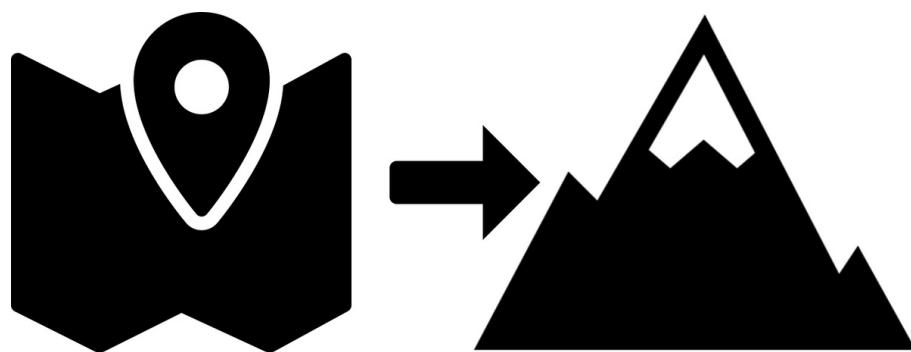


olto3d

Automatische Digitalisierung von OL-Karten



Flavian Kaufmann

1. Februar 2019

Maturaarbeit

Betreuungsperson:

Beni Keller

Kantonsschule Zug

2019

Inhaltsverzeichnis

1 Einleitung	3
2 Computer Vision	4
2.1 OpenCV	4
2.2 Anwendungen	5
2.2.1 Bilder entschärfen (Image smoothing)	5
2.2.2 Kantenerkennung (Edge detection)	6
3 3D	8
3.1 Transformationen	8
4 Resultate	9
4.1 Probleme	9
4.1.1 Isolinien Erkennung	9
4.2 Diskussion	12
4.3 Wie weiter?	12

1 Einleitung

Die Digitalisierung, der Prozess welcher analoge Daten ins digitale Format bringt, gibt es schon lange, wahrscheinlich seit der Geburt der Computer. Dies ist jedoch ein mühsamer Vorgang, beansprucht viel Zeit und muss teilweise auch von Hand bewältigt werden. Um die Digitalisierung zu beschleunigen, gibt es nur eine Lösung - Automation. Für viele Anwendungen gibt es die auch schon. Was jeder kennt ist die Spracherkennungen. Es gibt heutzutage kaum Smartphones, welche keinen Dienst wie Google Assistant, Siri, etc. zur Verfügung stellen. Dasselbe gilt für die Digitalisierung von Büchern, was sich schon seit Jahren mit Hilfe von OCR Software (Optical Character Recognition) etabliert hat. Dennoch gibt es viele Nischengebiete, bei denen es an Angeboten fehlt.

In diesem Projekt lag der Fokus auf der Digitalisierung von OL-Karten. Diese werden zwar auf dem Computer generiert und dann ausgedruckt. Man könnte also meinen, dass es zwecklos ist, die Karten wieder in die Welt der Nullen und Einer zu bringen, jedoch hat man als OL-Läufer den Zugang zu den digitalen Kartendaten oft nicht direkt. Generell können Gründe für die Digitalisierung über ein breites Spektrum variieren - sei es für die Erhaltung von Daten oder das vereinfachte Teilen über das Internet. Mein persönlicher Beweggrund war, dass ich als nicht perfekter OL-Läufer nach einem OL das Bedürfnis habe, meinen Lauf nochmals zu analysieren. Jedoch ist es nicht immer einfach, sich vorzustellen, was die beste Routenwahl gewesen wäre. Deshalb habe ich mich entschlossen, die Dinge selbst in die Hand zu nehmen, damit ich mir das Gelände besser unter die Lupe nehmen kann. Zu dem hat mich Informatik schon seit der Primarschule interessiert und ich habe auch schon mehrere Apps programmiert, welche momentan auf dem App Store zu finden sind.

2 Computer Vision

Wir Menschen haben kaum Schwierigkeiten die Welt um uns herum zu verarbeiten. Sei es ein Haus, ein Auto oder eine OL-Karte, wir erkennen ohne Problem, um was es sich dabei handelt, oder wie wir etwas zu interpretieren haben. Für einen Computer ist diese Aufgabe jedoch nicht selbstverständlich. Es wurde jedoch eine Vielfalt von Verfahren entwickelt, damit ein Computer fähig ist, anhand von digitalen Bildern, Entscheidungen zu treffen, dreidimensionale Räume zu rekonstruieren oder auch Bilder wieder weiterzuverarbeiten. Diese Prozesse sind aber nicht immer einfach zu verstehen. In der Computersprache werden Bilder als Zahlenraster repräsentiert. Dabei entsprechen die einzelnen Werte den Farbtönen der dazugehörigen Pixels (Siehe Abbildung 1). [4,9] Computer Vision beruht zum Beispiel darauf, diese Raster bzw. Matrizen mithilfe von mathematischen Operationen, oft Matrixmultiplikationen, weiterzuverarbeiten.

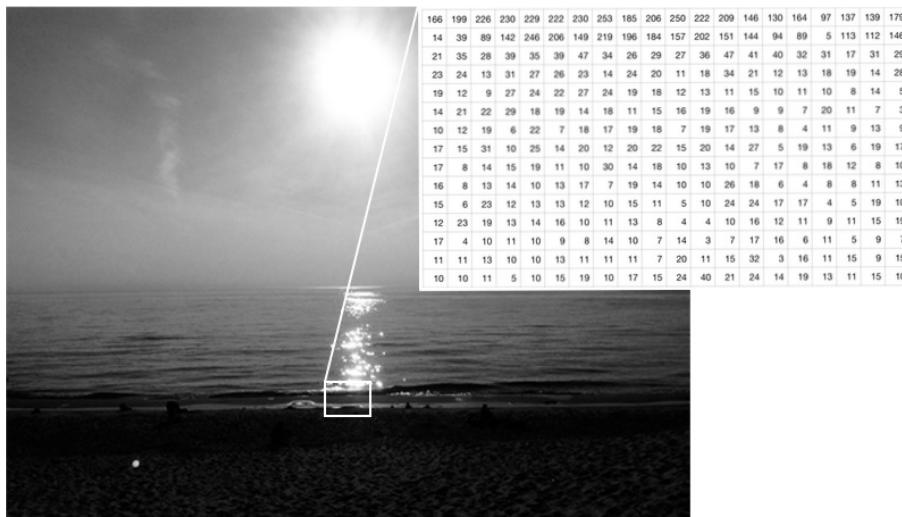


Abbildung 1: Rasterbild

2.1 OpenCV

Die Verarbeitung von visuellen Daten am Computer ist weit verbreitet und findet in vielen Sparten der Informatik seine Anwendungen. Um Projekte, welche das Verlangen von Computer Vision haben, zu realisieren, ist eine Mehrzahl von Bibliotheken vorhanden, eine davon OpenCV [6]. OpenCV ist eine open source Bibliothek, die primär in C und C++ geschrieben wurde. Es sind jedoch APIs auf verschiedenen Entwicklungsplattformen wie iOS oder Windows erhältlich. Zudem kann die Bibliothek auch mit den meisten aller häufigst gebrauchten Programmiersprachen wie Python, Java, etc. verwendet werden. Diese Bibliothek stellt diverse Funktionen

zur Verfügung, welche mit Bildverarbeitung, Bilderkennung, Machine Learning und vieles mehr zu tun haben. [9]

2.2 Anwendungen

2.2.1 Bilder entschärfen (Image smoothing)

Zu den häufigst angewendeten Bildverarbeitungsalgorithmen gehören Entschärfungen. Diese werden vor allem zum Entfernen von Bildrauschen angewendet. Unter Bildrauschen versteht man Abweichungen und Störungen der Farbwerte der Pixels eines digitalen Bildes im Vergleich zur Realität. Diese Störungen entstehen hauptsächlich bei Aufnahmen unterbelichteter Bilder oder bei der digitalen Kompression eines Bildes, um dessen Dateigröße zu vermindern. Um diese Farbabweichungen wieder herzustellen, d. h. zu interpolieren, wird jedes neu zu berechnende Pixel $g(i, j)$ mit einem gewichteten Durchschnitt des alten Pixels und dessen darumliegenden berechnet. Die Variablen i und j der Funktion $g(i, j)$ sind ganzzahlige Laufvariablen und werden von 0 bis zur Bildhöhe bzw. -breite hinaufgezählt. Mit der Funktion $f(i, j)$ werden die alten Pixelwerte (vor der Durchführung der Bildentschärfung) für die jeweilige Position angegeben. Das Ganze wird auch als linearer Filter (1) bezeichnet. Die Koeffizienten der Gewichtungen werden mit einer Matrize (auch Kernel genannt) $h(k, l)$ beschrieben. Ein solcher Kernel kann wie folgt (2) aussehen. Dabei geben m und n die Größe und h bzw. l den Index der jeweiligen Komponente des Kernels an. Dies ist jedoch einer der einfachsten linearen Filter und würde lediglich das arithmetische Mittel der benachbarten Pixel berechnen. [7, 9]

$$g(i, j) = \sum_{k,l}^{m,n} f(i + k, j + l)h(k, l) \quad (1)$$

[7]

$$h(k, l) = \frac{1}{m \cdot n} \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & 1 & 1 & \dots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 1 & \dots & 1 \end{bmatrix} \quad (2)$$

[7]

In der Praxis gibt es geeigneter Filter, um eine gewünschte Bildrausch-Reduktion zu erlangen. Einer der bekannteren wäre der Gaussian Blur, welcher die statistische Normalverteilung bzw. Gauss-Verteilung verwendet, um ein Mittel zu berechnen,

benannt nach dem Mathematiker Carl Friedrich Gauss. [1]

2.2.2 Kantenerkennung (Edge detection)

Bilder bestehen oft aus einer Mehrzahl von Farben und Intensitäten. Wenn wir Menschen gefragt werden, die Kanten in einem Bild zu markieren oder nachzumalen, dann orientieren wir uns intuitiv an abrupten Kontraständerungen. Die häufigst gebrauchten Algorithmen verwenden ein Verfahren, welches auf eine ähnliche Art und Weise funktioniert. Denn mathematisch gesehen, ist eine solche Kante nichts anderes als eine schlagartige Änderung der Farb- und Intensitätswerte. Diese Änderungsrate kann über die erste Ableitung charakterisiert werden. Da sich die Pixel eines Bildes im zwei dimensionalen Raum befinden, verwendet man dafür einen Gradienten in x und y Richtung (3). Der Gradient weist jedem Pixel einen Vektor mit zwei Komponenten zu, wobei die Länge der Änderungsrate entspricht. Eine grosse Rate bedeutet einen abrupten Farb- bzw. Intensitätswechsel, was meistens auf eine Kante hinweist. Die Richtung des Vektors gibt dann die Senkrechte deren an. In der Praxis empfiehlt es sich aber ein schwarz-weiss Bild zu verwenden, denn dann gibt es nur einen Farbwert bzw. Helligkeitswert pro Pixel. Zudem ist es sinnvoll, bevor man versucht Kanten zu erkennen, das gewünschte Bild zu entschärfen, da Bildrauschen durch den Gradienten verstärkt wird. [5]

$$\nabla I = \frac{\partial I}{\partial x} \vec{e}_x + \frac{\partial I}{\partial y} \vec{e}_y \quad (3)$$

[5]

(Die beiden Vektoren \vec{e}_x und \vec{e}_y sind die Einheitsvektoren der jeweiligen Koordinatenachse) In der Realität werden oft komplexere Verfahren verwendet, den Gradienten zu berechnen, um Kanten zu erkennen (zB. Canny Algorithmus), doch man kann sich dem Gradienten auch annähern, indem man einen Scharr Filter über das Bild laufen lässt. Ein Scharr Filter ist eine Matrix bzw. ein Kernel und kann wie folgt aussehen (4). [5]

$$\begin{bmatrix} -3 & 0 & 3 \\ -10 & 0 & 10 \\ -3 & 0 & 3 \end{bmatrix} \begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ 3 & 10 & 3 \end{bmatrix} \quad (4)$$

[5]

Die beiden Matrizen werden dann ähnlich wie bei der Bildentschärfung (Siehe Kapitel 2.2.2) über das Bild laufen gelassen, um für jedes Pixel eine lokale, diskrete

Änderungsrate in jeweils x und y Richtung zu berechnen (sprich einen angenäherten Gradienten).

3 3D

Ein Bild besteht örtlich gesehen aus zwei Dimensionen, der Breite und Höhe. Jeder Punkt dieses Bildes, also jedes Pixel hat einen genauen Ort, welcher durch einen Ortsvektor mit zwei Komponenten beschrieben werden kann. Wenn man eine dritte Koordinatenachse hinzufügt, z.B. die Länge, dann spricht man vom drei dimensionalen Raum. Wiederum kann jeder Ort mit einem Ortsvektor beschrieben werden, aber im Gegensatz zum zwei dimensionalen Raum, hat dieser drei Komponenten [8].

Objekte im 3D-Raum werden an Hand ihrer Eckpunkte beschrieben und Flächen bestehen aus lauter kleinen Dreiecken, welche miteinander verbunden sind [8].

3.1 Transformationen

Eine Transformation im 3D-Raum ist ein Vorgang, bei dem jedem Punkt P , auf den die Transformation angewendet wird, einen neuen Punkt im selben Koordinatensystem zugewiesen wird. Dazu gehören zum Beispiel Rotation, Translationen (Verschiebungen) und Skalierungen. Diese Transformationen werden mit einer Matrix charakterisiert. Um eine Transformation auf einen Punkt anzuwenden, wird der Ortsvektor des Punktes \vec{OP} mit der Matrix multipliziert. Damit die Transformationen affin sind, d.h. dass sie unter anderem umkehrbar sind und dass gerade Linien gerade bleiben [10], wird der Punkt mit einem Vektor mit vier Komponenten repräsentiert. Die Transformationen werden dann mit 4x4 Matrizen beschrieben (siehe Translationsmatrix (5)) [8].

$$\vec{OP} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, \vec{OP}' = \begin{bmatrix} 1 & 0 & 0 & x_t \\ 0 & 1 & 0 & y_t \\ 0 & 0 & 1 & z_t \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (5)$$

[8]

Um die Transformation nicht nur auf einen einzelnen Punkt sondern ein ganzes Objekt anzuwenden, wird lediglich die Transformation auf jeden einzelnen Punkt des Objektes angewendet.

4 Resultate

4.1 Probleme

Während dem Verarbeiten der OL-Karten und der Erstellung des 3D-Modells sind immer wieder Hürden aufgetaucht. Zum Teil konnten diese Hindernisse umgangen werden, jedoch gab es auch Probleme, zu denen noch keine Lösungen gefunden wurden. In den folgenden Abschnitten werden die schwierigsten Probleme besprochen, sowie allfällige Lösungen und Lösungsansätze diskutiert.

4.1.1 Isolinien Erkennung

Eine der Hauptschwierigkeiten war die Erkennung der Isolinien (Höhenkurven). Gründe dafür sind zum einen die nicht perfekten Photoaufnahmen der OL-Karten selbst (Siehe Abbildung 2). Zum Teil kann dies aber behoben werden, indem man eine Kamera mit höherer Auflösung verwendet, oder man einen Scanner benutzt. Ausserdem muss man beachten, dass die meisten Photoaufnahmegeräte die Auflösung und Qualität eines Bildes automatisch verringern, um dessen Dateigrössen zu minimieren. Dies kann jedoch häufig in den Einstellung des jeweiligen Gerätes verstellt oder ausgeschaltet werden. Für dieses Projekt wurden die OL-Karten mit einem Scanner digitalisiert und danach als JPEG (Bilddateiformat) mit einer Auflösung von 2408x3436 Pixel abgespeichert.

Eine weitere Ursache ist, dass die Höhenkurven auf den OL-Karten durch Flüsse und Bäche, sowie Strassen und Wege unterbrochen werden. Dies führt zu einer Zerstückelung dieser Linien (Siehe Abbildung 6). Um ein Relief erstellen zu können, ist jedoch wichtig, dass die Isolinien jeweils in einem Stück sind. Zur Zeit hat sich noch keine konkrete Lösung dieses Problems gefunden.

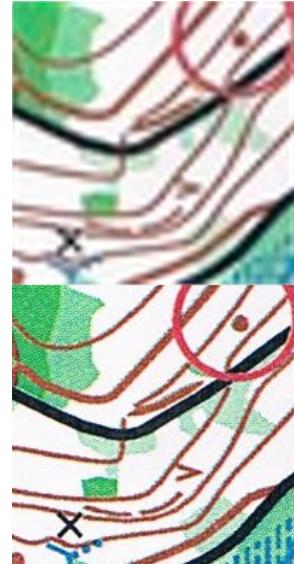


Abbildung 2: Photoaufnahme Auflösung Vergleich, oben: 50x50, unten 200x200

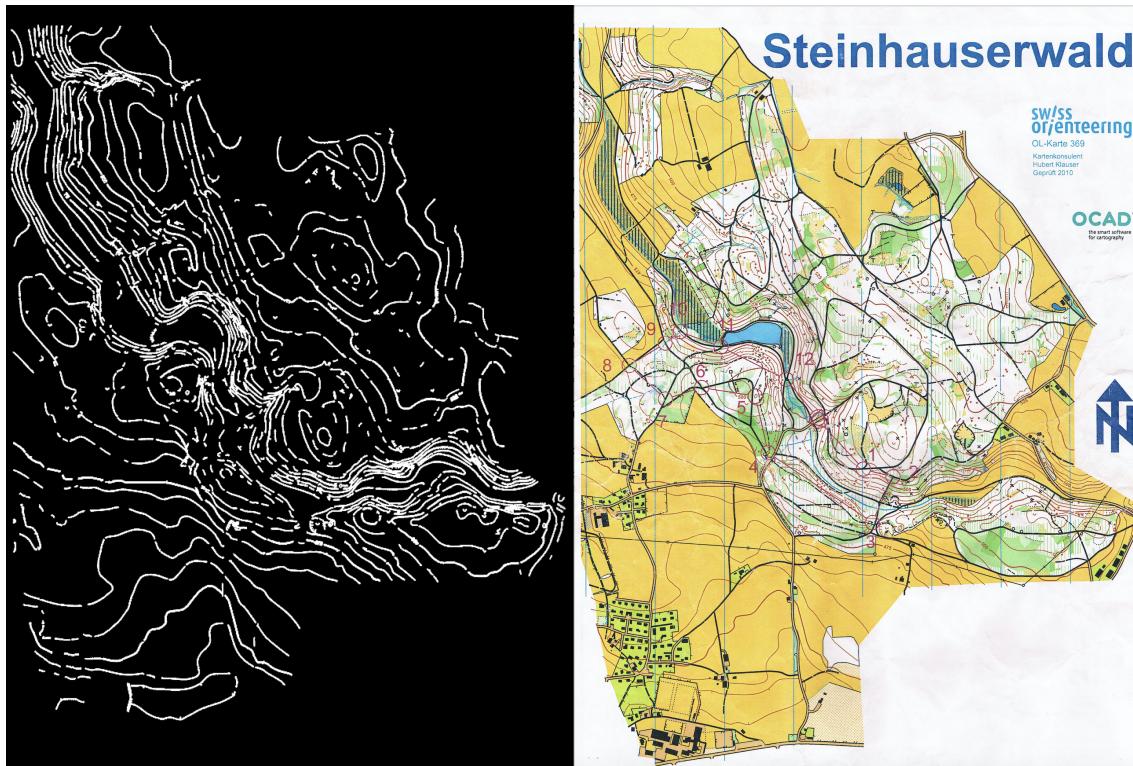


Abbildung 3: Höhenkurven Zerstückelung Vergleich, links: Höhenkurven, rechts: Original

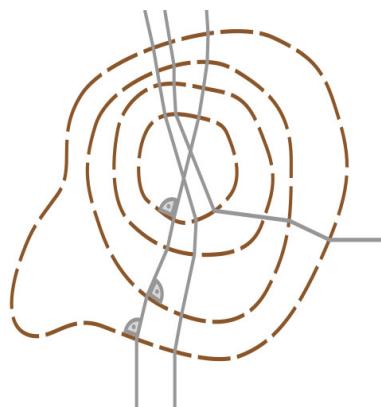


Abbildung 4: Gradienten Lösungsansatz

Ein möglicher Lösungsansatz für das Problem der zerstückelten Höhenkurven könnte darin bestehen, die Höhenkurven entlang eines Weges wieder zusammenzufügen. Dies kann durch die Verwendung von Senkrechten (Normalen) erreicht werden.

Eine Möglichkeit wäre aber, vom Bildrand aus Senkrechten bis zum ersten Linienabschnitt zu ziehen und dann von dort aus wieder eine Senkrechte zum nächsten Abschnitt zu berechnen, usw. Somit könnte man einen Gradienten konstruieren, um von diesem aus auf ein Höhendiagramm zu kommen.

Noch eine Option wäre, mit Hilfe von externen Daten ein Höhendiagramm zu erlangen. Dazu gäbe es zum Beispiel Openstreetmap [3]. Dies ist ein online Karten Service wie Google Maps [2], welcher aber eine online API zur Verfügung stellt, um Kartendaten wie Höhenkurven herunterzuladen. Dazu müsste man aber wissen, was die genaue Geolocation (Koordinaten) der OL-Karte ist.

Sobald man die Isolinien in einem Stück hat, kann man ein Höhendiagramm anfertigen. Zum Testen wurden ein paar Kurven gezeichnet und als JPEG gespeichert, welche dann erfolgreich vom Programm erkannt wurden und dann in ein Höhendiagramm verarbeitet werden konnten (Siehe Abbildung 5). Dies ist nur ein Beispiel zum Zeigen, dass dies überhaupt möglich ist (proof of concept).



Abbildung 5: Höhendiagramm aus Höhenkurven

4.2 Diskussion

Grundsätzlich war das Projekt ein Erfolg, denn die meisten meiner Ziele konnten erfüllt werden. Es gelang mir mit Hilfe eines Bildes der OL-Karten diverse Daten, wie bewaldete Regionen, Gewässer, OL-Posten, usw. zu extrahieren. Die Erkennung der Höhenkurven war mit den OL-Karten nicht möglich (Siehe Kapitel 4.1.1), jedoch funktionierte dies mit einem Testmodell, welches erfolgreich in ein 3D-Modell umgewandelt werden konnte.

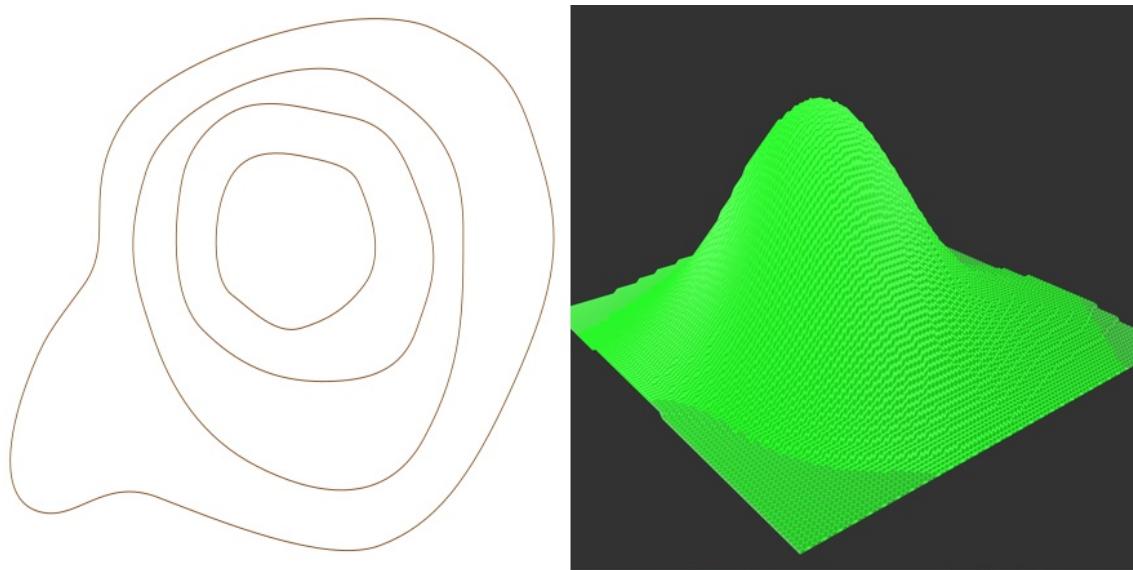


Abbildung 6: links: Höhenkurven rechts: 3D-Modell

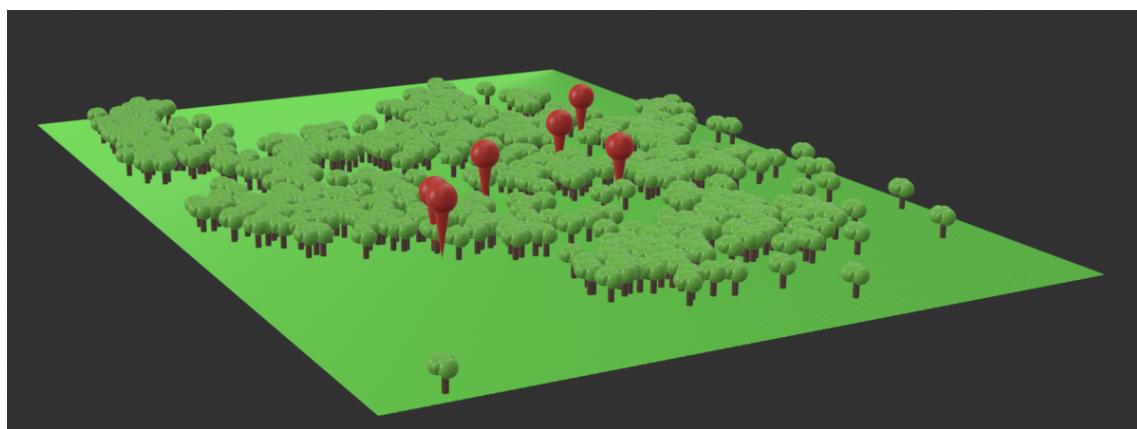


Abbildung 7: OL-Karte als Collada exportiert

4.3 Wie weiter?

Ein Fundament ist nun aufgebaut und es stellen sich viele interessante Ausbaumöglichkeiten zur Verfügung. Man könnte zum Beispiel noch mehr Objekte wie Sperrgebiete, usw. auf den OL-Karten erkennen. Eine andere Möglichkeit wäre, ein GUI

(Graphische Benutzeroberfläche) zu erstellen, damit die Benutzerfreundlichkeit erhöht wird und das Ganze einladender aussehen würde. Neben all den kosmetischen Veränderungen und nachdem die Höhenkurven zuverlässig erkannt werden können, wäre es sinnvoll, das Programm in eine Smartphoneapp einzubetten, denn solch ein Gerät trägt jeder mit sich und da bietet sich ein grosser Markt an.

Literatur

- [1] Larry Freeman. Carl friedrich gauss. <http://fermatslasttheorem.blogspot.com/2005/06/carl-friedrich-gauss.html>, Jun 2005. Besucht am 3. Sep 2018.
- [2] Google Inc. Google maps. <https://www.google.ch/maps>, 2019. Besucht am 25. Jan 2019.
- [3] Openstreetmap. Osm api v0.6. https://wiki.openstreetmap.org/wiki/API_v0.6, Dez 2019. Besucht am 25. Jan 2019.
- [4] Richard Szeliski. *Computer Vision - Algorithms and Application*, pages 3–15. Springer, 2011.
- [5] Richard Szeliski. *Computer Vision - Algorithms and Application*, pages 210–230. Springer, 2011.
- [6] OpenCV Team. Opencv. <https://opencv.org>, Sep 2018. Besucht am 3. Sep 2018.
- [7] OpenCV Team. Smoothing images. https://docs.opencv.org/2.4/doc/tutorials/imgproc/gaussian_median_bilateral_filter/gaussian_median Blur_bilateral_filter.html, Sep 2018. Besucht am 3. Sep 2018.
- [8] Tutorialspoint. Computer graphics. https://www.tutorialspoint.com/computer_graphics/3d_computer_graphics.htm, 2019. Besucht am 27. Jan 2019.
- [9] Gary Bradski und Adrian Kaehler. *Learning OpenCV - Computer Vision with the OpenCV Library*, pages 2–7. O'REILLY, 2008.
- [10] Wikipedia. Affine transformation. https://en.wikipedia.org/wiki/Affine_transformation, Jan 2019. Besucht am 27. Jan 2019.

Abbildungsverzeichnis

1	Rasterbild	4
2	Photoaufnahme Auflösung Vergleich, oben: 50x50, unten 200x200 . . .	9
3	Höhenkurven Zerstückelung Vergleich, links: Höhenkurven, rechts: Original	10
4	Gradienten Lösungsansatz	10
5	Höhendiagramm aus Höhenkurven	11
6	links: Höhenkurven rechts: 3D-Modell	12

7 OL-Karte als Collada exportiert 12