u}, \frac{\partial \Phi_z}{\partial u} \right) \times \left( \frac{\partial \Phi_x}{\partial v}, \frac{\partial \Phi_y}{\partial v}, \frac{\partial \Phi_z}{\partial v} \right) \right\| \end{aligned} \tag{3.12}$$

At last, we are ready to proceed to the methods for sampling the aforementioned shapes.

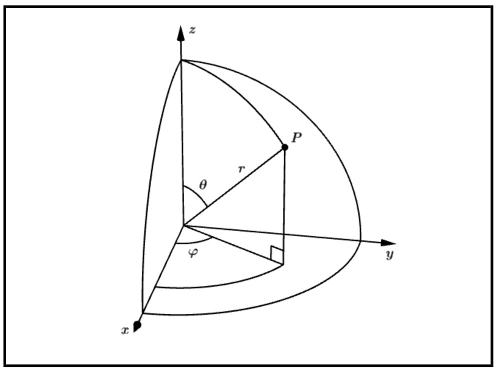


Figure 3.3: The relation between spherical and Cartesian coordinates

### (Hyper)Spheres and (Hyper)Balls

Firstly, we will discuss sampling spheres and balls. Let us begin with a 3-dimensional sphere. The unit 3-dimensional spheres centered at the origin can be expressed parametrically with spherical coordinates (see figure 3.3) by the following equations:

$$\begin{cases} x = \sin\theta\cos\varphi \\ y = \sin\theta\sin\varphi & \varphi \in [0, 2\pi)\,,\ \theta \in [0, \pi] \\ z = \cos\theta \end{cases} \tag{3.13}$$

So the mapping $\Phi$ is:

$$\Phi(\varphi, \theta) = (\sin\theta\cos\varphi, \sin\theta\sin\varphi, \cos\theta),\ (\varphi, \theta) \in [0, 2\pi) \times [0, \pi] \tag{3.14}$$

The partial derivatives of $\Phi$ are:

$$\begin{aligned} \frac{\partial\Phi}{\partial\varphi} &= (-\sin\theta\sin\varphi, \sin\theta\cos\varphi, 0) \\ \frac{\partial\Phi}{\partial\theta} &= (\cos\theta\cos\varphi, \cos\theta\sin\varphi, -\sin\theta) \end{aligned} \tag{3.15}$$

Hence, by equation 3.12, the differential area element is:

$$\begin{aligned} \left\|\frac{\partial\Phi}{\partial\varphi} \times \frac{\partial\Phi}{\partial\theta}\right\| &= \left\|(-\sin\theta\sin\varphi, \sin\theta\cos\varphi, 0) \times (\cos\theta\cos\varphi, \cos\theta\sin\varphi, -\sin\theta)\right\| \\ &= \left\|\begin{vmatrix} \hat{x} & \hat{y} & \hat{z} \\ -\sin\theta\sin\varphi & \sin\theta\cos\varphi & 0 \\ \cos\theta\cos\varphi & \cos\theta\sin\varphi & -\sin\theta \end{vmatrix}\right\| \\ &= \left\|\hat{x}\begin{vmatrix} \sin\theta\cos\varphi & 0 \\ \cos\theta\sin\varphi & -\sin\theta \end{vmatrix} - \hat{y}\begin{vmatrix} -\sin\theta\sin\varphi & 0 \\ \cos\theta\cos\varphi & -\sin\theta \end{vmatrix} + \hat{z}\begin{vmatrix} -\sin\theta\sin\varphi & \sin\theta\cos\varphi \\ \cos\theta\cos\varphi & \cos\theta\sin\varphi \end{vmatrix}\right\| \\ &= \left\|-\sin^2\theta\cos\varphi\hat{x} - \sin^2\theta\sin\varphi\hat{y} - (\sin\theta\cos\theta\sin^2\varphi + \sin\theta\cos\theta\cos^2\varphi)\hat{z}\right\| \\ &= \left\|-\sin^2\theta\cos\varphi\hat{x} - \sin^2\theta\sin\varphi\hat{y} - \sin\theta\cos\theta\hat{z}\right\| \\ &= \sqrt{(\sin^2\theta\cos\varphi)^2 + (\sin^2\theta\sin\varphi)^2 + (\sin\theta\cos\theta)^2} \\ &= \sqrt{\sin^4\theta\cos^2\varphi + \sin^4\theta\sin^2\varphi + \sin^2\theta\cos^2\theta} \\ &= \sqrt{\sin^4\theta + \sin^2\theta\cos^2\theta} \\ &= \sqrt{\sin^2\theta(\sin^2\theta + \cos^2\theta)} \\ &= \sqrt{\sin^2\theta} \\ &= \sin\theta \end{aligned} \tag{3.16}$$

The uniform distribution on the sphere is equal to the inverse of its area which is $4\pi$

and therefore, by theorem 3.28 the distribution in parameter space is:

$$f(\varphi,\theta) = \frac{\sin\theta}{4\pi} = \left(\frac{1}{2\pi}\right)\left(\frac{\sin\theta}{2}\right) \tag{3.17}$$

The above indicates that $\varphi, \theta$ are independent and can be sampled separately. In particular, $\varphi$ is uniform over $[0, 2\pi)$ and thus a simple scaling of a random variable over $[0, 1)$ suffices. For $\theta$, we can use inverse transform sampling (theorem 3.20). The cdf of $\theta$ is:

$$F_\theta(x) = \int_0^x \frac{\sin t}{2}\,dt = \frac{1-\cos x}{2} \tag{3.18}$$

And its inverse:

$$F_\theta^{-1}(x) = \arccos(1-2x) \tag{3.19}$$

We now move to a 2-dimensional ball, which is a disk. A unit disk centered at the origin is expressed parametrically with polar coordinates, which are shown in figure 3.4, as follows:

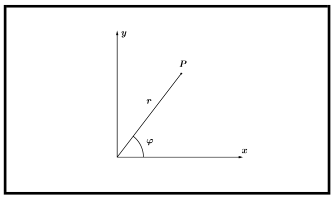


Figure 3.4: The relation between polar and Cartesian coordinates

$$\begin{cases} x = r\cos\varphi \\ y = r\sin\varphi,\ r \in [0,1] \end{cases} \quad \varphi \in [0, 2\pi) \tag{3.20}$$

or

$$\Phi(r,\varphi) = (r\cos\varphi, r\sin\varphi),\ (r,\varphi) \in [0,1]\times[0,2\pi)$$

For the differential area element we use equation 3.10:

$$\begin{aligned} |\det(J(\Phi(r,\varphi)))| &= \left|\begin{vmatrix} \frac{\partial\Phi_x}{\partial r} & \frac{\partial\Phi_x}{\partial\varphi} \\ \frac{\partial\Phi_y}{\partial r} & \frac{\partial\Phi_y}{\partial\varphi} \end{vmatrix}\right| \\ &= \left|\begin{vmatrix} \cos\varphi & -r\sin\varphi \\ \sin\varphi & r\cos\varphi \end{vmatrix}\right| \\ &= \left|r\cos^2\varphi + r\sin^2\varphi\right| \\ &= r \end{aligned} \tag{3.21}$$

Similarly to the sphere, the uniform distribution on a disk is equal to the inverse of its area, which $\pi$, and the distribution in the parameter space, again by theorem 3.28, is:

$$f(r,\varphi) = \frac{r}{\pi} = (2r)\left(\frac{1}{2\pi}\right) \tag{3.22}$$

Again, the two parameters are independent, $\varphi$ is uniform over $[0, 2\pi)$ and $r$ can be sampled with inverse transform sampling. The latter's cdf is:

$$F_r(x) = \int_0^x 2t\,dt = x^2 \tag{3.23}$$

And its inverse:

$$F_r^{-1}(x) = \sqrt{x} \tag{3.24}$$

Next, we have the 3-dimensional ball which can be parametrized with spherical coordinates (see figure 3.3 as follows:

$$\begin{cases} x = r\sin\theta\cos\varphi \\ y = r\sin\theta\sin\varphi \qquad r \in [0,1]\,,\ \varphi \in [0,2\pi)\,,\ \theta \in [0,\pi] \\ z = r\cos\theta \end{cases} \tag{3.25}$$

or

$$\Phi(r,\varphi,\theta) = (r\sin\theta\cos\varphi, r\sin\theta\sin\varphi, r\cos\theta),\ (r,\varphi,\theta) \in [0,1]\times[0,2\pi)\times[0,\pi]$$

For the differential volume element we use equation 3.11:

$$\begin{aligned}
|\det(J(\Phi(u,v,w)))| &= \left|\begin{vmatrix} \frac{\partial\Phi_x}{\partial r} & \frac{\partial\Phi_x}{\partial\varphi} & \frac{\partial\Phi_x}{\partial\theta} \\ \frac{\partial\Phi_y}{\partial r} & \frac{\partial\Phi_y}{\partial\varphi} & \frac{\partial\Phi_y}{\partial\theta} \\ \frac{\partial\Phi_z}{\partial r} & \frac{\partial\Phi_z}{\partial\varphi} & \frac{\partial\Phi_z}{\partial\theta} \end{vmatrix}\right| \\
&= \left|\begin{vmatrix} \sin\theta\cos\varphi & -r\sin\theta\sin\varphi & r\cos\theta\cos\varphi \\ \sin\theta\sin\varphi & r\sin\theta\cos\varphi & r\cos\theta\sin\varphi \\ \cos\theta & 0 & -r\sin\theta \end{vmatrix}\right| \\
&= \left|\cos\theta \begin{vmatrix} -r\sin\theta\sin\varphi & r\cos\theta\cos\varphi \\ r\sin\theta\cos\varphi & r\cos\theta\sin\varphi \end{vmatrix} - r\sin\theta \begin{vmatrix} \sin\theta\cos\varphi & -r\sin\theta\sin\varphi \\ \sin\theta\sin\varphi & r\sin\theta\cos\varphi \end{vmatrix}\right| \\
&= \Big|-\cos\theta(r^2\sin\theta\cos\theta\sin^2\varphi + r^2\sin\theta\cos\theta\cos^2\varphi) \\
&\qquad -r\sin\theta(r\sin^2\theta\cos^2\varphi + r\sin^2\theta\sin^2\varphi)\Big| \\
&= \left|-r^2\sin\theta\cos^2\theta - r^2\sin^3\theta\right| \\
&= \left|-r^2\sin\theta(\cos^2\theta + \sin^2\theta)\right| \\
&= r^2\sin\theta
\end{aligned} \tag{3.26}$$

The uniform distribution over a 3-dimensional ball is equal to the inverse of its volume, which is $\frac{4\pi}{3}$, and therefore, the distribution in the parameter space is:

$$f(r,\varphi,\theta) = \frac{3r^2\sin\theta}{4\pi} = (3r^2)\left(\frac{1}{2\pi}\right)\left(\frac{\sin\theta}{2}\right) \tag{3.27}$$

The parameters are, once again, independent, with $\varphi$, again, being uniform over $[0, 2\pi)$ and inverse transform sampling being applicable for the others. Specifically, their cdfs are:

$$\begin{aligned} F_r(x) &= \int_0^x 3r^2\,dt = x^3 \\ F_\theta(x) &= \int_0^x 3r^2\,dt = \frac{1-\cos x}{2} \end{aligned} \tag{3.28}$$

And their inverses:

$$\begin{aligned} F_r^{-1}(x) &= \sqrt[3]{x} \\ F_\theta^{-1}(x) &= \arccos(1-2x) \end{aligned} \tag{3.29}$$

Besides the methods developed up to this point, there are others that do not use parametric equations. We are going to describe now two such techniques that are very elegant, one for sampling uniformly an $n$-dimensional sphere (note that this is a set of $\mathbb{R}^{n+1}$) and one for sampling an $n$-dimensional ball.

The first method is very intuitive. We first sample from a vector isotropic $n+1$-dimensional normal distribution and then normalize it. The correctness of the method is due to symmetry. The following theorem what we just described:

**Theorem 3.29**

Let $X \sim \mathcal{N}(0, \sigma I_{n+1})$, then $\frac{X}{\|X\|}$ is distributed uniformly on the unit $n$-dimensional sphere centered at the origin.

The method for the $n$-dimensional ball is less intuitive. We can sample an $n+2$-dimensional isotropic normal distribution and simply drop the last two coordinates:

**Theorem 3.30**

Let $X = (x_1, x_2, \ldots, x_{n+1}, x_{n+2})$ be a random variable distributed uniformly on the $n+1$-dimensional unit sphere centered at the origin, then $(x_1, x_2, \ldots, x_n)$ is distributed uniformly on the unit $n$-dimensional ball centered at the origin.

*Proof.* Refer to [74]. □

Later, in section 5.2, we will use the method above to sample 2-dimensional disks in $\mathbb{R}^3$, which are perpendicular to some given vector. We note that in the theorem above, instead of dropping the last two coordinates, we can remove any other two independent components. Hence, we drop the last coordinate and then remove the component along the vector to which we want the disk to be perpendicular. Another way to do that would be to sample a 2-dimensional disk in $\mathbb{R}^2$, using the method discussed earlier in this section and then use a $3 \times 2$ matrix to transform it to $\mathbb{R}^3$. In theory, this could be faster, because samples from a uniform distribution are cheaper than those of a normal one. However, in

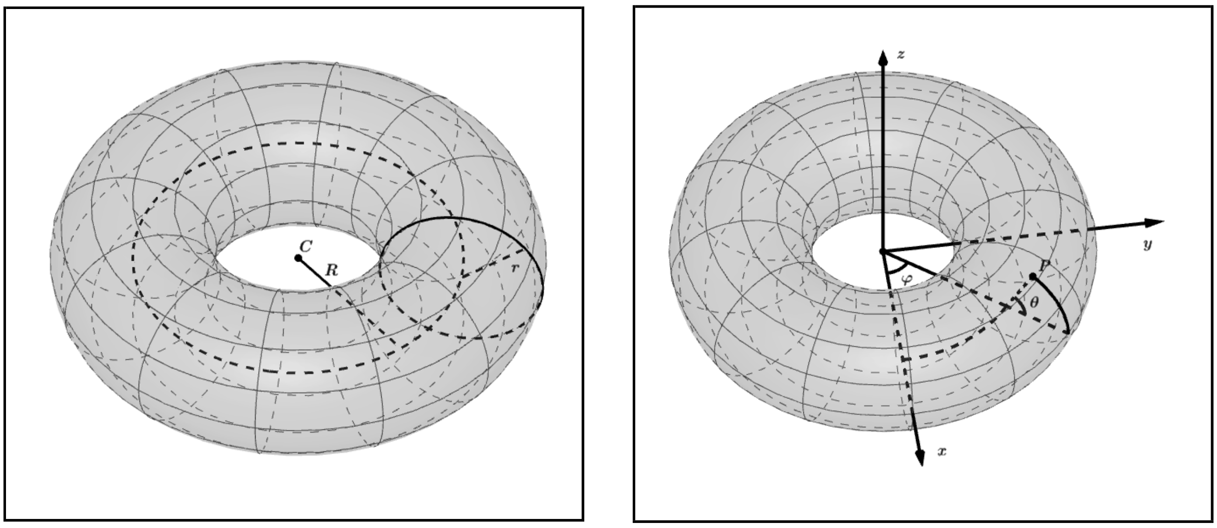


Figure 3.5: On the left, a torus with its center $C$ and major and minor radii, $R$ and $r$, respectively. On the right, the physical meaning of the parameters $\varphi, \theta$ of equation 3.30

a pure PyTorch [51] implementation, we found the method with the dropped coordinates to be faster, instead.

For a comprehensive collection of methods for sampling $n$-dimensional spheres and balls we refer the reader to [56].

### Tori

A torus can be described by its major and minor radii. Its major radius $R$ is the distance from the center of the tube to the center of the torus, while the minor radius $r$ is the radius of the tube. These quantities are depicted in figure 3.5. We will now describe a method for sampling a torus. We will assume that the circle of radius $R$ at the center of the tube lies on the $xy$ plane and that the torus is centered at the origin. The case where the circle lies on the $yz$ or the $xz$ planes is similar. We will utilize a parametric representation here as well, which is given below:

$$\begin{cases} x = (R + r\cos\theta)\cos\varphi \\ y = (R + r\cos\theta)\sin\varphi \qquad \varphi, \theta \in [0, 2\pi) \\ z = r\sin\theta \end{cases} \tag{3.30}$$

or

$$\Phi(\varphi, \theta) = ((R + r\cos\theta)\cos\varphi, (R + r\cos\theta)\sin\varphi, r\sin\theta),\ (\varphi, \theta) \in [0, 2\pi)^2$$

The partial derivatives of $\Phi$ are:

$$\begin{aligned} \frac{\partial\Phi}{\partial\varphi} &= (-(R + r\cos\theta)\sin\varphi, (R + r\cos\theta)\cos\varphi, 0) \\ \frac{\partial\Phi}{\partial\theta} &= (-r\sin\theta\cos\varphi, -r\sin\theta\sin\varphi, r\cos\theta) \end{aligned} \tag{3.31}$$

Ergo, by equation 3.12, the differential area element is:

$$\begin{aligned}
\left\|\frac{\partial\Phi}{\partial\varphi}\times\frac{\partial\Phi}{\partial\theta}\right\| &= \left\|(-(R+r\cos\theta)\sin\varphi, (R+r\cos\theta)\cos\varphi, 0)\times(-r\sin\theta\cos\varphi, -r\sin\theta\sin\varphi, r\cos\theta)\right\| \\
&= \left\|\begin{vmatrix} \hat{x} & \hat{y} & \hat{z} \\ -(R+r\cos\theta)\sin\varphi & (R+r\cos\theta)\cos\varphi & 0 \\ -r\sin\theta\cos\varphi & -r\sin\theta\sin\varphi & r\cos\theta \end{vmatrix}\right\| \\
&= \left\|\hat{x}\begin{vmatrix} (R+r\cos\theta)\cos\varphi & 0 \\ -r\sin\theta\sin\varphi & r\cos\theta \end{vmatrix} - \hat{y}\begin{vmatrix} -(R+r\cos\theta)\sin\varphi & 0 \\ -r\sin\theta\cos\varphi & r\cos\theta \end{vmatrix}\right. \\
&\qquad \left. +\hat{z}\begin{vmatrix} -(R+r\cos\theta)\sin\varphi & (R+r\cos\theta)\cos\varphi \\ -r\sin\theta\cos\varphi & -r\sin\theta\sin\varphi \end{vmatrix}\right\| \\
&= \Big\|(R+r\cos\theta)r\cos\theta\cos\varphi\hat{x} + (R+r\cos\theta)r\cos\theta\sin\varphi\hat{y} \\
&\qquad + \left((R+r\cos\theta)r\sin\theta\sin^2\varphi + (R+r\cos\theta)r\sin\theta\cos^2\varphi\right)\hat{z}\Big\| \\
&= \left\|(R+r\cos\theta)r\cos\theta\cos\varphi\hat{x} + (R+r\cos\theta)r\cos\theta\sin\varphi\hat{y} + (R+r\cos\theta)r\sin\theta\hat{z}\right\| \\
&= \sqrt{(R+r\cos\theta)^2r^2\cos^2\theta\cos^2\varphi + (R+r\cos\theta)^2r^2\cos^2\theta\sin^2\varphi + (R+r\cos\theta)^2r^2\sin^2\theta} \\
&= \sqrt{(R+r\cos\theta)^2r^2\cos^2\theta + (R+r\cos\theta)^2r^2\sin^2\theta} \\
&= \sqrt{(R+r\cos\theta)^2r^2} \\
&= r(R+r\cos\theta)
\end{aligned} \tag{3.32}$$

The area of the torus is $4\pi^2rR$, and so as we did for the cases above, the distribution in the parameter space is:

$$f(\varphi,\theta) = \frac{r(R+r\cos\theta)}{4\pi^2rR} = \frac{R+r\cos\theta}{4\pi^2R} = \left(\frac{1}{2\pi}\right)\left(\frac{R+r\cos\theta}{2\pi R}\right) \tag{3.33}$$

The parameters are independent and $\varphi$ is uniform over $[0,2\pi)$. In contrast to what we have seen so far, however, $\theta$ cannot be sampled with inverse transform sampling, because its cdf has no closed-form inverse. Nevertheless, we can use rejections sampling (see theorem 3.17), with a uniform proposal distribution. The optimal $M$ is given by equation 3.3:

$$M_{opt} = \sup\frac{\frac{R+r\cos\theta}{2\pi R}}{\frac{1}{2\pi}} = \frac{R+r\cos\theta}{R} = \frac{R+r}{R} \tag{3.34}$$

### Triangles and Meshes

The final shapes we are going to examine are triangles and triangular meshes. For triangles we follow PBRT [52]. Let $A$, $B$ and $C$ be the vertices of a triangle. A point $p$ on the triangle can be given as a combination of the vectors $\vec{AB}$ and $\vec{AC}$. More specifically:

$$p = \Phi(u, v) = u\vec{AB} + v\vec{AB},\ u \in [0, 1]\,,\ v \in [0, 1 - u] \tag{3.35}$$

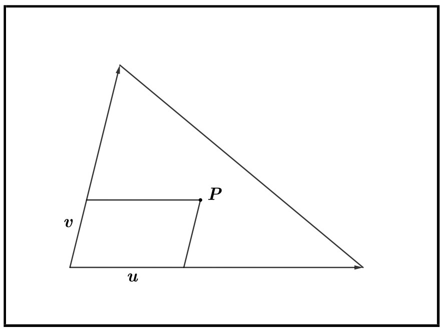


Figure 3.6: How the $uv$ coordinates map to the triangle

The situation is depicted in figure 3.6. The partial derivatives of $\Phi$ are:

$$\begin{aligned} \frac{\partial \Phi}{\partial u} &= \vec{AB} \\ \frac{\partial \Phi}{\partial v} &= \vec{AC} \end{aligned} \tag{3.36}$$

By equation 3.12 the differential area element is equal to the norm of the cross product of the partial derivative, which in this case are the vectors along the sides of the triangles, and so, is equal to twice the area of the triangle $\mathcal{A}$:

$$\left\| \frac{\partial \Phi}{\partial u} \times \frac{\partial \Phi}{\partial v} \right\| = \left\| \vec{AB} \times \vec{AC} \right\| = 2\mathcal{A} \tag{3.37}$$

We, thus, have the distribution in the parameter space:

$$f(u, v) = 2 \tag{3.38}$$

Even though this is a constant, the variables are not independent because the support of the distribution is not a rectangle. Using conditionality, we can write the distribution as follows:

$$f(u, v) = f_{V|U}(v; u) f_U(u) \tag{3.39}$$

We, then, have:

$$\begin{aligned} f_U(u) &= \int_0^{1-u} f(u, v)\, dv = \int_0^{1-u} 2\, dv = 2(1 - u) \\ f_{V|U}(v; u) &= \frac{f(u, v)}{f_U(u)} = \frac{1}{1 - u} \end{aligned} \tag{3.40}$$

We can sample both $u$ and $v$, in that order, using inverse transform sampling. Their cdfs are:

$$\begin{aligned} F_U(x) &= \int_0^x f_U(u))\, du = \int_0^x 2(1 - u)\, dv = 2u - u^2 \\ F_{V|U}(x; u) &= \int_0^x f_{V|U}(v; u)\, dv = \int_0^x \frac{1}{1 - u}\, dv = \frac{x}{1 - u} \end{aligned} \tag{3.41}$$

And their inverses:

$$\begin{aligned} F_U^{-1}(x) &= 1 - \sqrt{1 - x} \\ F_{V|U}^{-1}(x; u) &= x(1 - u) \end{aligned} \tag{3.42}$$

In order to sample a triangular mesh, we can simply choose one of each triangles with probability proportional to its area and, then, use the method above to sample a point inside that triangle.

## 3.3 Machine Learning

In this section we will give a brief overview of machine learning, with a focus on neural networks. The term machine learning encompasses a wide range of algorithms and related methods and theory that aim to solve a problem by "teaching" a computer how to do it. In contrast to other techniques, where expert knowledge is employed to craft specialized algorithms that target a problem, machine learning follows a data-driven approach. This means that data are used to change the behaviour of the algorithm to better adapt to the target task. This process of changing the algorithm's behaviour is called *training*.

**Example 3.31**

Let's say that at fish farming facility two species of fish are being bred. When fish are collected, we want to have a system automatically determine to which species a fish belongs to based on measurements given by optical sensors and weight sensors. One way to do that would be to bring an ichthyologist or marine biologist in order to develop a rule based system. Another route would be to collect this measurements along with the species that they correspond to and feed them to a machine learning algorithm.

Matchine learning algorthms usually comprise the following basic components:

- **Model** The model is what transforms the input of the problem to the output. It is, typically, expressed as a parametric function $f_\theta(x)$, where $x$ is the input and $\theta$ are the parameters. The parameters can range from something as simple as the weights of a matrix to something as complex as the syntax tree for an algebraic expression. It is through the alteration of $\theta$ that the algorithm is able to learn.

- **Optimization Procedure** This is the manner by which the parameters of the model adapt to the problem at hand. The choice of optimization procedure is highly influenced by the model. Some methods set requirements for the model. For example, gradient-descent requires the model to be differentiable with respect to the parameters. Other methods are developed in tandem with the model and suited to it. This is the case for support vector machines. Also, memory of time constraint affect the choice. A Newton-Raphson method might be very memory heavy for models that have many parameters due to it computing the Hessian matrix. Most optimization procedures work by minimizing some cost function that models the objective of the problem. The cost function itself might comprise subobjectives which need to be weighted.

- **Dataset** At the heart of a data-driven approach are, of course, the data. In fact, they may be considered its most important component since they are in some sense part of the problem. A dataset is precisely what the word's etymology implies, a set of data. What those data are exactly depends on many things. Some datasets are created with a specific task in mind, while others include as many pertinent information as possible so that they can be purposed for a variety of problems. In either case, most datasets are created by collecting data from the real world using sensors extended with extra measurements on those data or other metadata. Accordingly, a dataset might include images and/or sounds along with titles, creators, etc. The available datasets for a specific problem constrain the choice of model and, consequently, of optimization procedure. A small dataset is not suited for complex models and vice versa. Finally, the quality of a dataset, by which we mean its coverage of the data relevant to the problem, the level of noise, etc. , is a primary deciding factor for the efficacy of any algorithm that uses it. For example, we should not expect an algorithm that uses a dataset which includes images of dogs of only one breed to be very successful in generating realistic images of dogs of other breeds.

Machine learning algorithms are typically categorized based on the way the dataset is structured and how it is used by the optimization procedure. These catagories are:

### Supervised Learning

In supervised learning the dataset is required to comprise pairs. These pairs are samples, perhaps affected by noise, from a mapping and the aim is to learn this mapping. Let's assume that the dataset is composed of tuples $(x, y)$, $x \in X$, $y \in Y$, where $X$ is some input space and $Y$ is some output space. If $Y$ is discrete we refer to the task as classification, in which case $y$ is called label and the dataset is said to be annotated, otherwise we refer to it as regression, as in statistics. A point in the input spaces is commonly called *features* or *feature vector*, because in many tasks it is a collection of physical features of an object. In example 3.31, the solution we describe would be to use a supervised learning algorithm and the task itself is a classification task, since the output space comprise the two species of fish. Also, the features in this case would be the dimensions and weight given by the sensors. An example of a regression task would be to model power generation of solar panels given various environmental factors. The input space $X$, might be more complex then merely a multidimensional vector space. It might be a space of sequences of variable length. Text is an example of such a space. In these cases, the sequences are either embedded in a euclidean space, in order to use models fit for those, or processed by models that are able to handle sequences directly like Recurrent

Neural Networks (RNN) [59] and Long-Short Term Memory (LSTM) [36]. This last note also is not specific to supervised learning.

#### Unsupervised Learning

Unsupervised learning is usually used for tasks other than classification and regression and it is not as easy to define it directly, because it encompasses a larger variety of tasks. One classic example of unsupervised learning is the K-means algorithm. In general, clustering is an unsupervised learning problem. The datasets for unsupervised learning do not need to be annotated. The purpose is for the algorithm to "discover" patterns and relations in the data unprompted. Besides clustering, other unsupervised learning tasks are dimensionality reduction, image and text generation, language modeling, etc.

#### Reinforcement Learning

Reinforcement Learning differs greatly from the aforementioned categories, both in philosophy and in practice. The tasks that reinforcement learning is concerned with are those where an *agent* takes *actions* in an *environment*. The aim is to find a way of choosing actions based on the agent's *state*. This is called a *policy*. Accordingly, the datasets used for these tasks differ as well. In earnest, in a reinforcement learning context the term dataset is rarely used. However, we could say that the dataset is the environment and the allowable actions and states, instead of samples from some space. Learning proceeds by allowing the agent to act freely in the environment and getting "rewarded" or "punished" depending on its behaviour. This evaluation, then, guides the change in policy. Reinforcement learning is naturally applicable to games, that is teaching a computer to play a game, robotics, where the agent is a robot that needs to learn how to perform a specific task, etc.

The separation that we presented thus far is not absolute in practice. Hybrid approaches exist. For example, in a classification task only part of the data might be annotated and hence techniques from supervised and unsupervised learning are combined. Also, an algorithm in reinforcement learning might be enhanced by labeled data.

Having set the basis of machine learning, we will now discuss neural networks.

### 3.3.1 Neural Networks

Neural Networks, or Artificial Neural Networks, as they are often called to distinguish them from their biological counterparts are parametric models that that arose in the 1940s, but became prominent mainly after 2010, when computers became able to handle them efficiently. The are named, thusly, because they are, indeed, inspired by the biology of the brain. In analogy to biological neural networks, artificial ones comprise neurons which connect to each other. An artificial neuron takes a number of input values, multiplies each with a weight and adds the results together along with a constant, that is called the *bias*. This sum is then passed through a (non-linear) function giving the neuron's output (see figure 3.7). The non-linear function is commonly refered to as the *activation function*. The manner with which neurons are connected to form a network is called the network's *architecture*. The weights and biases of the neurons composing a neural network are the parameters of the model. For a comprehensive survey of the history of neural networks refer to Schmidhuber's report [61].

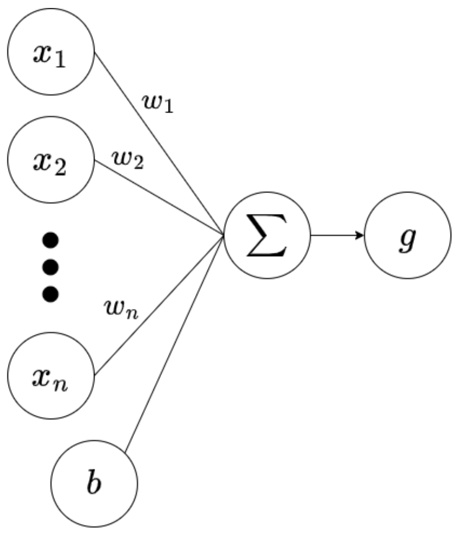


Figure 3.7: A schematic depiction of an artificial neuron. $x_i, w_i, 1 \leq i \leq n$ are the inputs and the weights, respectively, $b$ is the bias and $\sigma$ is the activation function

### The Perceptron Algorithm

The first modern neural network is considered to be Rosenblatt's perceptron. A perceptron is a single neuron whose activation function is a Heavyside step function $H$ (which equals 0 for input $< 0$ and 1 for inut $\geq 0$), that is used as a binary classifier. The term perceptron, however, is used to refer to the algorithm for training this particular model as well (see algorithm 3.2). If we consider the inputs and weights as vectors $x$ and $w$, respectively, then the function of the perceptron can be expressed as follows:

$$H(w \cdot x + b) \tag{3.43}$$

The perceptron is, hence, a linear classifier because is separates the space in two regions with a hyperplane. If the given dataset is linearly separable, that is, if the points belonging to the two classes can be separated by a hyperplane then, the perceptron is able to separate it perfectly and the algorithm converges in a finite number of steps. If the dataset is not linearly separated, however, no guarantee is given about some error metric. Below, we give the perceptron training algorithm in pseudocode. It is based on Hebbian learning and works by adding or subtracting to the weight vector the inputs that were misclassified.

**Algorithm 3.2:** Perceptron Algorithm

```
Data: $D$ : Dataset
      $w$ : Weights
      $b$ : Bias
foreach $(x_i, y_i) \in D$ do
    $\hat{y}_i \leftarrow H(w \cdot x + b)$
    $w \leftarrow w + (y_i - \hat{y}_i)x$
    $b \leftarrow b + (y_i - \hat{y}_i)$
end
```

### The XOR Problem

As we mentioned above, a single neuron acts as a linear classifier. However more complex problems cannot be solved by linear models. Imagine four points at the corners of a square with opposite corners belonging to the same class. The situation is shown in figure 3.8 where the two points of one class are depicted as disks and the two points of the other as crosses. It can be intuitively seen that in this case the dataset is not linearly separable. Nevertheless, a multilayer perceptron which we are going to examine next is apt for the task.

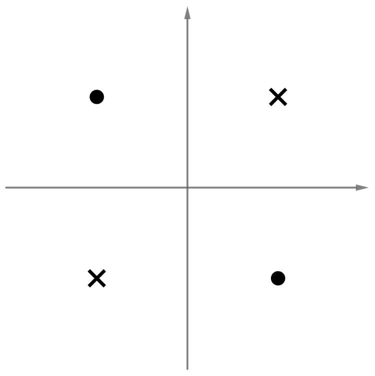


Figure 3.8: A dataset demonstrating the XOR problem

### Multilayer Perceptrons

In order to create more complex models we are going to have to use more neurons. Let us assume that we feed the same inputs to multiple neurons (all with the same activation function). We have, thusly, created a layer. The number of neurons used is called the layer's *width*. If we arrange the weights of the neurons in a matrix $W$ and the biases in a vector $b$ we can see that a layer is an affine transformation composed with the activation function, which we assume that is applied element-wise, i.e.:

$$\sigma(Wx + b) \tag{3.44}$$

A single layer, however, is nothing more than multiple independent linear models. The true power neural networks becomes apparent when we take the outputs of one layer and feed them to another. By stacking multiple layers we have created a multilayer perceptron (MLP), which is shown in figure 3.9. Then number of layers is called the *depth* of the

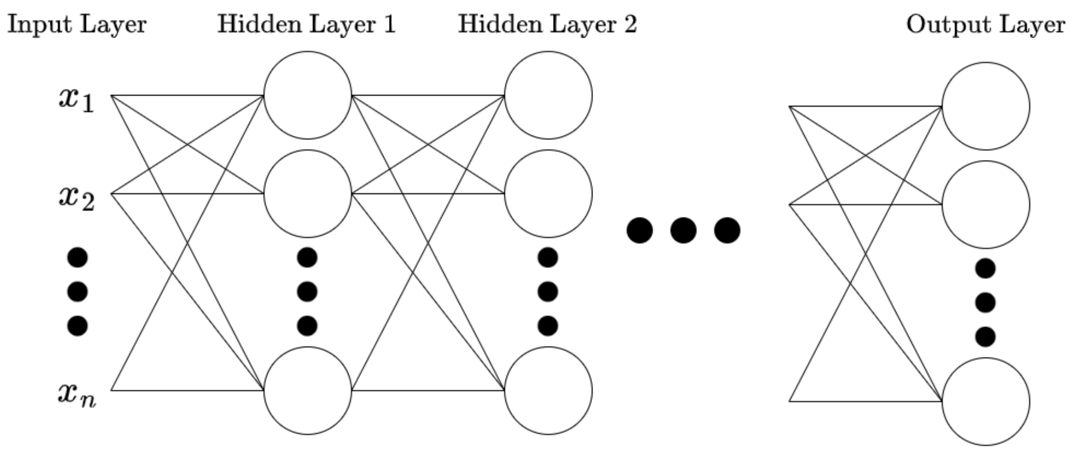
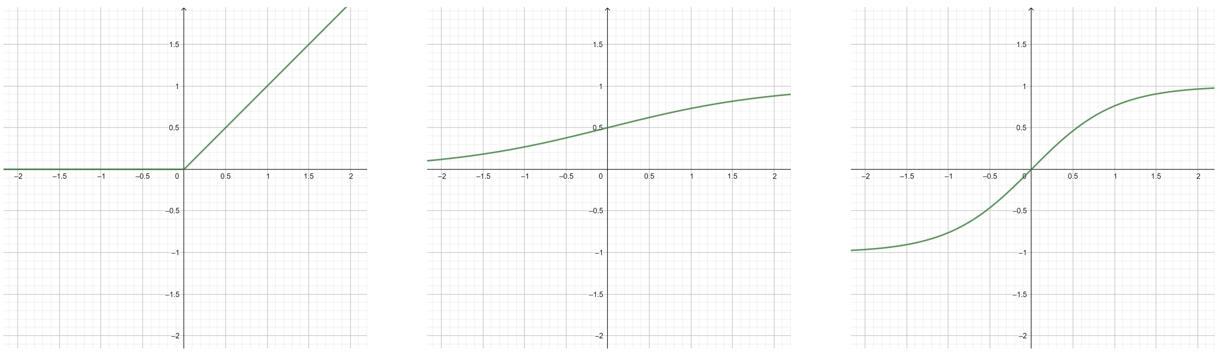


Figure 3.9: Graphical representation of a Multilayer Perceptron. Neurons are represented as circles

network. We usually refer to the inputs to the network as the *input layer*, even though these are not neurons, the last layer as the *output layer* and the ones between as *hidden layers*. Theoretically, the activation function of each layer might be different, but in practice the same activation function is used, except, perhaps, for the output layer, which might not have a different one or none at all (which we can think as having the identity function as activation). Usual choices for activation functions are the rectified linear unit (ReLU), the logistic function, which is synonymous to sigmoid in this context, and the hyperbolic tangent. The latter two differ only by a scale and translation. The graphs of these functions are shown in figure 3.10. Initially, the sigmoid was the preferred activation function. However, as the networks got deeper the problem of vanishing gradients, which we will explain later in the section, became prominent. ReLUs alleviate this issue and have become omnipresent in neural networks. Sigmoids are still oftentimes used as the activation function of the output layer, especially for classification tasks because they are bounded.



(a) ReLU (b) Sigmoid (c) Tanh

Figure 3.10: Plots of common activation function

Based on the above discussion, we now give a more formal definition of the MLP function inductively:

**Definition 3.32: MLP**

Let $W_1$ be a $n_0 \times n_1$ matrix, $b_1$ an $n_1$-dimensional vector and $\sigma_1 : \mathbb{R} \to \mathbb{R}$, then the following is an MLP with depth 1, $n_0$ inputs and $n_1$ outputs:

$$MLP_1(x) = \sigma_1(W_1 x + b_1)$$

If $MLP_{d-1}$ is an MLP with depth $d-1$, $n_0$ inputs and $n_{d-1}$ outputs, $W_d$ is a $n_{d-1} \times n_d$ matrix, $b_d$ an $n_d$-dimensional vector and $\sigma_d : \mathbb{R} \to \mathbb{R}$, then the following is an MLP with depth $d$, $n_0$ inputs and $n_d$ outputs:

$$MLP_d(x) = \sigma_d(W_d MLP_{d-1}(x) + b_d)$$

*Remark.* It would be more correct to parametrize the MLP with the matrices and the biases of each layer as well:

$$MLP_{d,\{W_i\},\{b_i\}}$$

We choose to omit them for the sake of simplicity.

### Other types of Layers and Networks

The type of layer we have described thus far is oftentimes called a *fully-connected* or *dense* layer, in order to differentiate it from other arrangements of neurons. Going back when we decided to feed all the inputs to all the neurons, we could have chosen to do something else instead.

For example we could have each neuron get only part of the input. A convolutional layer, does essentially that with the addition that the neurons share the weights amongst them. If we imagine that the inputs are arranged in a rectangular grid, then the neurons are arranged in a grid as well and get as input only a region around their position. This grid arrangement is very natural for images, for example, where convolutional neural networks (CNNs) excel.

Another alteration we could apply to our initial design is to feed the outputs of a layer as inputs to itself. This way we get recurrent neural networks (RNNs). RNNs are able to handle sequences of arbitrary length and are, accordingly, used in natural language processing. If no recurrent connections exist, the network is called *feedforwrd.* MLPs are, hence, synonymous to *feedforward fully connected networks.*

### Training

Many more architectures have been conceived in the field. Some of them general, while others specifically crafted to address a particular task. No matter the architecture used, however, the training of a neural networks almost always uses an algorithm based on gradient descent. This is the *optimization procedure* we mentioned at the beginning of the section. In order to apply gradient descent to a model we need two things:

- **Objective Function** Gradient descent, as an optimization procedure, works by minimizing a scalar *functional* that measures the performance of the model. This functional is also called *loss* or *cost* function, in the context of neural networks. We use the term functional, even though it not that common in the bibliography, to emphasize that, mathematically, it is a function that takes as input another function, the one expressed by the network in this case. Nevertheless, this functional depends on the network's parameters whose number is finite, which is why the term functional is uncommon. The exact form of the loss function depends on the problem. For example, in a regression problem, it can be the mean squared error (MSE):

$$MSE(\theta) = \frac{1}{|D|} \sum_{(x_i,y_i)\in D} (f_\theta(x_i) - y_i)^2 \tag{3.45}$$

where $\theta$ is a vector with the parameters of the network (*parameter vector*), $f_\theta$ is the network function, and $D$ is the dataset. The loss function can get quite complex by including various subobjectives.

- **Differentiability** Gradient descent requires that the objective function is differentiable (almost everywhere) with respect to the parameters. Consequently, the network function needs to be differentiable as well, which leads to the differentiability of the activation functions of the neurons. Typically this is not an issue for the activation functions, but for the loss function, sometimes it is a consideration that must be taken into account for the design.

The minimization of the loss function proceeds in an iterative fashion. Let us assume that $L$ is the loss function and $\theta_i$ is the parameter vector at the $i$-th iteration, then:

$$\theta_{i+1} = \theta_i - r\nabla L(\theta_i) \tag{3.46}$$

where $r$ is a positive real number that regulates the "speed" of the "descent", called, accordingly, learning rate. The gradient with respect to the parameters for feedforward networks can be computed with an algorithm called backpropagation [58]. This algorithms works by firstly computing the gradient at the last layer and proceeding to the previous ones using the chain rule [44]. When sigmoid activations are used, a deep architecture can exhibit a problem called *vanishing gradients*. Due to the sigmoid's derivative taking small values, the gradients at the first layers, where these derivatives get accumulated in products, get very small, and further, due to the limited precision of float point numbers in computers, are zeroed. This problem can be addressed by using ReLUs instead, as we mentioned above.

Modern programming frameworks, like PyTorch [51] and Tensorflow [46], allow the user to build arbitrary networks by simply specifying how the input and the layers are connected to each other. The gradient can then be computed based on the operations that took place without any other "help" from the user. This process is called *automatic differentiation*. It works by constructing a graph of the operations and traversing it backwards similarly to backpropagation.

### Metrics

Even though the loss function measures the performance of the network and guides the training process, usually, other functions are used to assess the final models. Such functions are called *metrics*. The reason for this discrepancy is that computing the derivative of these metrics is either not possible or intractable, and, consequently cannot be used as loss functions. Since minimizing (or maximizing) those metrics is the ultimate goal, loss functions are designed to be in as close correspondence to them as possible.

#### Hyperparameters

We note here that the widths of layers, the depth of the network, aspects of the architecture in general, as well as, numerical or other choices that concern the optimization algorithm and the dataset, like the specific algorithms used, the learning rate, etc., are called hyperparameters. This is so, because, they are not optimized during training, like the standard parameters. In order to determine favorable hyperparameters, one has to train multiple networks with different configurations. A dataset, called *validation dataset*, separate to the training dataset is used to evaluate the performance of these configurations.

#### Expressive Power

The biological inspiration of neural networks does not, of course, provide any direct evidence of their capabilities. What kind of functions can a neural network express? This is a very important question. Imagine that training a neural network fails, meaning its performance is subpar. What is the cause of this? Is it the quality of the dataset? Or does the network fail due to an intrinsic inability to learn the task? One of the first results in this area is due to Cybenko [20], who proved that an MLP with one hidden layer, whose activation is sigmoid, and no activation in the output layer can approximate any function on a compact set with arbitrary precision, given enough neurons in the hidden layer. Since then, many other results of that type have been proven for various architectures.

We should be wary, though, that these results do not explain fully the success of neural networks. They do not assure convergence of training for example, nor do they provide values for the hyperparameters. Besides, similar results are available for other models that have been surpassed by neural networks. The expressive properties of neural networks remain an active area of research.

### 3.3.2 Neural SDFs

In this thesis, we combine the SDFs of section 3.1 with the neural networks we have just described. We will train networks that approximate the SDFs of various surfaces, creating, thus, a *neural SDF*. This can be considered a regression task. However, in contrast to most machine learning settings, the aim here is to overfit the network. Our ultimate goal is to propose a method for editing the underlining surface through the parameters of the neural network. We present our method in chapter 5. In the next chapter, we examine previous work on surface representations, particularly on neural SDFs.

# Chapter 4

# Related Work

## Contents

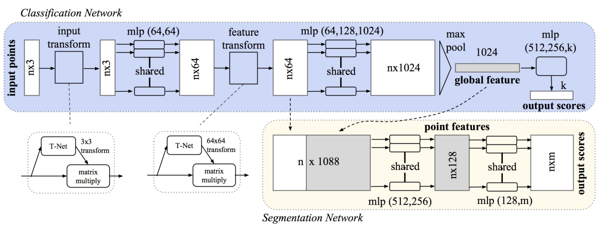
In this chapter, we review some of the choices for surface representation and their usage in the context of machine learning. In section 4.1 we discuss various explicit representations and present some characteristic works for each. Afterwards, in section 4.1, we analyze implicit representations in depth, focusing on neural SDFs.

## 4.1 Explicit Representations

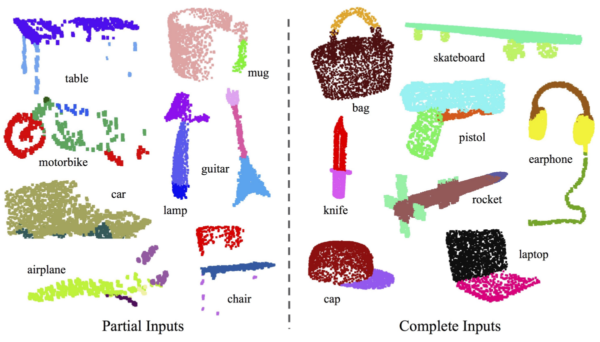
### 4.1.1 Point Cloud

A point cloud is just a set of points without any further structure. Usually point clouds are the most readily available real world data because it is the type of data produced by sensors, like LiDARs and depth cameras, that are used to scan real world objects. Consequently many of the works discussed here use datasets of point clouds.

PointNet [53] uses point clouds as the input to a neural network in order to do object classification and segmentation. In the case of classification the output of the network is a single vector of probabilities for each label, while in for segmentation it is a similar vector but for each point of the input. In both cases, the authors extract a global descriptor and since point clouds are unordered this needs to be done in a permutation invariant manner. To this purpose they use multilayer perceptrons (MLPs) to process the points individually and produce feature vectors and afterwards apply a max pooling operation over each feature dimension. The MLPs are shared across the points. This global descriptor is then passed to another MLP. They prove that this approach leads to a universal approximator. Another issue they address is invariance to rigid transformations of the input. For example a simple rotation or translation should not affect neither classification nor segmentation. They utilize networks that predict transformations to be applied to the input or an intermediate feature space. These transformations also need to be predicted in a permutation invariant manner and so the structure of the networks mimics the one described above. The overall architecture can be seen in figure 4.1a. In figure 4.1b some segmentation examples are shown.



(a) The architecture of PointNet.



(b) Segmentation examples from PointNet.

Figure 4.1: PointNet figures. Figures taken from [53]

Achlioptas et al. [1] use variational autoencoders (VAEs), generative adversarial networks (GANs) and gaussian mixture models (GMMs) in order to generate point clouds. The encoder part of the VAE as well as the discriminator of the GAN follow the architecture of PointNet [53] described above.

A important limitation of point clouds is that they do not define topology since they are disconnected points and so they do not comprise a real surface. Because of that they do not offer themselves to problems where nice visualisation is important. Also methods that use point clouds many times to have incorporate a fixed size to their architecture.

### 4.1.2 Polygon Mesh

Mesh is the most versatile and popular representation in computer graphics. Graphics cards have been optimized in order to render meshes making them one of the most, if not the most, efficient representation. It is only natural that it was one of the first to be used in machine learning and is still of research interest. A mesh in its simplest form is defined by a set of points called vertices and a set of faces, in most cases triangular or quadrilateral (referred to as triangle mesh and quad mesh, respectively).

A very popular application of meshes is for 3D morphable models (3DMMs) for faces. In these models the dataset comprises meshes of faces that are said to be in dense correspondence, which means that they all are isomorphic and that corresponding vertices are positioned to the same facial features. Of course the datasets that are available most ofter do not comply to the above specifications and a pre-processing of the meshes is required. Afterwards, since the meshes are then in dense correspondence, they can be described by their vertices' displacements relative to a tamplate mesh, which can be a mesh whose vertices' positions are the mean positions of the corresponding vertices in the dataset. A statistical analysis of these displacements that usually aims at dimensionality reduction gives the 3DMM. This is the approach taken by Booth et al. [11] where they learn such a model from 10,000 facial meshes.The pre-processing of the data is done by fitting a template to each mesh via a method called non-rigid iterative closest point (NICP) [2]. The statistical analysis performed is principal component analysis (PCA) which leads to the following linear model:

$$X^* = \bar{X} + \sum_{i=1}^{k_a} a_i U_i$$

where $X^*$ are the displacements, $\bar{X}$ are the mean positions, $k_a$ is the number of principal components, $U_i$ are the components and $a_i$ are the coefficients of those components. In figure 4.2 examples of faces generated by this model are shown.

More recent works have used neural networks to produce meshes. Kulon et al. [41] use images to predict hand poses, which are given by a mesh. To do that they employ a feed forward neural network which comprises a ResNet-50, followed by a mesh convolutional decoder. Mesh, and more generally, graph convolutions extend the convolution

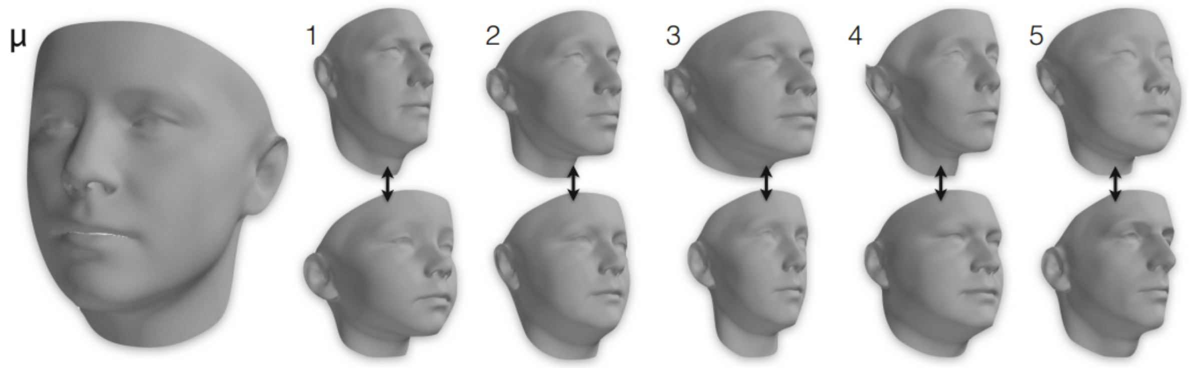


Figure 4.2: Examples of faces generated from the 3DMM of [11]. Left, the mean face is shown. The other columns correspond to one of the first five principal components' coefficients changing while the rest remain 0. Figure taken from [11].

operator to graphs. The decoder in the paper discussed here works by applying a kernel to the neighbourhood of each vertex. A comprehensive overview of the different ways of extending the convolution operator to graphs is provided by Bronstein et al. [12].

There is an extensive literature that uses meshes in machine learning and in conjunction with neural techniques. We refer to the surveys by Egger et al. [24] and Zollhöfer et al. [76] for thorough exploration of the techinques and applications for faces.

Even though meshes are versatile and very efficient to render, they have their own weaknesses. In most works, including the ones we discussed above, a certain template is used which has a fixed topology which cannot be altered, so meshes are not convenient when the objects being modeled are expected to differ in that manner. Its resolution is another aspect of the mesh that cannot be easily modified by most methods.

### 4.1.3 Voxel Grid and Octree

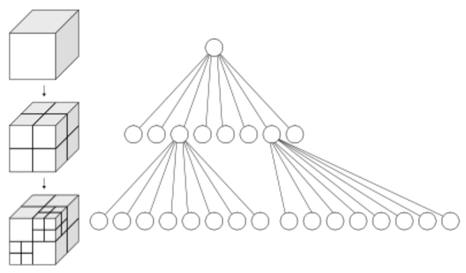


Figure 4.3: Visualisations of an octree as subdivisions of a cube (left) and as a tree (right). Figure taken from Wikipedia

A voxel is analogous to a pixel but for 3D. Like pixel it is a portmanteau derived from the words volume element (pixel comes from picture element). A voxel grid is nothing more than a subdivision of a cube into smaller cubes. In theory the grid could be rectilinear but mostly cubes are used because of their symmetries. An octree (figure 4.3), as the name suggests, is a tree data structure whose nodes have exactly eight children, or zero if they are leafs, of course. The eight children correspond to the subdivision of a cube by a factor of 2. The nodes of octrees are commonly reffered to as voxels as well. We group these two representations together because they are highly related. An octree can be viewed as an adaptive spatial subdivision leading to a sparse representation, while a voxel grid is the corresponding dense representation. However, octrees differ in

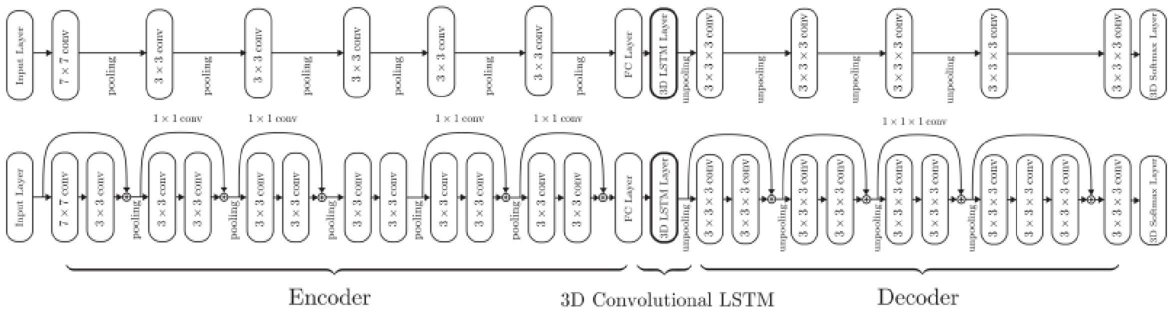


Figure 4.4: The two architectures for 3D-R2N2. Figure taken from [19]

that they are hierarchical structures and can leverage the levels of this hierarchy to store information at each one.

Unlike the rest of surface representations discussed in this chapter and although these two have inherent 3D characteristics, they need not define a 3D shape. The voxels could hold various types of information such as the values of a function sampled at the their centers, or feature vectors. The latter is what Takikawa et al. [67] do for example. In this section we are going to deal with voxel grids and octrees that hold occupancy information, that is whether a shape intersects a given voxel.

The natural way to incorporate voxel grids in neural networks is to use convolutional layers which have shown great success in image processing. After all images are just pixel grids. Choy et al. [19] use both kinds of convolutions networks in order to achieve object reconstruction from images. Specifically, the authors encode images using a 2D convolutional network and decode them into occupancy voxel grids using a 3D convolutional network (called deconvolutional because it uses upscaling layers). Between the two they place a recurrent network. This allows them to combine single and multi view reconstruction with a single network. The recurrent network is either a long short term memory (LSTM) network or a gated recurent unit (GRU). In both cases the basic structure of the network is altered by using 3D convolutions which imposes a structural regularisation. The resulting network is dubbed 3D-R2N2 from 3D recurrent reconstruction neural network. In figure 4.4 the two architectures proposed are shown.

Because the computational and memory requirements when storing all the voxels in a grid scales cubically with the resolution of the grid (assuming we subdivide all 3 dimension by the same number), it is difficult to achieve high resolutions. When used to represent surfaces, however, the majority of the voxels are expected to be empty since a surface is a 2D object and the occupied voxels should scale squarely with resolution. To address this issue researchers have tried using octrees instead which store only the occupied voxels for each subdivision layer. Even when a volume is to be modeled, an octree is more efficient because it can use the larger voxels of coarser levels to cover the interior of the volume. The new challenge, then, is how to generalize convolutions to work with octrees. Both Tatarchenko et al. [70] and Häne et al. [33] attempt to do just that.

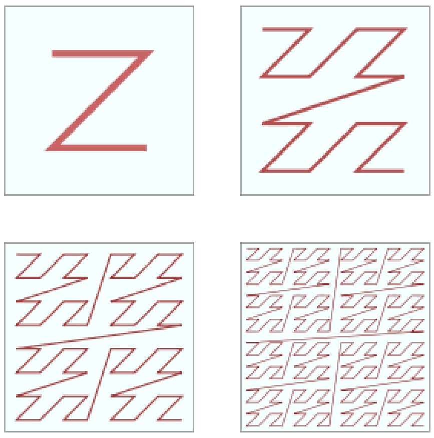


Figure 4.5: Z-order curve. Figure taken from Wikipedia

In [70] the authors use an efficient memory representation for octrees, sometimes referred to as linear octrees. The occupied voxels in each level of a tree are stored in a hash map with keys that are given by their Morton codes. Morton code, also known as Z-order [1], is a way to map multidimensional integer coordinates to an one dimensional one (figure 4.5). They construct a network whose layers output octree levels. These layers hold features for each voxel which are used by another layer to predict the probabilities of whether the voxel is empty, occupied or mixed, meaning that it need to be subdivided further. They propose two types of layers. The first one is an up-convolutional layer. It uses a voxel's and optionally its neighbors' features (this depends on the size of the filter) to produce the features of its children. The second one uses either the predicted label for each voxel to decide which should get propagated to be processed by the next layers. Besides the ones predicted as mixed, their neighbors might need to be propagated as well since they might be needed by the convolutional layer. Finally they use a softmax loss for each level of the octree.

Instead of an octree as it has been described so far, the authors of [33] use a structure they dub voxel block octree. This is an octree whose nodes instead of features for the respective voxel hold a voxel grid (or block) contained in the former. Since these voxel blocks are dense, standard convolutional networks can be used to process them. For every voxel three probabilities are computed. Contrary to [70] these indicate whether a voxel is inside or outside the volume or whether the boundary intersects it. A node's children are constructed if the maximum of this last probability over the node's voxel grid exceeds a certain threshold.

The above representations while surpassing the limitations of meshes by having flexible topology and adaptive resolution, have the disadvantage of not creating smooth surfaces. Due to the fact that they are a collection of axis aligned cubes they create jagged boundaries and cannot approximate the surface's normal vectors, which can have a very negative effect when the lighting is important in rendering.

## 4.2 Implicit Representations

The term implicit refers to something that is not given directly. In the context of geometry, an implicit surface comprises points which satisfy an equation:

$$F(x) = 0 \tag{4.1}$$

[1]https://en.wikipedia.org/wiki/Z-order_curve

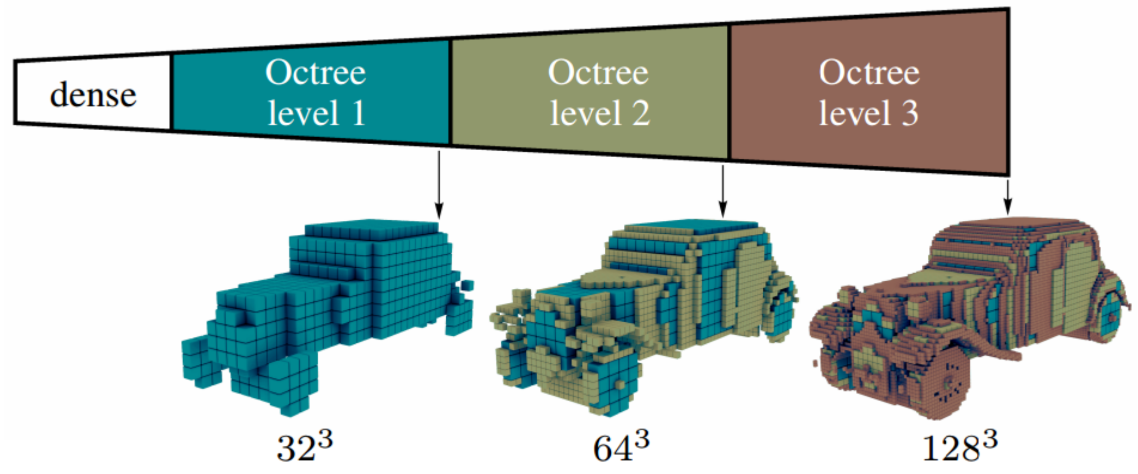


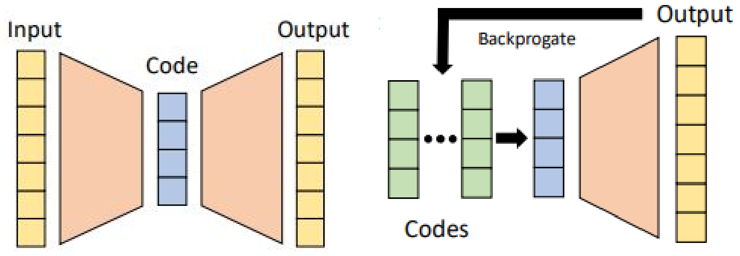
Figure 4.6: Example of a reconstruction by OGN. Figure taken from [70].

For example the equation $x^2 + y^2 - 1 = 0$ is an implicit representation of a circle with unit radius. Contrast this with $x = cos\theta$, $y = sin\theta$, $\theta \in [0, 2\pi)$, which is a parametric representation of a unit circle. The use of implicit surfaces in computer graphics is as old as meshes and octrees. However, meshes have proven much more efficient and so have become ubiquitous. There have been, though, continual attempts to use implicitly defined surface in computer graphics and machine learning.

In the shader art community analytic implicit representations have been used to render from simple primitives to complex scenes and fractal objects. Inigo Quilez has been at the forefront of these methods. Information and examples of these can be found at his site [55] and on Shadertoy [62] which is a site made by Quilez to host user submitted shaders.

On the other hand, in the machine learning community earlier approaches have relied upon radial basis functions (RBFs) [13] to represent the function $F$ of equation 4.1. More recently, the concurrent works of Park et al. [50], Chen and Zhang [16], and Mescheder et al. [47] sparked great interest for implicit representations. In these works, and the works that ensued, the function $F$ of equation 4.1 is given by a neural network. More specifically, in the first, the function that the network tries to learn is the signed distance function that we defined in section 3.1. In the latter two, it is the occupancy function, that is one that takes the value 0 inside the surface and 1 outside it. In both cases it is presupposed that the surface that is being modeled is closed. Sometimes, in the bibliography, functions that do not give a certain distance to the surface are also referred to as signed distance functions and the above is called a metric signed distance function to distinguish it. Here, we use the term signed distance function or SDF to refer to definition 3.9, unless explicitly stated otherwise.

The occupancy function has been used in other works [17, 47] as well. We will, however, focus on the distance representation, since it is where the bulk of research has focused as well.



(a) Encoder-decoder (b) Autodecoder

Figure 4.7: Visual comparison of encoder-decoder and autodecoder architectures. Figure taken from [50]

We will now analyze the work of Park et al. [50] in some depth as it pioneered these implicit representations. The learning of the signed distance function to a surface is formulated as a regression task. The authors use human made meshes from Shapenet [15] as the target surfaces which they normalize to a unit sphere and in order to form the dataset they sample points in space with greater focus on the space closer to the mesh. The loss function they use is a clamped $L_1$ loss, that is:

$$L(y, s) = |clamp(y, \delta) - clamp(s, \delta)| \tag{4.2}$$

where $y$ is the output of the neural network at some point $x$, $s$ is the ground truth signed distance at that point and $\delta$ is a positive constant. The clamping is done so that is predicted more accurately near the surface. The clamped signed distance is also called truncated signed distance function, this is not to be confused with the goal of the above loss function since the output of the network is also clamped.

Besides training a network to model the SDF of a single surface, they also use networks to model classes of shapes. Such a task can be achieved by using an encoder-decoder. However, the authors propose an architecture, inspired from they dub autodecoder, which they cite from [68]. In order to train an autodecoder architecture code vector is attached to each shape in the dataset. These codes get concatenated to the coordinates that are given as input to the network. During training these codes get optimized along the network weights, while during inference the weights are held constant and it's only the codes that get optimized. A advantage of an autodecoder over an encoder-decoder one is that, since the encoder part is not pertinent to the problem, there is no need to design and train it. A visual comparison between the two can be seen in figure 4.7. Finally, the architecture used in the paper is a feedforward network with skip connections and weight normalization [60].

Subsequent works have experimented with the loss and the architecture of the network used. We will review the different proposals starting with the form of the loss function.

### 4.2.1 Loss funtions

#### Sign agnostic loss

Computing ground truth signed distance values can be difficult, so the authors of [3] propose a loss function in order to learn a signed representation with unsigned ground

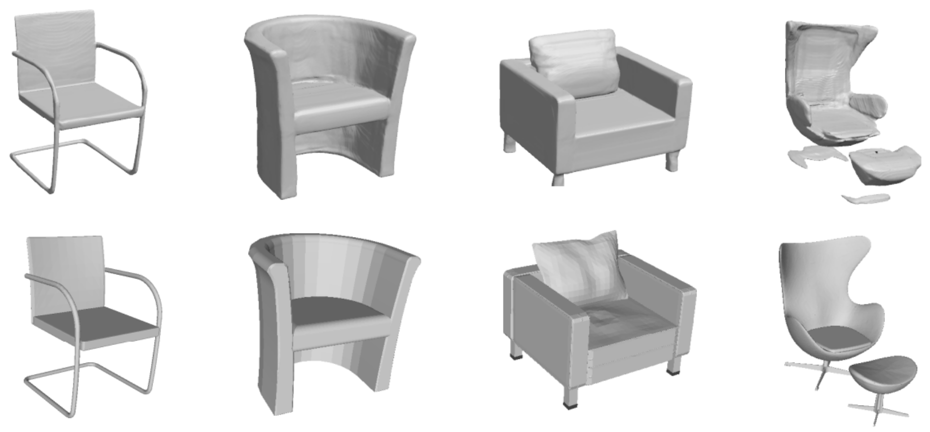


Figure 4.8: Examples of reconstructions by DeepSDF. First row shows the reconstructions, while the second the ground truth. Figure adapted from [50].

truth values. They propose the following function:

$$sal(x) = \Big| |f(x;\theta)| - d(x,S) \Big| \tag{4.3}$$

This, however, has the side-effect of introducing some undesired minima. Both the signed and unsigned distance functions, as well as their opposites constitute minima. The unsigned distance or its opposite is unlikely to be obtained as they have discontinuous derivatives. The opposite of the signed distance function is equally likely to be obtained. To alleviate that problem the authors propose an initialization scheme that as the width of the network goes to infinity, the output of the network converges to the signed distance to the sphere.

### Level set loss

Instead of sampling points using the target surface, the authors of [5] sample the level sets of the network (we will discuss sampling of the level sets latter in this report) and incorporate the level sets in the loss. In order to do so these points need to be expressed as functions of the network parameters so that the derivative with respect to them can be taken. The points are constrained to stay on the level sets as they change with the parameters, which gives the following equation:

$$f(p(\theta);\theta) = c \tag{4.4}$$

where $f$ is the network function, $p$ is a point on the $c$-level set and $\theta$ is the parameter vector. If we differentiate the equation above we get:

$$\nabla_p f \nabla_\theta p + \nabla_\theta f = 0 \tag{4.5}$$

This is reminiscent of what in physics is called material derivative. This underconstrains the gradient of the point, which is expected since the point can move in an infinity of directions and still stay on the level set. So, the authors use the Moore-Penrose pseudo-inverse of $\nabla_p f$ to solve for $\nabla_\theta p$. Now, since gradient descent uses only first order derivatives, the function that gives the point needs to match only this derivative and the proposed one is this:

$$p(\theta) = p - (\nabla_p f)^+ \left(f(p;\theta) - c\right) \tag{4.6}$$

Then the loss using the level sets is:

$$\sum_{c \in C} \left[ \int_{l_c(\theta)} |d(x,S) - |c||^p \, dv(x) \right]^{\frac{1}{p}} \tag{4.7}$$

where $C$ is the set of levels, $l_c$ is the c-level set, $S$ is the target surface and $dv(x)$ is the normalized volume element of $l_c$. Inside the the integral we can see that the sign agnostic version of the per point loss is used. In practice the integrals in the loss are approximated in practice by the sample mean. If we compare the loss above with the regression-type loss, we will see that the difference between the two, besides that the sampled points will generally have different distributions, is in the per point derivative with respect to the parameters. More specifically, in the loss examined here an extra term is present in the derivative and that is:

$$\left(\nabla_p d(p,S)\right) \left(-\left(\nabla_p f\right)^+\right) \tag{4.8}$$

This can be interpreted as a weight that encourages the parameters of the network to focus more on points that reduce their distance by moving along the normal direction of their level set.

### Normal vector loss

Oftentimes besides the positions of points on a surface information about the orientation of the surface at those points is available in the form of normal vectors. This is the case for directed point clouds and meshes with per-vertex normals. A cool feature of implicit representations is that normals do not need to be represented separately, but rather are equal to the gradient of the network function with respect to its input. Automatic differentiation packages allow us to easily compute this and do so in a differentiable manner, meaning that the result can be further be differentiated with respect to the network parameters as needed for gradient descent algorithms. The loss used per point, then, can be the euclidean distance between the gradient and the ground truth normal $n_x$, this is used in [31]:

$$\|\nabla_x f(x;\theta) - n_x\| \tag{4.9}$$

When using normals, however, we are more concerned with their direction than their magnitude. A loss more well adapted to that is one minus the cosine similarity:

$$1 - \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \|\mathbf{b}\|} \tag{4.10}$$

This is also called cosine distance and is used in [64].

Two versions of this normal loss analogous to the sign agnostic loss proposed by [4] are:

$$\min \{\|\mathbf{a} - \mathbf{b}\|, \|\mathbf{a} + \mathbf{b}\|\} \text{ and } |\sin \angle(\mathbf{a}, \mathbf{b})| \tag{4.11}$$

### Implicit geometric regularization

Sampling points around a surface and computing their (unsigned) distance can still be computationally expensive. Therefore, researchers have tried to find ways to circumvent that. [31] and [64], in the spirit of [63], formulate the learning of the SDF as a boundary value problem. As we mentioned at the end of section 3.1, SDFs satisfy an equation called eikonal equation (that is why the term discussed here is also referred to as the eikonal term):

$$\begin{aligned} &\nabla_x f = 1 \\ &f(x) = 0, \quad x \in S \end{aligned} \tag{4.12}$$

The above hard constraints are turned to soft ones to form the loss function:

$$L(f) = \lambda_S \mathbb{E}_{x \sim D_S} \big(|f(x)|\big) + \lambda_\Omega \mathbb{E}_{x \sim D_\Omega} \big(\big|\|\nabla_x f\| - 1\big|\big) \tag{4.13}$$

where $\mathbb{E}$ denotes the expected value, $D_S$ is a distribution whose support is the surface S, $D_\Omega$ is a distributions whose support includes the surface, typically the uniform distribution in an axis aligned bounding box, and $\lambda_S, \lambda_\Omega$ are Lagrange multipliers. With this the training of the network is semi-supervised, as it does not specify in any direct way the values that the network should obtain on points that are not on the surface. As in the case of sign agnostic learning careful initialization is required in order to avoid unwanted minima. Using a normal loss term (not sign agnostic version though) can also lead to the desired minimum since it specifies on what side of the surface the positive values of the SDF should lie.

### Empty space constraint

The above loss does not necessarily prevent the network to learn an SDF of a surface that is the superset of the target one. This lead to the formation of artifacts, blobs where there should be empty space. This is somewhat mitigated by the fact that such artifacts create discontinuities in the gradient which increase the eikonal term. However, when there are areas inside the bounding volume which are quite distant to the surface these artifacts are quite likely to appear. To the purpose of eliminating these the following term is employed in [64]:

$$\mathbb{E}_{x\sim D_\Omega}\left(e^{-af(x)}\right) \tag{4.14}$$

where $a \gg 1$. This term conflicts obviously with the first term of 4.13, yet, since the surface is expected to be a 2-dimensional set while $\Omega$ a 3-dimensional one the expected value above is not affected by the surface.

### 4.2.2 Network architectures

#### Implicit features

In [17] and [18] the authors discretize the input point clouds, converting them to voxel grids of some fixed resolution. These grids are then processed by a 3D convolutional network producing multiresolution features which are, in turn, fed to an MLP. In the first case the output is the occupancy, while in the latter, the unsigned distance. In figure 4.9, an overview of the architecture of [17] can be seen.

#### Sinusoidal activations

Instead of ReLU or leaky ReLU, the authors of [64] propose the use of sine as activation function, which yields impressive results across a variety of problems which can be formulated as boundary value problems, one of them being surface representation via its SDF, and the network they use for this is a simple MLP with sine activations for the layers except the last one which is just linear, which they dub *SIREN*. They also provide an initialization scheme for this network.

The success of the sine activations can possibly be explained by the work of [38], as it was used to explain the success of positional encoding and fourier features by [69]. Specifically, the authors prove that using sinusoidal functions in the first layer of a networks allows it to learn functions with higher frequency content.

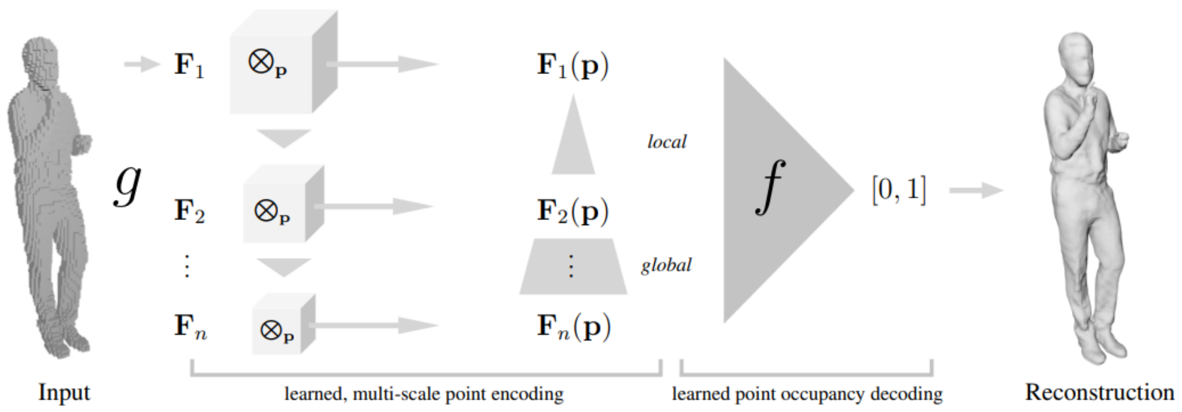


Figure 4.9: Overview of IF-net. Figure taken from [17]

#### Meta learning

Meta learning refers to the process of learning how to learn. Usually in machine learning the object is to train a single network that can be used for different tasks. Two meta learning approaches has been used by [64] and [65] to learn SDFs. The former employs hypernetworks while the latter the MAML (Model Agnostic Meta Learning) algorithm. These works attempt to represent a class of shapes, as was done in [50] and presented in the beginning of this section, by considering the SDF of each shape in the class as a different task.

Hypernetworks [32] are networks that are used to produce the weights of another network. The input to a hypernetowrk is a task, which is a particular shape in this case, codified in some manner (with a feature vector for example) and the output is the parameters to an SDF network.

MAML [28] is an algorithm that allows for a single network to be adapted to a task after only a small number of gradient descent steps. This is also called few-shot learning. The idea is to perform a small fixed number of gradient descent steps for each task in a batch, beginning with the same parameters for each one, and, then, perform a step by using the gradient of the sum of the per task losses with respect to the initial parameters.

#### Hybrid techniques

By hybrid we refer to techniques that use a combination of explicit and implicit representations. Chabra et al. [14] use sparse voxels, Takikawa et al. [67] octrees, and Müller et al. [49] use a technique they name hash encoding. In the former two cases, the vertices of the voxels hold learnable features. In order to calculate the SDF prediction at a point inside a voxel the feature vectors are combined via trilinear interpolation and the result is passed to an MLP. In [67] there can be mixing between the levels of the octree so that a continuous level of detail can be achieved. By having features that locally describe the surface, highly detailed results are possible.

### 4.2.3 Visualizing and sampling neural SDFs

Representing a surface more than often presumes being able to visualize it in some way. After all, in most applications around computer graphics the visual is the ultimate criterion and goal. Accordingly, in this section we are going to discuss such methods pertaining to implicit representations.

Visualising 3D geometry, especially when done through the lens of a virtual camera or when some lighting simulation is involved is referred to as rendering. Rendering algorithms generally fall into two broad families: rasterization and ray tracing. Rasterization algorithms use a series of transformations to project objects onto the screen and consequently work better with explicit representations. On the contrary, ray tracing al-

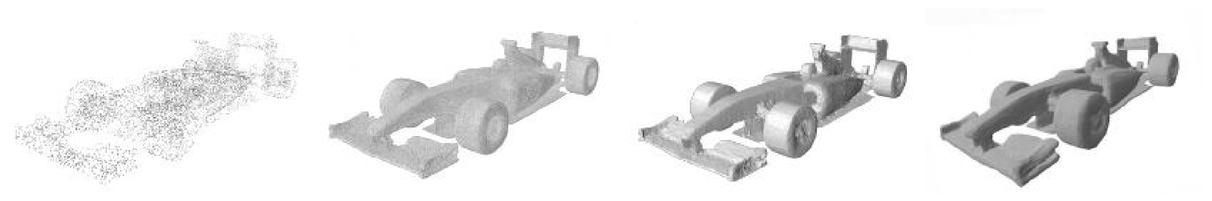


Figure 4.10: Renderings by [18]. Left to right: input point cloud, point cloud sampled with Newton-Raphson method, mesh reconstructed from samples point cloud with ball-pivoting and rendering with sphere tracing. Figure adapted from [18].

gorithms, as the name implies, use rays that start from the camera and intersect them with the scene geometry to produce the image. As these algorithms follow more closely the underlining physics of image formation they can generate more photorealistic results and incorporate effects that are due to light interactions with greater ease. Also, since the only requirement for a representation to work with ray tracing is the ability to find the intersection between the object and a ray they are are more versatile, although triangle meshes are preferred in this case, as well, because there exist very efficient ray-triangle intersection algorithms. All these advantages, however, come at the price of speed. Ray tracing is, generally, much slower. Therefore, it is, mainly, used when photorealism is desirable and time is not of the essence as is the case with animated films for example, while real-time rendering is done with rasterization and GPUs have been heavily optimized (and actually created) for this process. Even though this separation has been at effect for most of computer graphics' history, in recent years newer GPU models have hardware that support ray tracing, heralding an new era for computer rendering.

Most of the works presented previously do not actually use the neural network directly to render the zero-level set. Instead, they first convert it into a mesh or point cloud and use that for rendering. In [3, 4, 5, 18, 50, 64, 65] the marching cubes algorithm [43] is used to create a mesh. To do that the network function is sampled at a dense grid. Chibane et al. [18] propose an algorithm to densely sample the zero-level set. Essentially the same algorithm is independently developed by Atzmon et al. [5] and identified as a special case of the generalized Newton-Raphson method [6]. Firstly, points are sampled in the space around the surface, via a uniform distribution in its bounding box, for example. Then, each point is moved iteratively according to the following equation:

$$x_{i+1} = x_i - f(x_i)\frac{\nabla f(x_i)}{\|\nabla f(x_i)\|} \tag{4.15}$$

Essentially the point is moved along the direction of steepest descent, a distance equal to $f(x)$. If $f$ is exactly the SDF, then one iteration is enough for the point to end up on the zero-level set, since the direction of steepest descent is the direction towards the closest point of the zero-level set and $f(x)$ is equal to the distance. The neural networks, however, are not exact SDFs, but still the point can get very close to the surface with only a few iterations. In [18] a small variation of what was described above is used. An initial set of points is constructed as presented, and from this set, only the ones with distance below a certain threshold are retained. Afterwards, points from this set are sampled, perturbed with gaussian noise and, then, reprojected on the surface the same way as the initial ones. The set is augmented by the points that satisfy the threshold condition. This process is continued until the desired number of points is reached. Having produced this point cloud, it can be rendered as is or converted first to a mesh via classic algorithms such as ball-pivoting [8]. The images in the middle of 4.10 show a sampled point cloud and the mesh produced by ball-pivoting.

Another approach is to use ray tracing to render the zero-level set. As aforementioned, ray tracing requires a method of finding the first intersection (if one exists) along the ray with an object. In the case of primitives like triangles and spheres, there exist analytic formulas to do that. In general, for implicit representations a root finding algorithm must employed. More concretely, the equation whose roots we must find is the following:

$$\begin{aligned} &F(\mathbf{p}(t)) = 0 \\ &\mathbf{p}(t) = \mathbf{p}_0 + t\mathbf{d} \end{aligned} \tag{4.16}$$

where $\mathbf{p}(t)$ is the parametric ray equation and $\mathbf{p}_0$, $\mathbf{d}$ are the ray origin and direction respectively

For distance functions (and Lipschitz functions in general), a variation of ray tracing called sphere tracing proposed by [35] can be used to find the intersection. The idea of sphere tracing is to progressively advance the ray until an intersection is found or a termination condition is met. A ray is considered to intersect the surface if the value of $F(\mathbf{p}(t))$ becomes adequately small. The simplest termination condition is that a maximum numbers of steps is taken. The step size in sphere tracing is equal to the value of $F$ at the current point. If $F$ is the distance function, then that is the largest step we can safely take and not intersect the surface. A demonstration of the process is shown in figure 4.12

Sphere tracing is used by [14, 18, 42, 49, 67] to render neural implicit representations. Liu et al. [42] do so in a differentiable manner so that they learn the implicit representation from images. In [14, 67] the authors use voxels and octrees respectively to accelerate rendering and achieve higher detail. Sphere tracing requires the evaluation of the network at multiple point along the ray, which is an expensive operation. With the above

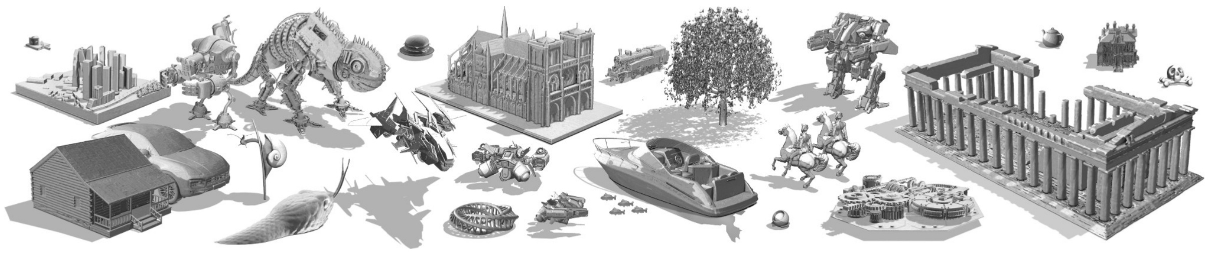


Figure 4.11: Renderings made by using sphere tracing with neural SDFs. Figure taken from [67].

representations many of these computations can be avoided because because we firstly find the intersected voxels and, then, use sphere tracing inside them. Takikawa et al. [67] in particular, achieves real-time rendering. Figure 4.11 shows their renderings of various models. Müller et al. [49] with their technique, also achieve real-time rendering, aw well as, real-time training.

### 4.2.4 Editability

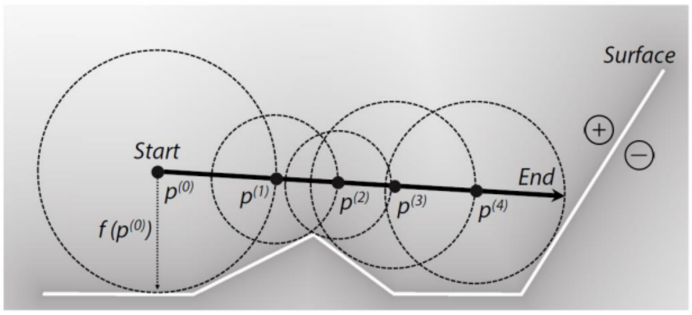


Figure 4.12: 2D illustration of sphere tracing. Figure taken from [42]

Apart from the efficiency issues of neural implicit representations, another issue which we study in this report is the editability of these representations. With this term we mean the ability to manipulate the surface in an interactive way. To our knowledge there are not many works which attempt to do that.

Hao et al. [34] propose a two level representation in order to manipulate an object in a semantic manner. One representation is a coarse one given by a collection of primitives with analytic SDFs whose parameters (e.g. the center and radius of a sphere) are given by a network which takes as input a shape code. The other one, more fine detailed, is a network with an autodecoder architecture. The two representations are trained for a class of objects in tandem. After training, one can manipulate the resulting shape by specifying an edit to the primitives' parameters and adjusting the shape code accordingly in a way

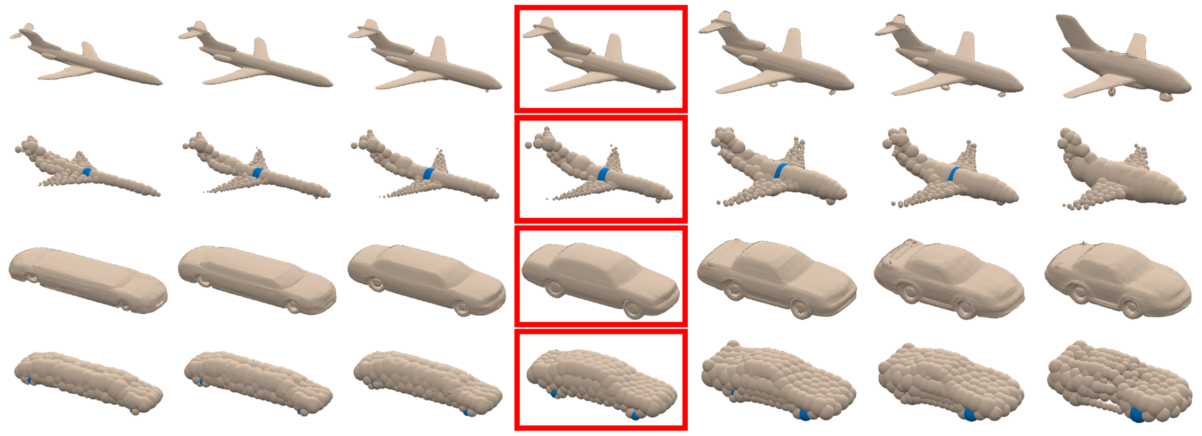


Figure 4.13: Examples of shape manipulation by [34]. The first and third rows show the fine representations, while the others show the representations by the collection of primitives. The blue spheres are the ones whose parameters are being edited, the radius of a single sphere in the case of planes and the distance between the centers of two spheres in the case of cars. Figure taken from [34].

similar to the inference process of an autodecoder. Examples of semantic manipulations possible by this system are shown in figure 4.13. This approach of searching in the shape space is also used by Deng et al. [22], where instead of manipulating the parameters of some primitives a sparse set of points is used.

Deng et al. [21] and Mu et al. [48] both deal with articulated objects, that is, objects which have joints and are rigid, besides the deformation due those. For example, human bodies are modeled as articulated objects. Deng et al. use an occupancy representation, while Mu et al. an SDF. Both use the joint parameters, which are the angles between, as inputs to a network.

## 4.3 Conclusion

Neural implicit representations have seen an explosive advancement in recent years. They have surpassed other representations and the exploration of applications where they can be useful is still at its beginning. Already there is interest of utilizing implicit representations in robotics [39, 66]. A survey by Xie et al. [75] and its accompanying project website preserve a large collection of works relating to implicit representations. There are still a lot of open problems as, for instance, are the efficiency improvement of these representations, their effectiveness in tasks for which other representations have traditionally been used, their generalization capabilities and their editability. In this thesis, we attempt to tackle the problem of editability, which has not been thoroughly examined. In the next chapter we present our method.

# Chapter 5

# Sampling and Editing Neural SDFs

## Contents

Our aim in this thesis is to propose a method for editing the surface represented by a neural SDF. As we saw in the previous chapter, even though there exist some works that allow manipulating the surface, they are somewhat restricted. Hao et al. [34] allow editing in a semantic manner and the editing of Deng et al. [22] follows a similar pattern, while Deng et al. [21] and Mu et al. [48] handle only articulated objects. Instead, we want to allow the user to deform the surface interactively in a free form manner. We draw inspiration from 3D sculpting software, which use tools that are called brushes to deform a surface in a localized region around a point, on the surface, specified by the user. In this chapter, we present our proposed method.

We will refer to the selected point as the *interaction point*, to the area around it as the *interaction area* and to the process in general as an *interaction* or *edit*. The main problem we encounter is how to enforce locality. In order to change the surface expressed by the network, its parameters must change though some sort of training. However, a change to the parameters of a neural network is expected to affect its output for an unbounded region of the input space. Consequently, naively using samples placed only in the interaction area, will ostensibly distort the rest of the surface as well. We confirm this experimentally in chapter 6. To ameliorate this adverse effect as much as possible we propose to combine samples from the surface expressed by the network (before training) outside the interaction area and from the desired edit. Below we will describe every part of the process in detail. Firstly, we define the loss function we use, in section 5.1. Then, in section 5.2, we deal with surface sampling. In section 5.3 we present our formulation of sculpting brushes and, afterwards, in section 5.2, the way we sample the interaction.

## 5.1 Loss function

We base our loss function on the one used in SIREN [64]. Accordingly, the loss function $L$ is as follows:

$$L(\theta) = L_S(\theta) + L_{eik}(\theta) + L_{es}(\theta) \tag{5.1}$$

$$L_S(\theta) = \mathbb{E}_{p_S}\{\lambda_1 \, |f_\theta(x)| + \lambda_2 \, g\left(\nabla_x f_\theta(x), n_x\right)\} \tag{5.2}$$

$$L_{eik}(\theta) = \lambda_3 \, \mathbb{E}_q\{\big|\|\nabla_x f_\theta(x)\| - 1\big|\} \tag{5.3}$$

$$L_{es}(\theta) = \lambda_4 \, \mathbb{E}_q\{e^{-\alpha|f_\theta(x)|}\} \tag{5.4}$$

$$g(a, b) = 1 - \frac{a \cdot b}{\|a\|\|b\|} \tag{5.5}$$

where $\lambda_1, \lambda_2, \lambda_3$ and $\lambda_4$ are balancing weights, $S$ is the target surface, $p_S$ is a distribution on the surface, $q$ is uniform distribution on a bounding box, $\theta$ is the parameter vector, $f_\theta$ is the network function, $g$ is the cosine distance, $n_x$ is the normal vector at $x$ and $\alpha$ a large positive number. It can be seen that the loss function comprises multiple terms. These terms were discussed in section 4.2.1 of the previous chapter. More specifically, it follows implicit geometric regularization (cf equation 4.13), with the inclusion of a term for the normals (second part of equation 5.2) and an empty space constraint (equation 5.4). Implicit geometric regularization is crucial to our method because it allows us to train the model without computing ground truth distances, which would be difficult in this case. Now, the only thing we need, in order to apply the loss above, is a point cloud (with normals) of the target surface, that is used to approximate the expected value in equation 5.2.

## 5.2 Surface Sampling

As we have seen in the previous chapter, two prior works [5, 18] have proposed similar algorithms that sample the 0-level set of a neural network function. We gave an overview of the algorithm in section 4.2.3. Here, we delve into more details, as our proposed algorithm builds on top of that. As we previously explained, a key procedure in the algorithm is the projection of points in space onto the 0-level set, i.e. the surface whose SDF the network expresses, using a Newton-Raphson type iterations. We give pseudocode for this in algorithm 5.1. Based on theorem 3.16 it follows that for a true SDF only one iteration suffices, but since neural SDFs are approximations, more are required, though, not many. In practice, we use 7 iterations.

**Algorithm 5.1:** Function that projects a points to the 0-level set of an SDF

```
Function ProjectSamples($X_{in}$, $f$, $n_{iter}$):
    Data: $X_{in}$ : A set of samples to be projected
          $f$ : SDF
          $n_{iter}$ : Number of iterations
    Result: The projected samples
    $X_{out} \leftarrow \emptyset$
    foreach $x_{in} \in X_{in}$ do
        $x_{out} \leftarrow x_{in}$
        for $i = 0$ to $n_{iter}$ do
            $x_{out} \leftarrow x_{out} - f(x_{out}) \frac{\nabla f(x_{out})}{\|\nabla f(x_{out})\|}$
        end
        $X_{out} \leftarrow X_{out} \cup \{x_{out}\}$
    end
    return $X_{out}$
end
```

In order to sample the 0-level set, the algorithm begins by uniformly sampling a bounding box around the 0-level set and projecting these samples onto it. Afterwards, from the projected samples, only those whose absolute value of the SDF is below a certain threshold are kept. Besides using too few iterations, the main reason why some samples' distances do not meet this requirement is because, close to the boundary of the bounding box, the neural network does not approximate the SDF well enough, leading to samples that started there needing many more iterations to converge to the surface or not converging at all. Atzmon et al. [5], actually, do not reject samples because the ones that do not converge do not pose that serious an issue. Here, we follow Chibane et al. [18], because we do require the samples to lie as close to the surface as possible. In our version of the algorithm we also reject samples that got moved outside the bounding box. This can happen, also, due to poor approximation close to the boundary of the bounding box. Its reasonable to question whether such a check is required, since, one might expect, samples outside the bounding box to be rejected by the threshold condition. However, because the network is not given samples outside the bounding box during training, it can take values that are below the threshold and so this condition, by itself, is ineffective.

**Algorithm 5.2:** Function that creates new samples by perturbing others with gaussian noise and reprojecting them

```
1 Function ExtendSamples($X_{in}$, $f$, $n$, $\sigma$):
      Data: $X_{in}$ : Inital set of samples
            $f$ : SDF
            $n$ : Number of samples
            $\sigma$ : Gaussian noise's standard deviation
      Result: The new samples
2     $X_{sampled} \leftarrow$ Sample $n$ elements with replacement from $X_{in}$
3     $X_{perturbed} \leftarrow \{x + z_x \,|\, x \in X_{sampled}, z_x \sim N(0,\sigma)\}$
4     $X_{out} \leftarrow$ ProjectSamples($X_{perturbed}$, $f$, $PROJECTION\_ITERS$)
5     return $X_{out}$
6 end
```

Due to rejection, the desired number of samples might not be reached. Even if no sample gets rejected, it is beneficial to begin by sampling less than the desired number of samples and, then, extend the initial set, because this leads to a better distribution. Imagine using a true SDF, then a point in space would be projected to the point on the surface closest to it (that is $\mathcal{C}(x, S)$, see definition 3.6). The space can, then, be partitioned to sets each corresponding to a point on the surface, that is, for each point $y$ on the surface, the points which get projected to it belong to such a set (we can safely ignore singular points, see remark of corollary 3.15). More formally, the set of the partition $\mathcal{P}(\dagger)$ corresponding to a point $y$ on the surface $S$ is:

$$\mathcal{P}(y) = \{x \,|\, \mathcal{C}(x, S) = \{y\}\} \tag{5.6}$$

According to theorem 3.13, $\mathcal{P}(y)$ comprises line segments, so the distribution on the surface that results from projecting points uniformly distributed inside the bounding box is proportional to the length of the the segments of the sets $\mathcal{P}(y)$. Depending on the surface there can be quite a disparity amongst these lengths. Also, the fact that a neural SDF is an approximation can worsen the situation. Instead of using only samples projected in this manner, Chibane et al. extend the sample set by using the samples that got initially accepted. This is done by sampling with replacement from the existing samples and then perturbing these new samples with gaussian noise and reprojecting them onto the surface. Pseudocode for this is given in algorithm 5.2. The resulting samples are again as described above. This process is repeated until the desired number of surface samples is reached. The whole algorithm is given in 5.3, where $N_{iter}$, $PROJECTION\_ITERS$, $T$ and $SIGMA$ are constants representing the samples to be taken per iteration, the iterations for projection, the threshold and the gaussian's standard deviation respectively, whose values we set to 40000, 7, 0.009 and 0.04.

**Algorithm 5.3:** Functions that samples the surface expressed neural SDF

```
1  Function SampleSDF($f$, $BB$, $n$):
       Data: $f$ : SDF
             $BB$ : A bounding box of the surface
             $n$ : Number of samples
             $\sigma$ : Gaussian noise's standard deviation
       Result: The surface samples
2      $X_{iter} \leftarrow$ Sample uniformly $N_{iter}$ points inside $BB$
3      $X_{iter} \leftarrow$ ProjectSamples($X_{iter}$, $f$, $PROJECTION\_ITERS$)
4      $X_{iter} \leftarrow \{x \in X_{iter} \mid f(x) \leq T \wedge x$ inside $BB\}$
5      $X \leftarrow X_{iter}$
6      while $|X| < n$ do
7          $X_{iter} \leftarrow$ ExtendSamples($X_{iter}$, $f$, $N_{iter}$, $SIGMA$)
8          $X_{iter} \leftarrow \{x \in X_{iter} \mid f(x) \leq T \wedge x$ inside $BB\}$
9          $X \leftarrow X \cup X_{iter}$
10     end
11     return $X$
12 end
```

For our method surface samples need to be produced every training iteration. A straightforward way to do that is to use the algorithm described thus far. However, this approach has two major drawbacks. Firstly, it is time inefficient. Secondly, despite the what was discussed above, the distribution of the resulting samples can, still, be quite non-uniform. Inspired by the way the samples are extended by perturbing them, we produce the sample set for the next training iteration using the existing set. To that purpose we perturb and reproject every sample in the current set. Consequently, the algorithm for producing the next sample set is similar to the `ExtendSamples` function (algorithm 5.2), with the difference that the entire sample set is perturbed and projected instead of sampled elements of it. The other difference is that in this case we perturb a sample by adding to it a vector uniformly sampled from a disk tangent to the surface at the sample's location. To sample these vectors we use the technique described in section 3.2.2. The reason we opt for tangential perturbations, instead of gaussian noise is so

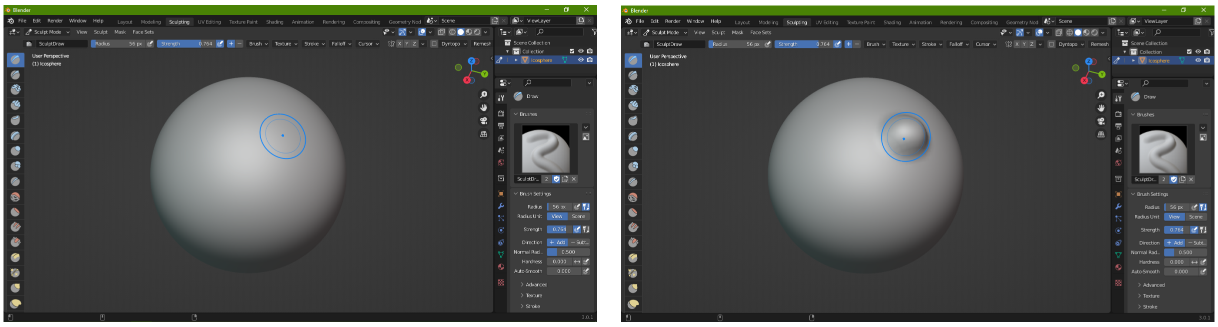


Figure 5.1: Example of sculpting in Blender. Left: a model of a sphere. Right: the same model after a stroke.

that we explore the surface as much as possible without moving too far from it. We use the *SIGMA* constant for the radius of the disks, although schemes where the radius is a function of some property, like the curvature, are potentially useful. If, after the projection some points of the sample set, fail to meet the requirements mentioned above, we use the `ExtendSamples` function with the set of accepted points as input to replace them.

This way of sampling forms a Markov Chain (see section 3.2.1). Naturally, the stationary pdf distribution is of interest. It is relatively easy to see that the requirements of theorem 3.27 are satisfied and, hence, the stationary distribution exists and has support over the surface. It is hard to theoretically reason about the shape of the distribution. However, we provide experimental results in Section 6.6, that demonstrate that our sampling process produces more uniformly distributed samples than the naive way outlined at the beginning of the previous paragraph. Uniformness is a desired property because it guarantees that every region of the surface will be included equally during training. Nevertheless, distributions which focus more on areas where the surface is more complex might be more desirable and, so, a topic for further research.

## 5.3 Brushes

One of the most popular techniques for designing 3D models of natural looking objects such as animals is 3D sculpting. It is named such because it is reminiscent of clay sculpting (this is why the objects in 3D sculpting have been referred to as digital clay), whereas a sculptor uses tools to remove/add material and shape the clay, in 3D sculpting the artist uses digital tools called brushes to edit a surface in analogous manners. The edits are usually done in strokes, which are series of interactions happening while holding down a mouse button (or pen in the case of drawing tablets). An example of an edit made in Blender [10] is shown in figure 5.1. Below we present our formulation of brushes and how they affect the surface inside the interaction area.

### 5.3.1 Brush Templates and Families

We define a brush template as a $C^1$ (or higher) 2D function defined over the unit disk centered at the origin which reaches a maximum value of 1 and vanishes at the unit circle (ideally its gradient and its higher derivatives vanish as well). Suppose $b_T(x)$ is a brush template, then the properties above are summarized as follows:

$$b_T : \{x \in \mathbb{R}^2 \,|\, \|x\| \leq 1\} \rightarrow \mathbb{R} \tag{5.7}$$

$$\max\{b_T(x)\} = 1 \tag{5.8}$$

$$\|x\| = 1 \Rightarrow b_T(x) = 0 \ (\wedge \nabla_x b_T(x) = 0) \tag{5.9}$$

We can, then, define a brush family $B_{r,s}$ parametrized over radius $r$ and intensity $s$:

$$B_{r,s}(x) = s\, b_T\left(\frac{x}{r}\right),\ r \in \mathbb{R}^+, s \in \mathbb{R} \tag{5.10}$$

The radius $r$ affects the size of the interaction area and the intensity $s$ controls how much the surface is deformed. As will become apparent a positive intensity creates a bump (extrusion), while a negative intensity creates a dent (intrusion) on the surface.

### 5.3.2 Radially Symmetric Brushes

Oftentimes, the effect of the brush is radially symmetric around the interaction point. This can be desirable because various useful deformations have this property, but, also, for ease of application since, otherwise, we would also need to parametrise the brush family with an angle of rotation as well. A radially symmetric brush template is a function of the norm of $x$ only and so can be expressed, using an 1D function $f$, as:

$$b_T = f(\|x\|) \tag{5.11}$$

In order for $b_T$ to comply with equations 5.7-5.9, $f$ must satisfy the following:

$$f : [0, 1] \rightarrow R_+ \tag{5.12}$$

$$max\{f(x)\} = 1 \tag{5.13}$$

$$f(1) = 0\,(\wedge f'(1) = 0) \tag{5.14}$$

The above requirements are similar to the ones satisfied by smoothstep functions. We are now going to explain how to use smoothstep functions to define brush templates.

### 5.3.3 Smoothstep Functions

In computer graphics, the need to transition smoothly from one real number to another arises very frequently. For this purpose, various functions are used, which take the value 0 for $x < 0$, the value 1 for $x > 1$, and go from 0 to 1 in the interval $[0, 1]$ in an continuously differentiable increasing manner, with vanishing derivatives at 0 and 1. In graphics, such a function is usually called *smoothstep*. Refer to the Inigo Quilez's site [55] for a presentation of some smoothstep functions.

If $f$ is a smoothstep function, then $f(1 - x)$ satisfies the requirements stated above and so a radially symmetric brush template ensues:

$$b_T(x) = f(1 - \|x\|) \tag{5.15}$$

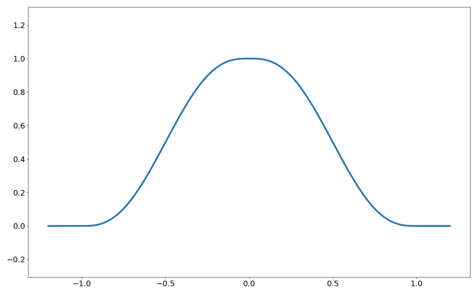


Figure 5.2: Profile of the radially symmetric brush template created with the quintic polynomial smoothstep function (equation 5.16).

Notice that, since smoothsteps are required to be increasing in $[0, 1]$, they are more restricted than what equations 5.12-5.14 dictate. Nevertheless, most commonly the deformation is desired to be more pronounced directly on top of the interaction point and, consequently, this increasing property (or decreasing, since we $1 - x$ as input to the smoothstep) is naturally suitable. For our experiments we use a quintic polynomial smoothstep function:

$$P_5(x) = 6x^5 - 15x^4 + 10x^3 \tag{5.16}$$

The profile of the resultant brush template is shown in figure 5.2.

## 5.3.4 Other Brush Types

Besides non symmetric brushes, which, as aforementioned, would require an angle parameter to fully take advantage of their functionality, another extension we can make to our formulation is to allow for *vector brushes*. By that term we mean using a brush template which is a vector function instead of a scalar one. Such a brush could allow to twist the surface around the interaction point. This, and perhaps other, extension to the brushes are interesting future directions.

## 5.3.5 Brush Application

In order to apply the brush at a point on the surface we consider it defined on the tangent plane at that point, with the $z$ axis aligned with the (outward pointing) normal. Due to the implicit function theorem [44] we can express the surface, which is the 0-level set of the network function, in a region of that point, as the graph of a 2D function over the same plane. We can, thus, apply the brush by simply adding the brush function to the latter, which means that the points on the surface are moved along the interaction point's normal a distance equal to the brush's value. If $S(x)$ is this 2D function, then, the function for the deformed surface $S_d$ is given by:

$$S_d(x) = S(x) + B_{r,s}(x) \tag{5.17}$$

It is now clear that we placed the requirement that the brush template and its derivatives vanish at the unit circle in order to maintain the connectivity and smoothness of the surface after the edit. It is also clear why positive intensity values create bumps and negative ones create dents.

## 5.4 Interaction Sampling

Having defined brushes and how they act upon a surface, we now move on to describing how to sample the interaction area in order to include these samples in the loss function for the training.

We need to produce samples that lie on the deformed surface, which means calculating their positions and normals. We begin by placing uniform samples on a disk tangent to the interaction point whose radius is the same as the brush's. In order to utilize equation 5.17 we need to compute the value of the implicit function $S(x)$ at the coordinates of each point. To that purpose we project the sampled points on the surface. Theoretically, for the analysis above to be applicable this should be a parallel projection along the interaction normal, however, we use the same procedure that we use for the surface sampling algorithm so that we do not affect the surface in a greater area then intended. This way tangent disk gets "wrapped" around the interaction point. Having found the corresponding surface points we simply add to them vector components along the interaction normal with lengths equal to the values of the brush at the coordinates of the tangent circle. This process is depicted in figure 5.3. Next, we compute the normals after the edit.

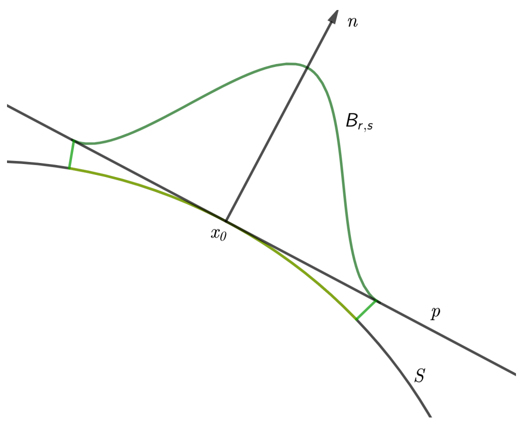


Figure 5.3: The brush application is demonstrated above. $S$ is the surface, $x_0$ is the interaction point, $n$ is the normal vector at $x_0$, $p$ is the tangent plane and $B_{r,s}$ is the brush function whose graph over $p$ is the dark green curve. The lighter green region on $S$ is the projection of the tangent circle.

Suppose that $f$ is a 2D function defined over a 3D plane (its gradient $\nabla f$ then lies on the plane), and $n$ is the normal vector to that plane (analogous to the z axis), then the following is an unnormalized vector, perpendicular to the graph of the function:

$$-\nabla f + n \tag{5.18}$$

Accordingly, what we need in order to compute the deformed normals is the gradient of the 2D function whose graph is the deformed surface ($S_d(x)$ of equation 5.17). As we explained previously, this function is the sum of the brush function and the function defined implicitly by the network. We can directly calculate the gradient of the brush function, since it is given in a closed form. By the implicit function theorem the gradient of the implicit function is given by:

$$-\frac{(\nabla_x f_\theta(x))_\parallel}{(\nabla_x f_\theta(x))_\perp} \tag{5.19}$$

In this case $\nabla$ denotes the 3D gradient, $f_\theta$ is the network function, $\parallel$ denotes the component parallel to the tangent plane and $\perp$ the component perpendicular to it. The full expression for the normal is, thus:

$$\begin{aligned}\frac{-\nabla S_d + n}{\|-\nabla S_d + n\|} &= \frac{-\nabla S - \nabla B_{r,s} + n}{\|-\nabla S - \nabla B_{r,s} + n\|} \\ &= \frac{\frac{(\nabla_x f_\theta(x))_\parallel}{(\nabla_x f_\theta(x))_\perp} - \nabla B_{r,s} + n}{\left\|\frac{(\nabla_x f_\theta(x))_\parallel}{(\nabla_x f_\theta(x))_\perp} - \nabla B_{r,s} + n\right\|}\end{aligned} \tag{5.20}$$

## 5.5 Combining Surface and Interaction Samples

As we stated in the beginning of the chapter, our proposed method works by using both samples from the surface expressed by the network before the interaction and samples from the edited area. In the sections until now we described how to produce these samples. Now we are going to discuss how we use them to construct the final sample set for the training iteration.

Firstly, we need to remove those of the surface samples that lie inside the interaction area, since they "contradict" the edit. We do this by simply removing those that lie inside a sphere centered at the interaction point with radius equal to the brush's. Let us assume, that we initially produce $N$ surface samples and that, of those, $M$ are removed.

Secondly, we need to specify the number of interaction samples we take. For the choice of this number we need to take into consideration balancing between the two kinds of samples. Essentially, the numbers of samples will affect the distribution $p_S$ used in the loss function (equation 5.2). Notice that, the balancing is not done explicitly through the use of weights, but rather through the ratios of the two kinds of samples. Too many interaction samples and the surface samples will not manage to regulate the edit, leading to greater distortion outside the interaction area. Too few and the edit will not happen in the intended way. We would like the number of interaction samples to increase with the interaction area. Although we cannot compute this area analytically, due to our surface sampling algorithm producing relatively uniform samples, the ratio $M/N$ is probabilistically proportional to the area. Consequently, we can use $M$ to scale the number of interaction samples. Additionally, we multiply $M$ by a factor (we choose 10 in practice) to get the number of samples, in order to increase their importance in the loss function, since they are responsible for the surface deformation.

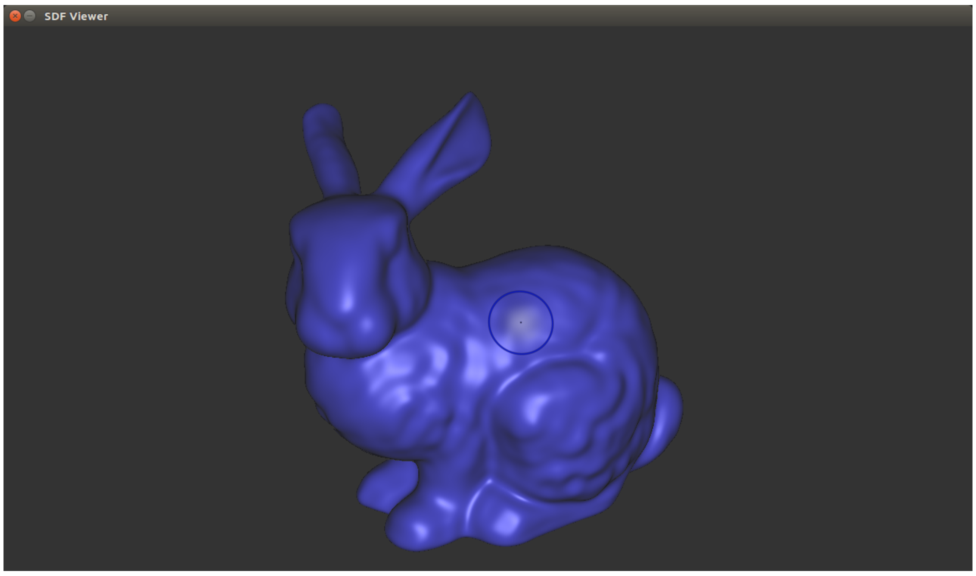


Figure 5.4: A scene from our interactive editor. The blue circle is the brush's indicator.

## 5.6 Method Overview

In the preceding sections, we presented each aspect of our proposed method in detail. Now, we give a brief overview of the whole process:

- The user specifies the interaction point on the surface along with the brush parameters.
- A copy of the network is made to be used for surface sampling, the sample set is initialized and some burnout iterations are run in order to approximate the stationary distribution.
- For each training iteration:
  - The surface is sampled (using the copy of the network at its initial unaltered state).
  - From these surface samples those that lie inside the interaction are discarded.

  - The interaction, i.e. the desired deformation, is sampled. The number of samples to be produced is calculated according to the number of surface samples discarded.
  - The two sample sets are combined.
  - The loss and its gradient with respect to the network parameters are are computed (here the original network is used, not the copy).
  - The weights are updated.

In the case of sequential edits we do not reinitialize the sample set for the surface sampling before each edit. Instead, after each edit we copy the updated network again and use that to project the samples. We run the burnout iterations in this case as well. A scene from our interactive editor is shown in figure 5.4. The brush's position and radius is depicted with a blue circle. For a video showcasing the editor please refer to `https://pettza.github.io/3DNS/`.

# Chapter 6

# Experiments

## Contents

In this chapter, we conduct a series of experiments in order to demonstrate the properties of our sampling algorithm and the capabilities of our method, as well as prove its advantages over traditional mesh editing. For our experiments we assembled a small dataset, which we present in Section 6.1 and for quantitative comparisons we use Chamfer distance, which we define in Section 6.2. As we mentioned in the previous chapter, our method is architecture-independent; for our purposes here, we employ the SIREN [64] architecture, which we discuss in section 6.3. The only alteration to the original design is the addition of Weigh Normalization [60] at each layer. We perform an ablation study of this in section 6.5. Afterward, in section 6.6 we evaluate our sampling algorithm and compare with the naive approach of reinitializing the set of surface samples. Then, we show the effect of a brush for various values of its parameters, in section 6.7. Later, in section 6.8 we compare the editing capabilities of our method to those of traditional mesh sculpting. Finally, in section 6.9, we use our method to do multiple edits on models and showcase, thus, the artistic results that can be achieved.

## 6.1 Datasets

Sculpting is mostly used to create natural looking objects, like animals, human faces, and plants. Most brushes are designed to deform the surface smoothly, or at least in manners that treat the surface as having plasticity, since their purpose is to simulate actions that are possible by their physical counterparts. If the object being modeled is man-made or otherwise has hard edges and corners there exist more appropriate tools.

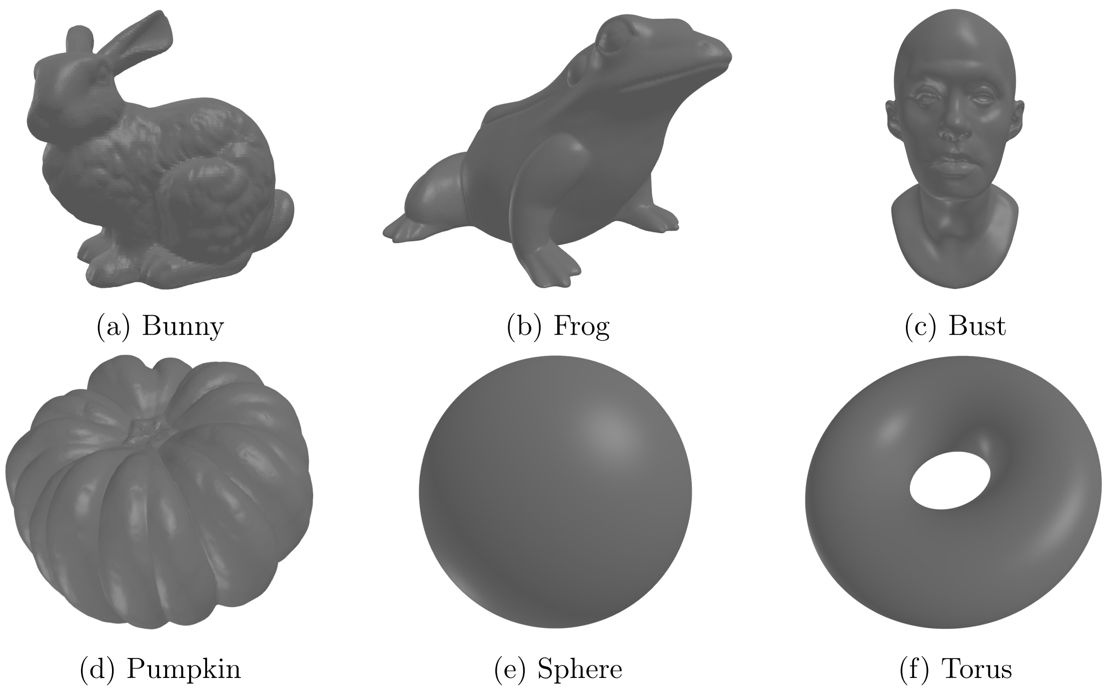
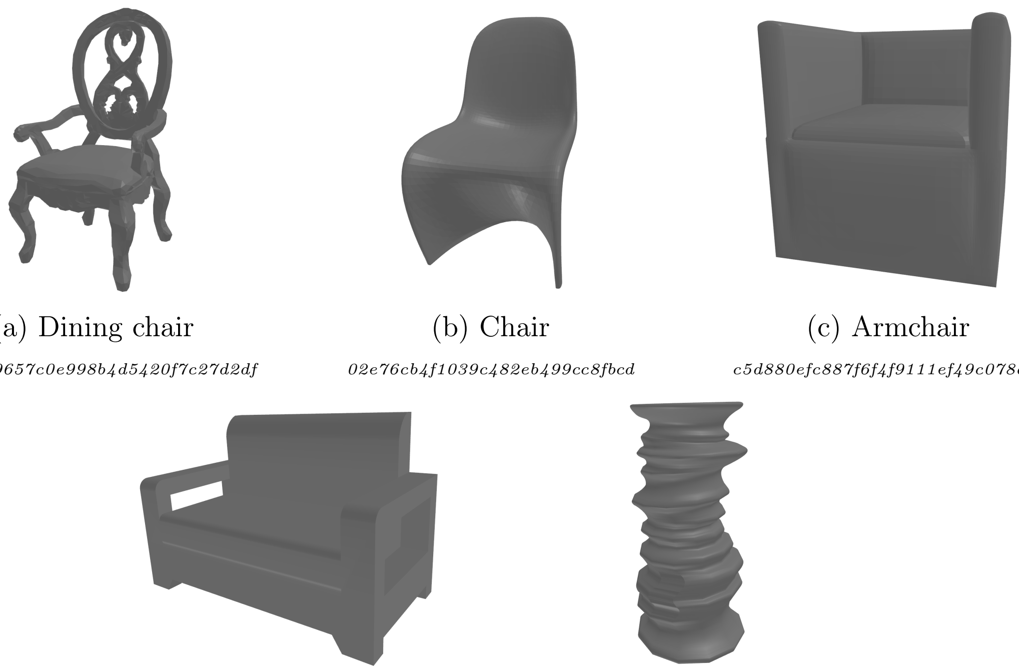

(a) Bunny (b) Frog (c) Bust
(d) Pumpkin (e) Sphere (f) Torus

Figure 6.1: The dataset of four selected meshes and two analytical shapes that we assembled for our experiments.



(a) Dining chair *4dd46b9657c0e998b4d5420f7c27d2df*
(b) Chair *02e76cb4f1039c482eb499cc8fbcd*
(c) Armchair *c5d880efc887f6f4f9111ef49c078dbe*
(d) Sofa *bcff6c5cb4127aa15e0ae65e074d3ee1*
(e) Vase *13375f8fce3142e6597d391ab6fcc1*

Figure 6.2: The ShapeNet meshes used as additional shapes for the mesh editing comparison. Beneath each caption the ShapeNet model ID of the corresponding shape is written in italics.

Nevertheless, sculpting can also be used to add surface details, like dents and scratches, to the latter objects.

Taking the above into consideration, we collected four meshes of natural objects to experiment with. The first is the, ever popular for graphics, Stanford Bunny, which comes from [72]. The other three we selected from TurboSquid [71]. They are meshes of a frog, a bust, and a pumpkin. Besides modeling natural surfaces we chose them, we took into account the vertex count and the fact that the surfaces are closed. The latter is required in order for an SDF to be applicable, although small holes do not pose a problem an get filled through training. As for the vertex count we need it to be adequate to express the desired deformation of the surface, because we use these meshes for the mesh editing comparison. We also use two shapes, a sphere and a torus, that are described analytically. Specifically, we use a sphere with a radius of 0.6 and a torus with a major radius of 0.45 and a minor radius of 0.25. These six shapes are shown in figure 6.1.

For the mesh editing comparison experiment (section 6.8) we extend the above collection with five shapes from ShapeNet [15], in order to also include some shapes that have widespread usage in the community. We note, however, ShapeNet consists of meshes of man-made objects, which typically would not be created via 3D sculpting. These additional shapes are depicted in figure 6.2.

As a pre-processing step for the meshes, we normalize them by translating and scaling them uniformly so that they lie inside $[-(1-b), 1-b]^3$, where $b$ is a positive number.

This parameter is used so that there is space around the models where we can edit them. We set $b$ to 0.15.

## 6.2 Performance Metric

For quantitative comparisons, we use the Chamfer distance, which is a common metric used for geometric tasks. In particular, it has been employed in many works on Neural SDFs [31, 50, 67]. Chamfer distance is used to indicate the similarity between two point clouds and is defined as follows:

**Definition 6.1: Chamfer Distance**

If $A$, $B$ are two point clouds, then their Chamfer distance is:

$$CD(A,B) = \frac{1}{|A|}\sum_{a\in A}\min_{b\in B} d(a,b) + \frac{1}{|B|}\sum_{b\in B}\min_{a\in A} d(a,b)$$

*Remark.* If $CD(A,B) = 0$, then either $A \subseteq B$ or $B \subseteq A$.

Despite Chamfer distance being defined for point clouds, it can be used to compare surfaces, as well, by sampling the surfaces and comparing the resulting point clouds instead. In this case, however, the result is not deterministic, due to the sampling. Other than that, the number of samples used, also affects the result. The mean value of the distance decreases when the number of samples increases, since the *min* inside the sum is approximated more effectively. This is why when Chamfer distance is used the number of samples used is also mentioned. Below, when we use Chamfer distance to compare between two meshes or between a mesh and a neural SDF. We sample meshes uniformly as described in section 3.2.2 and neural SDFs using our sampling algorithm.

## 6.3 Network Architecture

During the exposition of our method in the previous chapter we made no mention of specific neural network architecture, because, since the method concerns the construction of the set of training samples, it can be used with a variety of architectures. Nevertheless, for experimenting we do require a concrete architecture, of course, and for this we choose SIREN [64], which we extended with weight normalization [60].

### 6.3.1 SIREN

We presented this architecture in chapter 4. We give a reprise here for convenience's sake. SIREN is simply a multilayer perceptron that uses sines instead of the usual ReLUs as activation functions. Evidently, sines allow the network to more accurately represent finer details. The use of sines is related to Fourier features [7], for which the aforementioned property has been theoretically explained [69] using the NTK framework [38].

Despite its simplicity, it achieved impressive results at a multitude of tasks. Specifically, the authors experiment with audio, image, video, and surface representation, and solving various differential equations. The authors also provide a weight initialization scheme suited for sine activations. We choose this architecture for our experiments due to its simplicity and effectiveness.

### 6.3.2 Weight Normalization

This is a general technique that aims to accelerate convergence during training [60]. The idea is to take better optimization steps during gradient descent by reparametrizing the neural network. More specifically a neuron's weights get represented by a direction vector and a length parameter. This approximately *whitens* the gradients. Relative to other techniques which follow more direct approaches toward this goal like KFAC [45] and Batch Normalization [37], the latter of which is an inspiration for this technique, Weight Normalization is computationally cheaper and more memory efficient. We apply Weight Normalization to each layer of SIREN and show, in section 6.5, that it indeed has a positive effect.

## 6.4 Training Details

For our implementation we use PyTorch [51]. We set the values of the hyperparameters $\lambda_1$, $\lambda_2$, $\lambda_3$, and $\lambda_4$ to $1.5 \cdot 10^3$, 5, 2.5, and 5 respectively, and $a$ to 100. We train a SIREN network with 2 hidden layers, width 128, and Weight Normalization (except in section 6.5) for each shape for $10^6$ iterations using the Adam optimizer [40] with a learning rate of $10^{-4}$. For each iteration we use 120000 points sampled on the the surface of the target shape for the loss term of equation 5.2 and another 120000 points sampled uniformly inside the bounding box $[-1.15, 1.15]^3$ for the ones of equations 5.3 and 5.4. We use a bounding box larger than $[-1, 1]^3$ because it helps our sampling algorithm. We sample the meshes, the sphere, and the torus uniformly using the techniques presented in section 3.2.2.

## 6.5 Weight Normalization ablation

Our first experiment is an ablation study that proves the improvements provided by Weight Normalization. We train two networks for each of the six shapes of figure 6.1, one with Weight Normalization and one without. The Chamfer distance between a point cloud sampled from the ground truth models and one sampled from the trained network by our sampling algorithm is given in Table 6.1. As can be seen, even though not by a large margin, the models trained with weight normalization perform better in every case.

| Shape | Chamfer Distance $\times 10^3$ ($\downarrow$) | |
|---|---|---|
| | Without WN | With WN |
| Bunny | 9.021 | **8.995** |
| Frog | 8.095 | **7.921** |
| Bust | 7.167 | **7.139** |
| Pumpkin | 8.646 | **8.506** |
| Sphere | 7.087 | **6.861** |
| Torus | 7.198 | **6.882** |
| *Average* | 7.869 | **7.717** |

Table 6.1: Chamfer distances computed with 100000 points for Weight Normalization ablation.

## 6.6 Sampling Algorithm Evaluation

We want to study the uniformness of the stationary distribution of our sampling algorithm. Since the support of this distribution is a complex 3D surface, instead of a plane for example, the complexity is shared by the task as well. Our solution is to estimate the probability density function (pdf) at multiple points over the surface. More accurately, what we aim to estimate is the mean value of the pdf over separate regions of the surface. If the regions are relatively small and the estimated mean values are close to the pdf value of a true uniform distribution, which is the inverse of the surface area, then the stationary distribution is, too, quite uniform.

Let's consider that the surface is partitioned into mutually exclusive patches/regions. The probability of a sample belonging in a patch can be approximated by taking multiple samples with our algorithm and calculating the frequency of those that fall inside the patch. In other words:

$$\mathbb{P}(x \in D) \approx \frac{c_D}{N} \tag{6.1}$$

where $x$ is random point following the stationary distribution, $D$ is a patch, $c_D$ is the number of samples that fell in the patch, and $N$ it the total number of samples. Essentially, this is is an estimator of a Bernoulli random variable. When $N$ goes to infinity the frequency converges to the probability on the left hand side.

The above probability relates to the pdf through the following equation:

$$\mathbb{P}(x \in D) = \int_D p_s(x)\, dx \tag{6.2}$$

where $p_s$ is the stationary distribution. Therefore the mean value of the pdf can be approximated as:

$$\frac{1}{A_D}\int_D p_s(x)\, dx = \frac{\mathbb{P}(x \in D)}{A_D} \approx \frac{c_D}{N \cdot A_D} \tag{6.3}$$

where $A_D$ is the area of the patch $D$. If $D$ shrinks to a limiting point with its area $A_D$ going to 0 than the mean value approaches the pdf value at that point. This, and what was mentioned above, is why we require small enough patches and an adequate number samples to get a good estimation. However, if we consider the way our algorithm works we can see that the patches need not be extremely small. A point results from its previous with the addition of random tangent vector (and reprojection). We can see the resulting distribution as a generalized convolution on the surface between the initial distribution and the distribution of the tangent vector. This has a smoothing action on the the initial distribution and, consequently, the stationary distribution changes gradually.

This approach, as described thus far, poses a problem. The patches cannot be represented exactly. We address this by adding another layer of approximation. We create a proxy triangle mesh of the surface from the neural SDF using the Marching Cubes algorithm [43] and use its triangles instead of the patches. The Marching Cube algorithm works by discretizing the space using a regular grid and, so, the approximation will not be particularly accurate in areas where the surface has high curvature. Then, we generate samples with our algorithm for $N$ iterations, with $M$ samples per iteration (so $N \cdot M$ total samples). In effect, we are simulating $M$ Markov chains. For each triangle of the mesh, we count the total number of samples $c$ that are closest to it. According to what we discussed above, if $A$ is the area of the triangle, then the estimated mean value of the pdf over the triangle is:

$$pdf = \frac{c}{N \cdot M \cdot A} \tag{6.4}$$

We visualize the estimated pdf for the six shapes of figure 6.1 with the face colors in figure 6.3. In the same figure, we show histograms of the pdf estimations, as well. The value of the uniform pdf, which is equal to the inverse of the surface area, is shown in the histograms with a dashed vertical red line. We can see that the histograms are centered tightly around this value, indicating that the stationary distributions of our sampling algorithm are quite uniform. For comparison, we give the corresponding results for the naive sampling, i.e. reinitializing the sample set at each iteration, in figure 6.4. Here, we notice the bright areas which are sampled more frequently than the rest of the surface, as well as the longer tails of the histograms and the fact that they are off-centered.

## 6.7 Brush Parameters

We demonstrate how the brush parameters allow the user to control the edit in figures 6.5 and 6.6. Specifically, we use different radii and intensities on the same interaction point on the sphere shape. The intensity values for the former figure are positive, and for the latter negative, showcasing how both bump/extrusions and dents/intrusions are possible. In each figure we use the same intensity across each row, whose value is shown on the left, and the same radius across each column, whose value is shown on the top. We can see that by adapting the number of interaction samples to the radius of the brush, as we explained in section 5.4, the desired deformation is achieved for wide range of radii. More

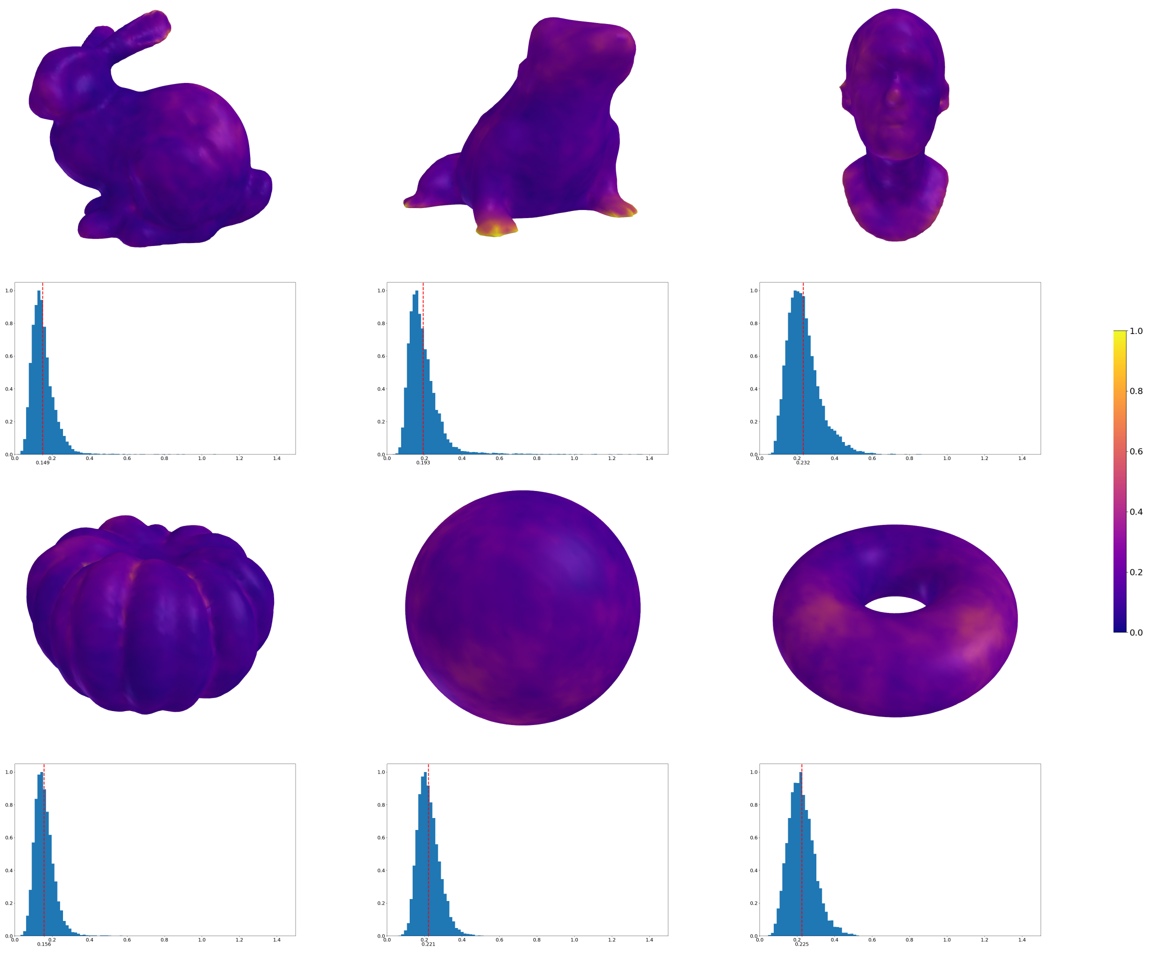


Figure 6.3: The results of estimating the probability density function of our proposed sampling algorithm.

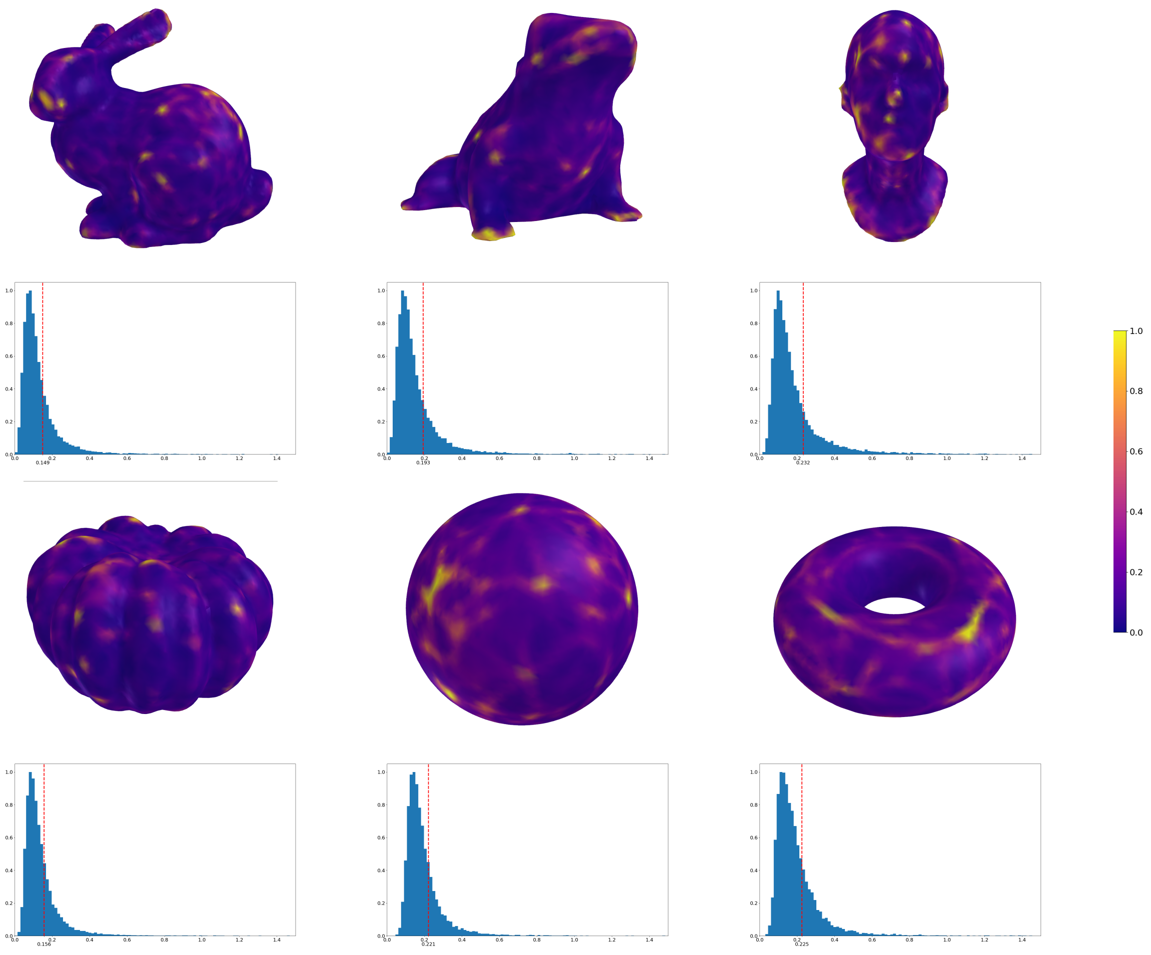


Figure 6.4: The results of estimating the probability density function of the naive sampling algorithm.

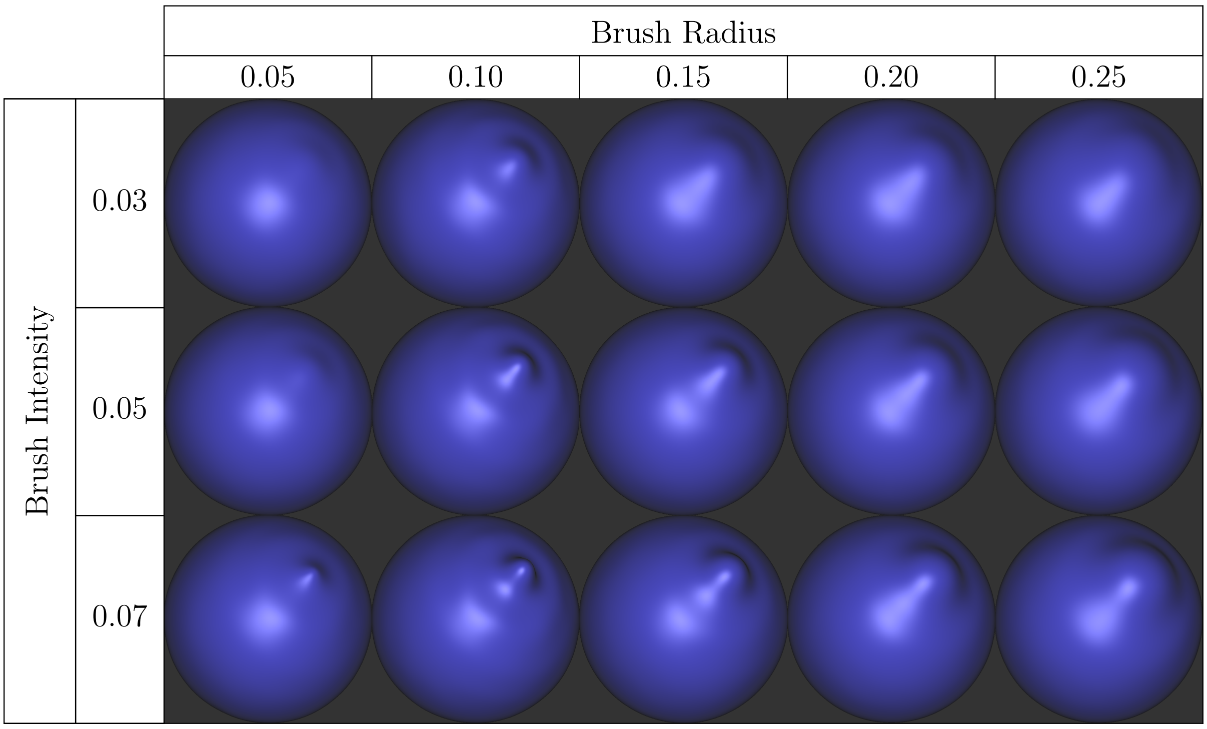


Figure 6.5: The effect of a brush applied at the same interaction point on a sphere with different values for the radius and intensity. The intensities are positive and so the brush creates bumps/extrusions.

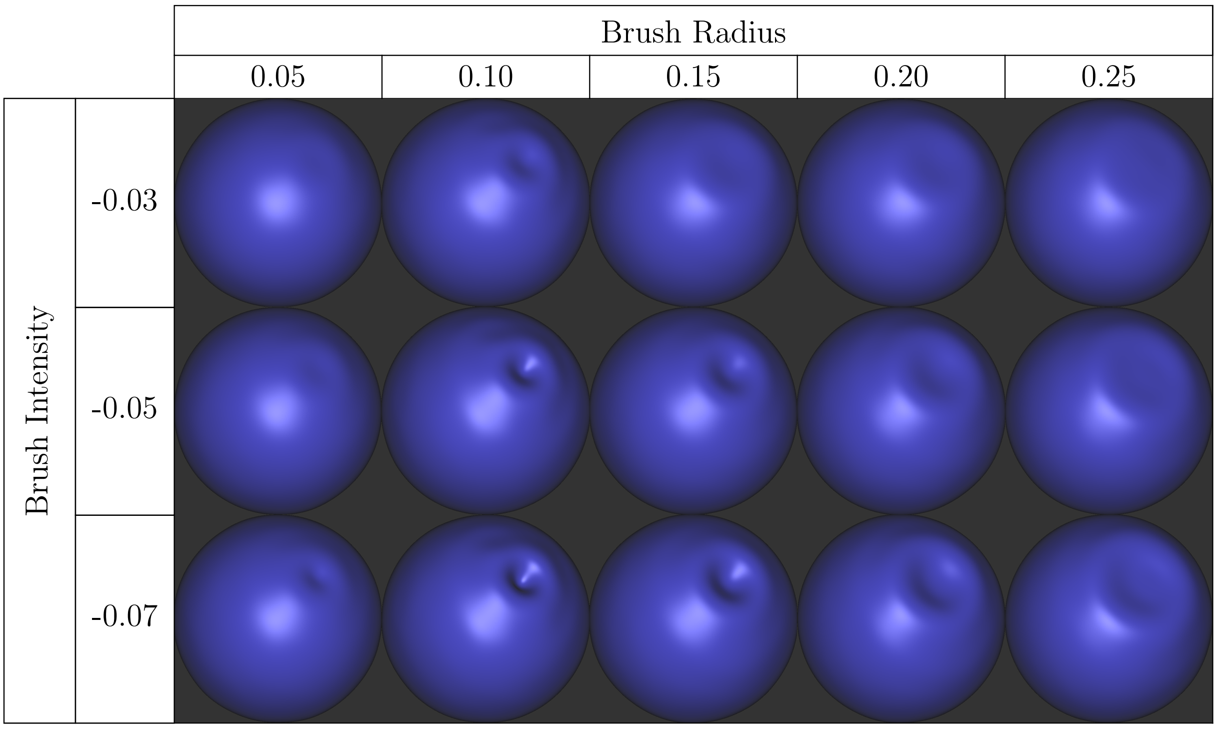


Figure 6.6: The same setting as in figure 6.5, but with negative values for the intensity, creating dents/intrusions.

complex editing is possible through multiple interactions with varying brush parameters. We show such examples later in section 6.9.

## 6.8 Mesh Editing Comparison

As we have mentioned again in this thesis, by far the most popular surface representations for 3D modeling and sculpting are meshes. In order to evaluate our method, consequently, we have to make some comparison with mesh-based sculpting. However, a direct quantitative comparison with commercial 3D software is not possible because the brushes used there work differently to ours. Accordingly, we implement a mesh editing in a manner that mimics the action of a brush on a neural SDF.

When sampling the interaction in order to apply the brush on the surface expressed by a neural SDF, we projected samples from the tangent disk onto the surface. Here, since the vertices already lie on the surface represented by the mesh, we need to follow the inverse procedure in order to compute their displacement along the interaction normal. The vertices whose positions we need to edit are those that lie inside a sphere centered at the interaction point with radius equal to the brush's radius. For each of them, we promptly find the intersection of the ray starting at the vertex with direction along its normal and the plane tangent at the interaction point. The 2D position of that intersection point on the tangent plane gives the value of the brush function corresponding to the vertex. The deformed normals are computed similarly, as we have described.

Besides the above, it is not clear how such different approaches to surface representation can be compared, in the first place. We opt for a comparison of the expressive capabilities of the two on a fixed memory budget. We use a high resolution mesh as ground truth and a low resolution one as baseline. The four meshes of figure 6.1 are detailed enough to be used as ground truth. The ShapeNet shapes (figure 6.2), however, are not and, so, for them we construct the ground truth mesh form the network using Marching Cubes [43]. We, also, do this for the sphere and the torus. Following NGLoD [67], we construct the baseline mesh using quadratic decimation [30], which simplifies a

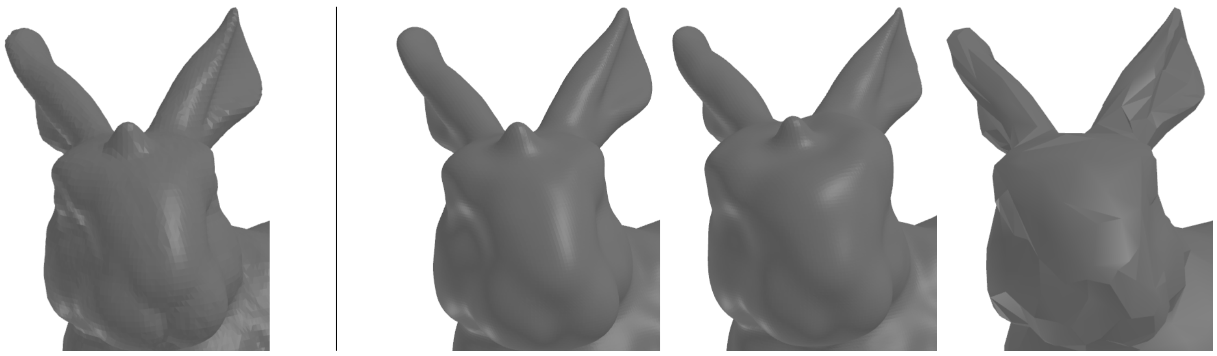


Figure 6.7: Example of an edit using different methods. From left to right, ground truth mesh (ideal edit result), ours, naive and baseline (simplified) mesh.

| Shape | Mean Chamfer Distance $\times 10^3$ ($\downarrow$) | | | | | |
|---|---|---|---|---|---|---|
| | Over whole surface | | | Inside interaction area | | |
| | Ours | Naive | Simple Mesh | Ours | Naive | Simple Mesh |
| Bunny | **9.407** | 14.106 | 11.127 | **5.527** | 12.919 | 17.707 |
| Frog | **8.172** | 12.865 | 8.756 | **4.750** | 9.805 | 17.051 |
| Bust | **7.279** | 11.486 | 7.901 | **3.818** | 8.779 | 14.926 |
| Pumpkin | **8.774** | 13.558 | 11.489 | **4.315** | 5.910 | 20.693 |
| Sphere | **7.209** | 12.550 | 7.399 | **3.555** | 6.117 | 12.982 |
| Torus | **7.142** | 13.516 | 7.415 | **3.574** | 5.980 | 13.402 |
| Dining chair | **7.256** | 20.545 | 8.260 | **8.843** | 11.288 | 10.703 |
| Chair | **8.498** | 30.620 | 8.650 | **8.741** | 19.072 | 15.344 |
| Armchair | **14.152** | 19.433 | 14.311 | **5.085** | 10.942 | 23.654 |
| Sofa | **11.160** | 18.268 | 11.899 | **4.477** | 8.623 | 23.940 |
| Vase | **9.063** | 15.565 | 10.345 | **10.477** | 15.713 | 16.725 |
| *Average* | **8.919** | 16.592 | 9.778 | **5.742** | 10.468 | 17.012 |

Table 6.2: Comparison of our editing method with and without model samples (Ours and Naive, respectively) and direct mesh editing on a mesh with equivalent size (Simple Mesh). Chamfer distances are computed with 100000 points. The mean for each shape is taken over 10 independent edits.

mesh given a budget on its faces. We perform a binary search to find the least number of faces such that the mesh's memory footprint is greater or equal to the networks.

Having defined the ground truth and the baseline meshes the we perform the same ten independent edits on them and the neural SDF. We edit the neural SDF using our method, as well as the naive approach of using only interaction samples (i.e. no surface samples). By *independent edits* we mean that each edit is done on the unedited representations. The interaction point for each edit is randomly chosen on the surface. The brush parameters are set to 0.08 for the radius and to 0.06 for the intensity for all of them, though. The neural SDFs are edited using 100 iterations. We compute the mean Chamfer distance (using 100000 samples) between the neural SDFs and the ground truth mesh, and between the simplified mesh and the ground truth mesh, over the whole surface, as well as only inside the interaction areas. The results are summarized in table 6.2, where it is shown that our method outperforms the other approaches. We, also, provide a visual example of a single edit in figure 6.7.

## 6.9 Multiple Edits Examples

In this section, we provide three examples of what can be achieved through multiple applications of the brush at different points and with different values for the parameters, showcasing in this way the artistic possibilities. These edits were done using our interactive editor, which renders the neural SDF using Sphere Tracing [35], and allows the user to choose the interaction point using the cursor. For a single edit the network is trained for

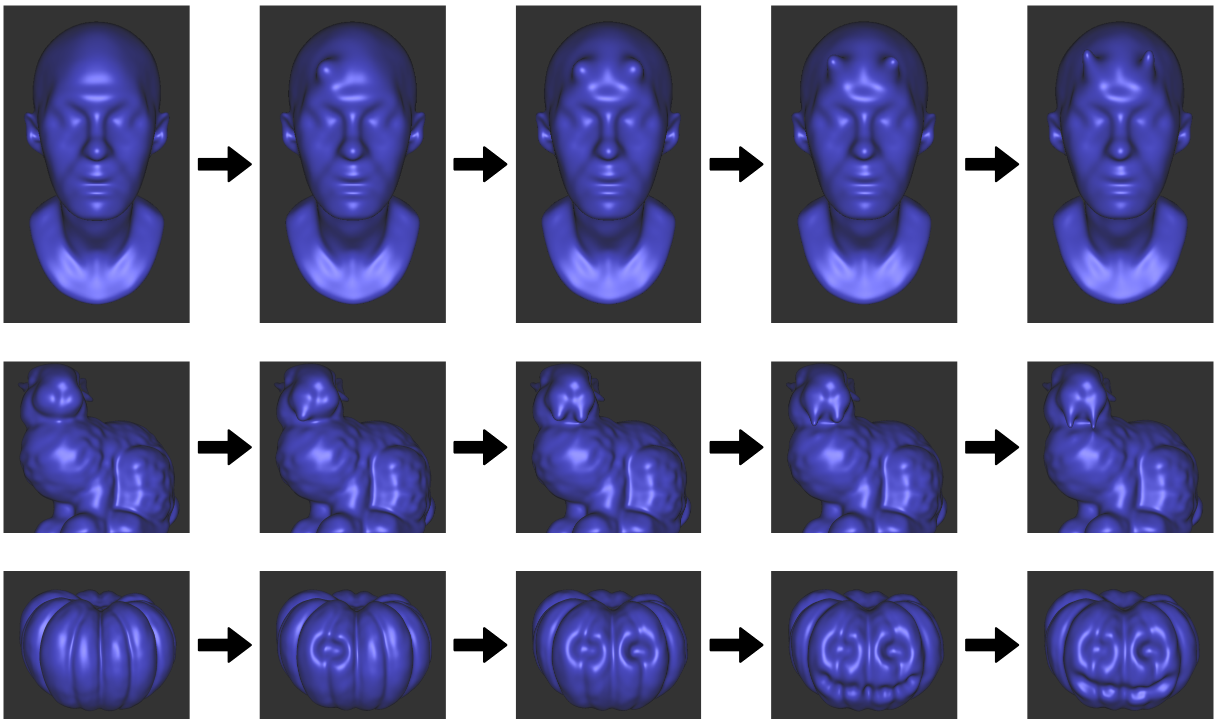


Figure 6.8: Examples of shape editing capabilities offered by our method, acting directly on the Neural SDF representation. First column: original shapes. Following columns: results of multiple edits using various brush settings, visualizing intermediate stages of the process. All edits and renderings are done on the implicit neural representation, avoiding completely the use of triangular meshes. Images were taken from our interactive editor, which uses Sphere Tracing [35] to render the zero-level set of the Neural SDFs.

100 iterations. The examples are depicted using renderings of intermediate stages of the the editing process.

# Chapter 7

# Going the Distance

*That's the place for us, Hazel. High, lonely hills, where the wind and the sound carry and the ground's as dry as straw in a barn. That's where we ought to be. That's where we have to get to.*

Richard Adams, *Watership Down*

In this thesis we ventured into unknown territory and hopefully have come out wiser. More than that, we hope to have made a fruitful contribution to the exciting and up-and-coming field of Neural SDFs. Our proposed method is the first, to our knowledge, to implement sculpting-like capabilities for these representations. We commenced by stating the problem of particular concern arising from the use of neural networks, which is the locality of the modification to the surface. Unless explicitly handled during training, the network will not preserve the shape outside the intended edit. Our method is tailored around this challenge. With our experiments we showed that our solution of using samples produced using a copy of the network in order to regulate its training produces superior results relative to the corresponding techniques for meshes, at least as far as memory efficiency is concerned. Even though, our experimental results were based on a single neural network architecture, the generality of our method make it applicable to almost any other architecture, without doubt for its performance, since the method concerns itself with the construction of the training set rather than the network structure.

Along with the above, there's our extension to existing sampling algorithms for Neural SDFs. In essence, it is a Markov Chain sampling algorithm. Looking at it through this mathematical prism, allows one to utilize the relevant theorems for analyzing it. Aside from that, we also proved practically its advantage over the algorithms we based it on, which is its ability to sample the surface much more uniformly.

It is our strong conviction that research on the editability of neural representations is key towards their adoption in mainstream applications and finding their footing as equals to the established representations. Particularly, our aspiration is for Neural SDFs to be incorporated in 3D sculpting software as well as scientific environments. There already exist works on robotics that avail of them. To that purpose, of course, more effort is required. In previous chapters we highlighted areas of interest that future work can focus on. Specifically:

- A more thorough mathematical examination of the sampling algorithm properties and how different strategies of perturbing the samples affect the distribution.
- Extending the brush formulation can provide tools for more complex editing
- Employment of more recent architectures. Hybrid architectures use spatial data structures that might help with the problem of locality.

Improving the present arsenal of tools and widening this collection is the only way of making the signed distance go the distance.

# Bibliography

[1] Panos Achlioptas et al. *Learning Representations and Generative Models for 3D Point Clouds*. 2018. arXiv: 1707.02392 [cs.CV] (Cited on page 61).

[2] Brian Amberg, Sami Romdhani, and Thomas Vetter. "Optimal Step Nonrigid ICP Algorithms for Surface Registration". In: *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2007. DOI: 10.1109/CVPR.2007.383165 (Cited on page 61).

[3] Matan Atzmon and Yaron Lipman. *SAL: Sign Agnostic Learning of Shapes from Raw Data*. 2020. arXiv: 1911.10414 [cs.CV] (Cited on pages 6, 66, 72).

[4] Matan Atzmon and Yaron Lipman. *SALD: Sign Agnostic Learning with Derivatives*. 2020. arXiv: 2006.05400 [cs.CV] (Cited on pages 6, 69, 72).

[5] Matan Atzmon et al. *Controlling Neural Level Sets*. 2019. arXiv: 1905.11911 [cs.LG] (Cited on pages 8, 67, 72, 79, 80).

[6] Adi Ben-Israel. "A Newton-Raphson Method for the Solution of Systems of Equations". In: *Journal of Mathematical Analysis and Applications* 15.2 (1966), pp. 243–252. ISSN: 0022-247X. DOI: https://doi.org/10.1016/0022-247X(66)90115-6. URL: https://www.sciencedirect.com/science/article/pii/0022247X66901156 (Cited on page 72).

[7] Nuri Benbarka et al. *Seeing Implicit Neural Representations as Fourier Series*. 2021. arXiv: 2109.00249 (Cited on page 93).

[8] Fausto Bernardini et al. "The ball-pivoting algorithm for surface reconstruction". English (US). In: *IEEE Transactions on Visualization and Computer Graphics* 5.4 (1999), pp. 349–359. ISSN: 1077-2626. DOI: 10.1109/2945.817351 (Cited on page 73).

[9] Volker Blanz and Thomas Vetter. "A Morphable Model for the Synthesis of 3D Faces". In: *The Conference on Computer Graphics and Interactive Techniques (SIGGRAPH*. 1999 (Cited on page 2).

[10] *Blender*. URL: https://www.blender.org (Cited on page 82).

[11] James Booth et al. "Large Scale 3D Morphable Models". In: *International Journal of Computer Vision* 126 (2018). DOI: https://doi.org/10.1007/s11263-017-1009-7 (Cited on pages 2, 61, 62).

[12] Michael M. Bronstein et al. "Geometric Deep Learning: Going beyond Euclidean data". In: *IEEE Signal Processing Magazine* 34.4 (2017), pp. 18–42. ISSN: 1558-0792. DOI: 10.1109/msp.2017.2693418. URL: http://dx.doi.org/10.1109/MSP.2017.2693418 (Cited on page 62).

[13] J. C. Carr et al. "Reconstruction and Representation of 3D Objects with Radial Basis Functions". In: *The Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*. 2001 (Cited on pages 2, 5, 65).

[14] Rohan Chabra et al. *Deep Local Shapes: Learning Local SDF Priors for Detailed 3D Reconstruction*. 2020. arXiv: 2003.10983 [cs.CV] (Cited on pages 7, 71, 73).

[15] Angel X. Chang et al. *ShapeNet: An Information-Rich 3D Model Repository*. 2015. arXiv: 1512.03012 [cs.GR] (Cited on pages 12, 66, 91).

[16] Zhiqin Chen and Hao Zhang. *Learning Implicit Fields for Generative Shape Modeling*. 2019. arXiv: 1812.02822 [cs.GR] (Cited on pages 2, 6, 65).

[17] Julian Chibane, Thiemo Alldieck, and Gerard Pons-Moll. *Implicit Functions in Feature Space for 3D Shape Reconstruction and Completion*. 2020. arXiv: 2003.01456 [cs.CV] (Cited on pages 65, 70).

[18] Julian Chibane, Aymen Mir, and Gerard Pons-Moll. *Neural Unsigned Distance Fields for Implicit Function Learning*. 2020. arXiv: 2010.13938 [cs.CV] (Cited on pages 8, 70, 72, 73, 79–81).

[19] Christopher B. Choy et al. *3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction*. 2016. arXiv: 1604.00449 [cs.CV] (Cited on pages 2, 63).

[20] George V. Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals and Systems* 2 (1989), pp. 303–314 (Cited on page 57).

[21] Boyang Deng et al. "Neural Articulated Shape Approximation". In: *The European Conference on Computer Vision (ECCV)*. 2020 (Cited on pages 7, 75, 78).

[22] Yu Deng, Jiaolong Yang, and Xin Tong. "Deformed Implicit Field: Modeling 3D Shapes With Learned Dense Correspondence". In: *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021 (Cited on pages 75, 78).

[23] Persi Diaconis and David Freedman. "On Markov chains with continuous state space". In: *Technical Report no. 501* (1997) (Cited on page 39).

[24] Bernhard Egger et al. *3D Morphable Face Models – Past, Present and Future*. 2020. arXiv: 1909.01815 [cs.CV] (Cited on pages 2, 62).

[25] Kenneth Falconer. *The Geometry of Fractal Sets*. Cambridge University Press, 1986 (Cited on page 27).

[26] Kenneth Falconer. *Fractal Geometry: Mathematical Foundations and Applications*. Wiley, 2003 (Cited on page 27).

[27] Herbert Federer. *Geometric Measure Theory*. Springer, 1996 (Cited on page 29).

[28] Chelsea Finn, Pieter Abbeel, and Sergey Levine. *Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks*. 2017. arXiv: 1703.03400 [cs.LG] (Cited on page 71).

[29] Robert G. Gallanger. *Stochastic Processes: Theory for Applications*. Cambridge University Press, 2014 (Cited on page 38).

[30] Michael Garland and Paul S. Heckbert. "Surface simplification using quadric error metrics". In: *The Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*. 1997 (Cited on page 99).

[31] Amos Gropp et al. *Implicit Geometric Regularization for Learning Shapes*. 2020. arXiv: 2002.10099 [cs.LG] (Cited on pages 6, 68, 69, 92).

[32] David Ha, Andrew Dai, and Quoc V. Le. *HyperNetworks*. 2016. arXiv: 1609.09106 [cs.LG] (Cited on page 71).

[33] Christian Häne, Shubham Tulsiani, and Jitendra Malik. *Hierarchical Surface Prediction for 3D Object Reconstruction*. 2017. arXiv: 1704.00710 [cs.CV] (Cited on pages 2, 63, 64).

[34] Zekun Hao et al. "DualSDF: Semantic Shape Manipulation using a Two-Level Representation". In: *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020 (Cited on pages 7, 74, 78).

[35] John C. Hart. "Sphere Tracing: A Geometric Method for the Antialiased Ray Tracing of Implicit Surfaces". In: *The Visual Computer* 12.10 (1996), pp. 527–545. ISSN: 1432-2315. DOI: 10.1007/s003710050084. URL: https://doi.org/10.1007/s003710050084 (Cited on pages 73, 100, 101).

[36] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-term Memory". In: *Neural computation* 9 (1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735 (Cited on page 51).

[37] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 (Cited on page 93).

[38] Arthur Jacot, Franck Gabriel, and Clément Hongler. *Neural Tangent Kernel: Convergence and Generalization in Neural Networks*. 2020. arXiv: 1806.07572 [cs.LG] (Cited on pages 70, 93).

[39] Korrawe Karunratanakul et al. *Grasping Field: Learning Implicit Representations for Human Grasps*. 2020. arXiv: 2008.04451 [cs.CV] (Cited on page 75).

[40] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. arXiv: 1412.6980 (Cited on page 93).

[41] Dominik Kulon et al. *Weakly-Supervised Mesh-Convolutional Hand Reconstruction in the Wild*. 2020. arXiv: 2004.01946 [cs.CV] (Cited on page 61).

[42] Shaohui Liu et al. *DIST: Rendering Deep Implicit Signed Distance Function with Differentiable Sphere Tracing*. 2020. arXiv: 1911.13225 [cs.CV] (Cited on pages 73, 74).

[43] William E. Lorensen and Harvey E. Cline. "Marching Cubes: A High Resolution 3D Surface Construction Algorithm". In: *SIGGRAPH Computer Graphics* 21.4 (1987), pp. 163–169 (Cited on pages 12, 17, 72, 95, 99).

[44] Jerrold E. Marsden and Anthony Tromba. *Vector Calculus*. W. H. Freeman, 2011 (Cited on pages 9, 41, 56, 84).

[45] James Martens and Roger Grosse. *Optimizing Neural Networks with Kronecker-factored Approximate Curvature*. 2015. arXiv: 1503.05671 (Cited on page 93).

[46] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/ (Cited on page 56).

[47] Lars Mescheder et al. *Occupancy Networks: Learning 3D Reconstruction in Function Space*. 2019. arXiv: 1812.03828 [cs.CV] (Cited on pages 2, 6, 65).

[48] Jiteng Mu et al. *A-SDF: Learning Disentangled Signed Distance Functions for Articulated Shape Representation*. 2021. arXiv: 2104.07645 [cs.CV] (Cited on pages 7, 75, 78).

[49] Thomas Müller et al. "Instant Neural Graphics Primitives with a Multiresolution Hash Encoding". In: *ACM Trans. Graph.* 41.4 (2022), 102:1–102:15. DOI: 10.1145/3528223.3530127. URL: https://doi.org/10.1145/3528223.3530127 (Cited on pages 7, 71, 73, 74).

[50] Jeong Joon Park et al. "DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation". In: *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019 (Cited on pages 2, 6, 65–67, 71, 72, 92).

[51] Adam Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019. arXiv: 1912.01703 (Cited on pages 12, 46, 56, 93).

[52] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering: From theory to implementation*. Morgan Kaufmann, 2016 (Cited on pages 32, 47).

[53] Charles R. Qi et al. *PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation*. 2017. arXiv: `1612.00593 [cs.CV]` (Cited on pages 60, 61).

[54] Inigo Quilez. *FBM detail in SDFs*. 2022. URL: `https://iquilezles.org/articles/fbmsdf/` (Cited on pages 2, 20).

[55] Inigo Quilez. *Inigo Quilez's site*. URL: `https://iquilezles.org` (Cited on pages 65, 84).

[56] Martin Roberts. *How to generate uniformly random points on n-spheres and n-balls*. URL: `http://extremelearning.com.au/how-to-generate-uniformly-random-points-on-n-spheres-and-n-balls/` (Cited on page 46).

[57] Sheldon Ross. *Introduction to Probability Models*. Elsevier, 2017 (Cited on page 38).

[58] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323 (1986), pp. 533–536 (Cited on page 56).

[59] David E. Rumelhart and James L. McClelland. "Learning Internal Representations by Error Propagation". In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*. 1987, pp. 318–362 (Cited on page 51).

[60] Tim Salimans and Diederik P. Kingma. *Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks*. 2016. arXiv: `1602.07868 [cs.LG]` (Cited on pages 12, 66, 90, 92, 93).

[61] Jürgen Schmidhuber. "Deep learning in neural networks: An overview". In: *Neural Networks* 61 (2015), pp. 85–117 (Cited on page 52).

[62] *Shadertoy*. URL: `https://www.shadertoy.com` (Cited on page 65).

[63] Justin Sirignano and Konstantinos Spiliopoulos. "DGM: A deep learning algorithm for solving partial differential equations". In: *Journal of Computational Physics* 375 (2018), pp. 1339–1364. ISSN: 0021-9991. DOI: `10.1016/j.jcp.2018.08.029`. URL: `http://dx.doi.org/10.1016/j.jcp.2018.08.029` (Cited on pages 6, 69).

[64] Vincent Sitzmann et al. "Implicit Neural Representations with Periodic Activation Functions". In: *The Conference on Neural Information Processing Systems*. 2020 (Cited on pages 6, 7, 69–72, 78, 90, 92).

[65] Vincent Sitzmann et al. *MetaSDF: Meta-learning Signed Distance Functions*. 2020. arXiv: `2006.09662 [cs.CV]` (Cited on pages 71, 72).

[66] Edgar Sucar et al. "iMAP: Implicit Mapping and Positioning in Real-Time". In: *The International Conference on Computer Vision (ICCV)*. 2021 (Cited on page 75).

[67] Towaki Takikawa et al. "Neural Geometric Level of Detail: Real-time Rendering with Implicit 3D Shapes". In: *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021 (Cited on pages 7, 17, 63, 71, 73, 74, 92, 99).

[68] Shufeng Tan and Michael L. Mavrovouniotis. “Reducing Data Dimensionality through Optimizing Neural Network Inputs”. In: *Aiche Journal* 41 (1995), pp. 1471–1480 (Cited on page 66).

[69] Matthew Tancik et al. *Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains*. 2020. arXiv: 2006.10739 [cs.CV] (Cited on pages 70, 93).

[70] Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. *Octree Generating Networks: Efficient Convolutional Architectures for High-resolution 3D Outputs*. 2017. arXiv: 1703.09438 [cs.CV] (Cited on pages 2, 63–65).

[71] *TurboSquid*. URL: https://www.turbosquid.com (Cited on pages 12, 91).

[72] Greg Turk and Marc Levoy. “Zippered Polygon Meshes from Range Images”. In: *The Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*. 1994 (Cited on pages 12, 91).

[73] Petros Tzathas, Petros Maragos, and Anastasios Roussos. “3D Neural Sculpting (3DNS): Editing Neural Signed Distance Functions”. In: *The IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. 2023 (Cited on page 22).

[74] Aaron R. Voelker, Jan Gosmann, and Terrence C. Stewart. *Efficiently sampling vectors and coordinates from the n-sphere and n-ball*. Tech. rep. Waterloo, ON: Centre for Theoretical Neuroscience, 2017. DOI: 10.13140/RG.2.2.15829.01767/1 (Cited on page 45).

[75] Yiheng Xie et al. *Neural Fields in Visual Computing and Beyond*. 2021. arXiv: 2111.11426. URL: https://neuralfields.cs.brown.edu/ (Cited on page 75).

[76] M. Zollhöfer et al. “State of the Art on Monocular 3D Face Reconstruction, Tracking, and Applications”. In: *Computer Graphics Forum (Eurographics State of the Art Reports 2018)* 37.2 (2018) (Cited on page 62).