



universität
wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Entwicklung eines neuen Erscheinungsbildes für 3D-
Stadtmodelle am Beispiel der Stadt Wien“

verfasst von / submitted by

Kaufmann Thomas, BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of
Master of Science (MSc)

Wien, 2021 / Vienna 2021

Studienkennzahl lt. Studienblatt /
degree programme code as it appears on
the student record sheet:

A 066 856

Studienrichtung lt. Studienblatt /
degree programme as it appears on
the student record sheet:

Kartographie und Geoinformation

Betreut von / Supervisor:

Ass.-Prof. Mag. Dr. Andreas Riedl

Inhaltsverzeichnis

Inhaltsverzeichnis	ii
Abbildungsverzeichnis	1
Tabellenverzeichnis	5
Abkürzungsverzeichnis	5
Kurzfassung	6
Abstract	7
1. Einleitung	8
1.1 Motivation und Zielsetzung	8
1.2 Problemstellung und Relevanz der Arbeit	8
1.3 Forschungsfragen	10
1.4 Arbeitsfragen	10
1.5 Methodische Vorgehensweise	10
2. 3D-Stadtmodelle	12
2.1. Standards und Formate	12
2.1.1. Standard versus Format	12
2.1.2. Standards für 3D-Stadtmodelle	12
2.2. CityGML	15
2.2.1. Grundlagen	15
2.2.2. Levels of Detail	17
2.2.3. Module	19
2.3. Das 3D-Stadtmodell von Wien	40
2.3.1. LOD-Matrix der Stadt Wien	40
2.3.2. Bestandteile	41
2.3.3. Software	45
3. Farben	47
3.1. Entstehung von Farben	47
3.2. Lichtfarben	49
3.3. Körperfarben	50
3.4. Farbsysteme	51
3.4.1. CIE-Normvalenzsystem	51
3.4.2. RGB	53
3.4.3. CMYK	54
3.4.4. HSV und HSL	54
4. Luftbildfotografie	56
4.1. Aufbau und Funktionsweise einer Digitalkamera	56

4.2.	Bildflug	57
4.3.	Eigenschaften von Luftbildern	58
4.3.1.	Auflösung	58
4.3.2.	Farbtiefe	59
4.3.3.	Dateiformate	59
4.3.4.	Orientierung	60
4.3.5.	Orthofoto	61
4.3.6.	Bildsturz	62
5.	Grundlagen der Statistik	63
5.1.	Allgemeines	63
5.2.	Arten der Statistik	63
5.2.1.	Deskriptive Statistik	63
5.2.2.	Inferenzstatistik	64
5.3.	Statistische Kennzahlen	64
5.3.1.	Arithmetischer Mittelwert	64
5.3.2.	Modus (Modalwert)	65
5.3.3.	Median (Zentralwert)	65
5.3.4.	Minimum und Maximum	65
5.3.5.	Quantile	65
5.3.6.	Spannweite	66
5.4.	Clusteranalyse	66
5.4.1.	Partitionierende Verfahren	66
5.4.2.	Hierarchische Verfahren	67
6.	Das neue Erscheinungsbild für 3D-Stadtmodelle	68
6.1.	Beschreibung des Erscheinungsbildes bestehender 3D-Stadtmodelle	68
6.1.1.	Wien	68
6.1.2.	Berlin	69
		69
6.1.3.	Bremen	70
6.1.4.	Posen	71
6.1.5.	Schweiz	72
6.1.6.	Paris	73
6.1.7.	New York	73
6.1.8.	Kurzübersicht bestehender Erscheinungsbilder	74
6.2.	Vergleich verschiedener LOD-Konzepte für Gebäude	75
6.2.1.	CityGML	75
6.2.2.	BLOM3D™	75

6.2.3.	NAVTEQ	76
6.3.	Entwurf eines neuen Erscheinungsbildes	76
6.3.1.	Allgemeines	76
6.3.2.	Idee.....	77
6.3.3.	Versuch einer eigenen LOD-Definition	78
7.	Softwarewahl zur Farbwertextraktion aus Gebäudetexturen	81
7.1.	Anwendungssoftware oder Programmiersprache?	81
7.1.1.	Anwendungssoftware	81
7.1.2.	Programmierersprache	82
7.1.3.	Wahl zwischen Anwendungssoftware und Programmiersprache.....	83
7.2.	Vergleich ausgewählter Programmiersprachen.....	84
7.2.1.	Auswahl zu vergleichender Programmiersprachen	84
7.2.2.	Kriterien für Vergleich.....	84
7.2.3.	Vergleich	85
7.3.	Warum Python?	85
8.	Automatisierte Farbwertextraktion	86
8.1.	Ausgangslage.....	86
8.2.	Ausgangsdaten.....	87
8.3.	Texturierung.....	88
8.4.	Theoretische Methoden zur Ermittlung eines durchschnittlichen Farbwertes (Farbwertextraktion).....	90
8.4.1.	Repräsentativer Farbwert	91
8.4.2.	Licht – Schatten	92
8.4.3.	Vegetation	93
8.5.	Funktionsweise des Python-Codes	95
8.5.1.	Umrechnung zwischen Farbsystemen (<i>colorconverter.py</i>).....	96
8.5.2.	Umrechnung der Koordinaten (<i>coordinates.py</i>).....	96
8.5.3.	Farbwertberechnung (<i>color.py</i>)	97
8.5.4.	Farbwertextraktion (<i>farbwertextraktion.py</i>)	97
8.6.	Ergebnisse der Farbwertextraktion	99
8.6.1.	Hinterhainbach – Kachel 083-091	99
8.6.2.	Franz-Josefs-Bahnhof – Kachel 104-086.....	102
8.6.3.	Richard-Wagner-Platz – Kachel 099-082	107
8.6.4.	Stephansplatz – Kachel 106-082.....	110
9.	Darstellung der Bodenbedeckung.....	112
9.1.	Definition.....	112
9.2.	Notwendigkeit einer Darstellung der Bodenbedeckung.....	112
9.3.	Art der Darstellung	114

9.3.1.	Bestehende Darstellungsformen der Bodenbedeckung.....	114
9.3.2.	Neue Darstellungsform der Bodenbedeckung	115
9.4.	Datenquelle(n)	117
9.4.1.	Mehrzweckkarte (MZK).....	117
9.4.2.	Flächen-Mehrzweckkarte (FMZK)	117
9.5.	Bodenbedeckungsklassen	118
9.5.1.	Bodennutzungsklassen der Flächen-Mehrzweckkarte	118
9.5.2.	„Gewässer-Namens-Klassen“ der Flächen-Mehrzweckkarte	120
9.5.3.	Einteilung der Bodenbedeckungsklassen	120
9.5.4.	Farbgebung der Bodenbedeckungsklassen	123
9.5.5.	Technische Aspekte der Bodenbedeckungsklassen.....	128
10.	Fazit und Ausblick.....	131
10.1.	Zusammenfassung.....	131
10.2.	Beantwortung der Forschungsfragen	131
10.3.	Beantwortung der Arbeitsfragen	132
10.4.	Herausforderungen und Probleme	132
10.5.	Ausblick	133
	Quellenverzeichnis	134
	Anhang	146
	Ablaufdiagramm	146
	Umrechnung zwischen Farbsystemen (<i>colorconverter.py</i>)	147
	Umrechnung der Koordinaten (<i>coordinates.py</i>).....	148
	Farbwertberechnung (<i>color.py</i>)	149
	Farbwertextraktion (<i>farbwertextraktion.py</i>).....	157

Abbildungsverzeichnis

Abbildung 1: Level of Detail 0 (Quelle: nach IGG Uni Bonn o.J.).	18
Abbildung 2: Level of Detail 1 (Quelle: nach IGG Uni Bonn o.J.).	18
Abbildung 3: Level of Detail 2 (Quelle: nach IGG Uni Bonn o.J.).	18
Abbildung 4: Level of Detail 3 (Quelle: nach IGG Uni Bonn o.J.).	19
Abbildung 5: Level of Detail 4 (Quelle: nach IGG Uni Bonn o.J.).	19
Abbildung 6: Texture Mapping (Quelle: Tech Target 2021).	24
Abbildung 7: Wrap-Mode: Textur (a), none (b), wrap (c), mirror (d), clamp (e), border (f) (Quelle: Open Geospatial Consortium 2012: 38).	24
Abbildung 8: TIN: links in 2D mit Umkreisen der Dreiecke und rechts in 3D mit höhenabhängiger Einfärbung (QGIS o.J.).	25
Abbildung 9: Raster Höhenmodell aus SRTM-Daten (Quelle: USGS o.J.).	26
Abbildung 10: Arten von Dachformen (Quelle: Prefa 2021).	29
Abbildung 11: LOD1 Stadtmodell Leipzig (Quelle: Stadt Leipzig 2021a).	30
Abbildung 12: LOD2 Stadtmodell Leipzig (Quelle: Stadt Leipzig 2021b).	31
Abbildung 13: LOD3 Stadtmodell Leipzig (Quelle: Stadt Leipzig 2021c).	32
Abbildung 14: IndoorGML (Quelle: Open Geospatial Consortium o.J.).	32
Abbildung 15: LOD-Matrix von Biljecki (Quelle: Biljecki 2017: 76).	33
Abbildung 16: Arten von Vegetation (Quelle: Open Geospatial Consortium 2012: 132).	34
Abbildung 17: Detailliertes Baummmodell als implizite Geometrie (Quelle: Evermotion o.J.).	35
Abbildung 18: Schematische Darstellung von Fächergeometrien (Quelle: SACH 2018).	35
Abbildung 19: Analytische Geometrien (Quelle: HELDENS et. al. 2020).	36
Abbildung 20: Parameter Analytischer Geometrien (Quelle: HELDENS et. al. 2020).	36
Abbildung 21: Tunnel in den Levels of Detail 1 bis 4 (Quelle: nach Open Geospatial Consortium 2012: 82).	37
Abbildung 22: Brücken in den Levels of Detail 1 bis 4 (Quelle: nach Open Geospatial Consortium 2012: 99).	38
Abbildung 23: LOD-Matrix der Stadt Wien (Quelle: nach LEHNER UND DORFFNER 2020).	40
Abbildung 24: Baukörpermodell (Quelle: Stadt Wien o.J.a).	41
Abbildung 25: LOD2.1 Gebäudemodell (Quelle: Stadt Wien o.J.f).	42
Abbildung 26: LOD2.4 Gebäudemodell (Quelle: Stadt Wien o.J.a).	42
Abbildung 27: LOD3.4 Gebäudemodell (Quelle: Stadt Wien o.J.g).	43
Abbildung 28: Brücken im 3D-Stadtmodell von Wien (Quelle: Stadt Wien o.J.a).	43
Abbildung 29: Bodenbedeckung mit Orthofoto (Quelle: Stadt Wien o.J.a).	44
Abbildung 30: Bodenbedeckung mit Grundkarte, welche ein ähnliches Design wie die Standardkarte von basemap.at aufweist (Quelle: Stadt Wien o.J.a).	44
Abbildung 31: 3D-Geländemodell von Wien (Quelle: Stadt Wien o.J.a).	44
Abbildung 32: 3D-Modell einer U-Bahnstation in Wien (Quelle: UVM-Systems o.J.d).	45
Abbildung 33: Spektrum des sichtbaren Lichts (Quelle: SCHÜNGEL 1999).	47
Abbildung 34: Sonnenlicht (Quelle: Pxhere o.J.a).	48
Abbildung 35: Kerzenlicht (Quelle: Pxhere o.J.b).	48
Abbildung 36: roter Farbfilter (Quelle: FERMUM o.J.).	48
Abbildung 37: additive Farbmischung (Quelle: WECKNER o.J.).	49
Abbildung 38: Einzelpixel eines Monitors bestehend aus den Farben Rot, Grün und Blau (Quelle: Imaging Arts Printing 2015).	49
Abbildung 39: subtraktive Farbmischung (Quelle: WECKNER o.J.).	50
Abbildung 40: Entstehung von Körperfarben (Quelle: eigene Abbildung).	50
Abbildung 41: Farbkreis von Isaac Newton (Quelle: BAUMANN 2011).	51
Abbildung 42: Farbkreis von Johannes Itten (Quelle: KÖSTNER o.J.).	51

Abbildung 43: CIE-Normfarbtafel (Quelle: ANDERS 2005).....	52
Abbildung 44: RGB-Farbwürfel und subtraktive Farbmischung (Quelle: GONZALEZ UND WOODS o.J.; QUARK67 2006).	53
Abbildung 45: CMYK Farben (Quelle: Sorembâ GmbH 2020).	54
Abbildung 46: Unterschied zwischen HSV und HSL (Quelle: (vgl. HÄSSLER 2018).	55
Abbildung 47: Aufbau einer (Digital-) Kamera (Quelle: Creative Dragon Works 2020).....	56
Abbildung 48: Luftbildflugzeug Piper PA 31 Chieftain der Firma GeoFly (Quelle: CT Aeropics o.J.).....	57
Abbildung 49: Bildüberdeckung (Quelle: Spektrum der Wissenschaft Verlagsgesellschaft mbH 2001d).....	57
Abbildung 50: Vergleich verschiedener Farbtiefen (Quelle: MAIER 2015).....	59
Abbildung 51: Unterschied zwischen Luftbild (re.) und Orthofoto (li.) (Quelle: ÖTK Österreichischer Touristenklub 2018).	61
Abbildung 52: Bildsturz beim Ringturm in Wien (Quelle: Stadt Wien und Österreichische Länder 2016).	62
Abbildung 53: LOD2.1-Gebäude mit roten Dächern und grauen Fassaden (Quelle: Stadt Wien o.J.f).....	68
Abbildung 54: texturierte LOD2.1-Gebäude mit Orthofoto als Bodenbedeckung (Quelle: Stadt Wien intern).....	68
Abbildung 55: detailliertes LOD3-Architekturmodell (Quelle: Stadt Wien o.J.g).	69
Abbildung 56: LOD2-3D-Stadtmodell von Berlin mit texturierten Gebäuden, Bäumen und Orthofoto (Quelle: Virtual City Systems o.J.c).	69
Abbildung 57: LOD2-3D-Stadtmodell von Berlin mit weißen Gebäuden, Bäumen und OpenStreetMap (Quelle: Virtual City Systems o.J.c).....	69
Abbildung 58: 3D-Stadtmodell von Berlin als 3D-Mesh (Quelle: Virtual City Systems o.J.c).	70
Abbildung 59: Texturiertes Oberflächenmodell der Stadt Bremen (Quelle: Virtual City Systems o.J.d).....	70
Abbildung 60: Graue LOD2-Gebäude in Bremen mit Grundkarte als Bodenbedeckung (Quelle: Virtual City Systems o.J.d).....	70
Abbildung 61: 3D-Stadtmodell Posen mit realitätsnahe eingefärbten Dächern und grauen Fassaden, Grundkarte als Bodenbedeckung (Quelle: Stadt Posen o.J.).	71
Abbildung 62: 3D-Stadtmodell Posen mit roten Dächern und grauen Fassaden, Grundkarte als Bodenbedeckung (Quelle: Stadt Posen o.J.).	71
Abbildung 63: 3D-Stadtmodell Posen mit texturiertem Oberflächenmodell (3D-Mesh) (Quelle: Stadt Posen o.J.).	72
Abbildung 64: 3D-Stadtmodell Posen mit texturierten LOD2-Gebäuden, Orthofoto als Bodenbedeckung (Quelle: Stadt Posen o.J.).	72
Abbildung 65: 3D-Modell der Schweiz (hier: Zürich) (Quelle: Bundesamt für Landestopographie swisstopo o.J.).....	72
Abbildung 66: Paris in 3D (Quelle: GLAVENU 2020).	73
Abbildung 67: 3D-Stadtmodell von New York, erstellt von der TU München (Quelle: Kolbe 2016).....	74
Abbildung 68: Visualisierung dreidimensionaler OpenStreetMap-Daten durch die F4map (Quelle: F4 o.J.).....	78
Abbildung 69: Gebäudemodell, texturiert mit einer Software der Firma ViewApp (links: Originalgebäude, rechts: texturiertes Gebäude) (Quelle: ViewApp GmbH o.J.).	80
Abbildung 70: texturiertes LOD2-Gebäudemodell der Stadt Wien (Quelle: Stadt Wien intern).	80
Abbildung 71: LOD2.4 Gebäudemodell von Wien ohne Texturen (Quelle: Stadt Wien o.J.a).	87

Abbildung 72: texturiertes LOD2.1 Stadtmodell von Wien (Quelle: Stadt Wien intern)	87
Abbildung 73: Kacheln im Blattschnitt 1:1000 (Quelle: Stadt Wien o.J.e).	87
Abbildung 74: Texturdatei für ein Gebäude der Kachel 079-072 (Quelle: Stadt Wien intern).	88
Abbildung 75: Ausschnitt aus der CityGML-Datei der Kachel 106-082. Die relativen Koordinaten sind hier in den Zeilen 28 und 33 zu sehen (Quelle: Stadt Wien intern).....	89
Abbildung 76: Zusammenhang zwischen der Koordinatenangabe in der CityGML-Datei (Werte zwischen 0 und 1) und in der Texturdatei (in Pixel) (Quelle: eigene Abbildung).....	90
Abbildung 77: Fassade mit unterschiedlichen Farben und Fenstern (Quelle: Stadt Wien intern).	91
Abbildung 78: einfarbige Fassade mit Fenstern (Quelle: Stadt Wien intern).	91
Abbildung 79: Schattenseite von Gebäuden (Quelle: Stadt Wien intern).	92
Abbildung 80: Sonnenseite von Gebäuden (Quelle: Stadt Wien intern).....	92
Abbildung 81: unerwünschte Vegetation an Gebäudefassade (Quelle: Stadt Wien intern).	94
Abbildung 82: vollständig mit Vegetation bedeckte Gebäudetexturen (Quelle: Stadt Wien intern).	94
Abbildung 83: Grundlegender Ablauf der Farbwertextraktion (Quelle: eigene Abbildung). ..	95
Abbildung 84: Lage von Hinterhainbach innerhalb von Wien (Quelle: Stadt Wien und Österreichische Länder 2016).	99
Abbildung 85: Lage der Kachel 083-091 innerhalb von Hinterhainbach (Quelle: Stadt Wien und Österreichische Länder 2016).	100
Abbildung 86: Hinterhainbach ohne Texturen (Quelle: Stadt Wien intern).....	100
Abbildung 87: Hinterhainbach mit Texturen (Quelle: Stadt Wien intern).	100
Abbildung 88: Hinterhainbach mit Farbwerten, berechnet aus dem Mittelwert (Quelle: Stadt Wien intern).....	101
Abbildung 89: Hinterhainbach mit Farbwerten, berechnet aus dem Modalwert (Quelle: Stadt Wien intern).....	101
Abbildung 90: Hinterhainbach mit Farbwerten, berechnet mittels Clusteranalyse (Quelle: Stadt Wien intern).....	101
Abbildung 91: Lage des Franz-Josefs-Bahnhofs innerhalb von Wien (Quelle: Stadt Wien und Österreichische Länder 2016).	102
Abbildung 92: Karte des Untersuchungsgebietes um den Franz-Josefs-Bahnhof (Quelle: Stadt Wien und Österreichische Länder).	102
Abbildung 93: Franz-Josefs-Bahnhof ohne Texturen (Quelle: Stadt Wien intern).	103
Abbildung 94: Franz-Josefs-Bahnhof mit Texturen (Quelle: Stadt Wien intern).....	103
Abbildung 95: Gradationskurve in GIMP (Quelle: Screenshot).	103
Abbildung 96: Angepasster Screenshot als Hintergrund in Desmos und Punkte aus Tabelle (über Hintergrund ermittelt) (Quelle: Screenshot).	104
Abbildung 97: Angepasste Kurve (Exponentialfunktion) in MyCurveFit (Quelle: Screenshot).	105
Abbildung 98: Franz-Josefs-Bahnhof mit Farbwerten aus dem Modalwert (Quelle: Stadt Wien intern).....	105
Abbildung 99: mit Farbwerten aus dem Modalwert und geringfügiger Erhöhung der Helligkeit in dunklen Bereichen (Quelle: Stadt Wien intern).	105
Abbildung 100: Franz-Josefs-Bahnhof mit Farbwerten, berechnet mittels Clusteranalyse (Quelle: Stadt Wien intern).....	106
Abbildung 101: Lage des Richard-Wagner-Platzes innerhalb von Wien (Quelle: Stadt Wien und Österreichische Länder 2016).	107
Abbildung 102: Karte des Untersuchungsgebietes um den Richard-Wagner-Platz (Quelle: Stadt Wien und Österreichische Länder 2016).....	107
Abbildung 103: Richard-Wagner-Platz ohne Texturen (Quelle: Stadt Wien intern).....	108

Abbildung 104: Richard-Wagner-Platz mit Texturen (Quelle: Stadt Wien intern).	108
Abbildung 105: 3D-Stadtmodell ohne Algorithmus (links) und mit Algorithmus (rechts) (Quelle: Stadt Wien intern).....	108
Abbildung 106: 3D-Stadtmodell ohne Algorithmus (links) und mit Algorithmus (rechts) (Quelle: Stadt Wien intern).....	109
Abbildung 107: Richard-Wagner-Platz mit Farbwerten, berechnet mittels Clusteranalyse (Quelle: Stadt Wien intern).....	109
Abbildung 108: Lage des Richard-Wagner-Platz innerhalb von Wien (Quelle: Stadt Wien und Österreichische Länder 2016).	110
Abbildung 109: Karte des Untersuchungsgebietes um den Stephansplatz (Quelle: Stadt Wien und Österreichische Länder 2016).	110
Abbildung 110: Stephansplatz ohne Texturen (Quelle: Stadt Wien intern).	111
Abbildung 111: Stephansplatz mit Texturen (Quelle: Stadt Wien intern).	111
Abbildung 112: Stephansplatz mit Farbwerten, berechnet mittels Clusteranalyse (Quelle: Stadt Wien intern).....	111
Abbildung 113: 3D-Stadtmodell mit 3D-Geländedarstellung (Quelle: F4 o.J.).	113
Abbildung 114: 3D-Stadtmodell ohne 3D-Geländedarstellung (Quelle: F4 o.J.).	113
Abbildung 115: 3D-Stadtmodell von Wien mit Gebäuden passend zur Grundkarte (Quelle: Stadt Wien o.J.a).....	114
Abbildung 116: 3D-Stadtmodell von Wien mit Gebäuden passend zum Orthofoto (Quelle: Stadt Wien intern).....	114
Abbildung 117: 3D-Stadtmodell von Wien mit texturierten Gebäuden und Grundkarte als Bodenbedeckung (Quelle: Stadt Wien intern).....	115
Abbildung 118: 3D-Stadtmodell von Wien mit untexturierten Gebäudemodellen und Orthofoto als Bodenbedeckung (Quelle: Stadt Wien intern).....	115
Abbildung 119: Mehrzweckkarte (Quelle: Stadt Wien o.J.d).	117
Abbildung 120: Flächenmehrzweckkarte (Quelle: Stadt Wien o.J.c).	117
Abbildung 121: fehlende Eisenbahnbrücke (Quelle: Stadt Wien intern).	121
Abbildung 122: fehlende Brücken bei Autobahnknoten (Quelle: Stadt Wien intern).....	121
Abbildung 123: Unterschiedliche Farben von Donau und Neuer Donau (Quelle: Stadt Wien und Österreichische Länder 2016).	121
Abbildung 124: Farbe von Wienfluss (Quelle: Stadt Wien und Österreichische Länder 2016).	121
Abbildung 125: Versiegelte Flächen (Quelle: Stadt Wien intern).	124
Abbildung 126: Tendenz zu dunkleren Verkehrsflächen und helleren Fußgängerflächen (Quelle: Stadt Wien und Österreichische Länder 2016).....	125
Abbildung 127: Nachahmung der in Abbildung 122 beschriebenen Tendenz (Quelle: Stadt Wien intern).....	125
Abbildung 128: Unterschiedliche Farbgebung der Gewässerflächen (hier: Donau und Neue Donau) (Quelle: Stadt Wien intern).	126
Abbildung 129: Gewässerfläche in nicht-blauer Farbgebung (Quelle: Stadt Wien intern). ..	126
Abbildung 130: Darstellung der Vegetationsflächen in der Bodenbedeckung (Quelle: Stadt Wien intern).....	127
Abbildung 131: Unterschiedliche Grasflächen in der Bodenbedeckung (Wiese und Grünfläche) (Quelle: Stadt Wien intern).	128
Abbildung 132: bestehendes Design für die Bodenbedeckung (Quelle: Stadt Wien intern). ..	129
Abbildung 133: neu entworfenes Design für die Bodenbedeckung (Quelle: Stadt Wien intern).	129
Abbildung 134: Layer Eigenschaften des neuen Bodenbedeckungslayers (Quelle: Stadt Wien intern).	129

Tabellenverzeichnis

Tabelle 1: Mögliche Materialeigenschaften für das X3D-Material (Quelle: Coors et al. 2016: 81).....	23
Tabelle 2: Auswirkung der Veränderung der Helligkeit auf die Werte von RGB und HSV (Quelle: eigene Tabelle).	55
Tabelle 3: Kurzübersicht bestehender Erscheinungsbilder.....	74
Tabelle 4: Vergleich der Programmiersprachen Python, Java und C.	85
Tabelle 5: Liste der Bodenbedeckungsklassen (Reihenfolge wie Layerstruktur).	122

Abkürzungsverzeichnis

2D	Zweidimensional
3D	Dreidimensional
DEM	Digital Elevation Model
DGM	Digitales Geländemodell
DOM	Digitales Oberflächenmodell
DSM	Digital Surface Model
GIS	Geoinformationssystem
GML	Geographic Markup Language
JSON	JavaScript Object Notation
KML	Keyhole Markup Language
LOD	Level of Detail
OBJ	Objekt (Dateiformat)
OGC	Open Geospatial Consortium
OGD	Open Government Data
SIG	Spatial Interest Group
SRTM	Shuttle Radar Topography Mission
TIFF	Tagged Image File Format
TIN	Triangulated Irregular Network
XML	Extensible Markup Language
XSD	XML-Schema Definition

Kurzfassung

Bei den meisten 3D-Stadtmodellen werden so viele Details wie möglich in das Erscheinungsbild mit eingebunden. Meist ist dies eine Darstellung mittels Texturen. Durch Generalisierung von Texturen ließe sich jedoch ein etwas übersichtlicheres Erscheinungsbild erreichen. Genau an diesem Punkt setzt diese Masterarbeit an. Ziel ist es, ein neues Erscheinungsbild für 3D-Stadtmodelle zu entwickeln, in dem jede Gebäudefläche nur einen einzelnen Farbwert besitzt, welcher von einer Textur extrahiert wurde. Auch die Bodenbedeckung wird bei dem neuen Erscheinungsbild berücksichtigt, indem für jede Bodenbedeckungsklasse ein passender Farbwert vergeben wird.

Für den praktischen Teil dieser Arbeit diente die Stadt Wien als Beispiel. Um die große Anzahl an Gebäuden verarbeiten zu können, musste eine vollautomatische Lösung gefunden werden. Diese lag in der Entwicklung eines Python-Skripts, welches jeder Gebäudefläche auf Basis der jeweiligen Textur einen eigenen Farbwert zuweist. Somit kann auch ohne Texturen ein relativ guter Wiedererkennungswert gewährleistet werden, ohne die Nachteile von Texturen in Kauf nehmen zu müssen. Diese wären Schatten, Vegetation und andere Objekte, welche in der Realität nicht Teil einer Gebäudefläche sind. Bei der Bodenbedeckung wurden die Farbwerte manuell für die jeweiligen Klassen, wie zum Beispiel Verkehrsflächen oder Gewässerflächen, festgelegt.

Der praktische Teil wurde in Zusammenarbeit mit der Stadtvermessung Wien MA41 im Zuge einer Anstellung als Saisontechniker erarbeitet.

Abstract

In most 3D city models, as many details as possible are included in their appearances. Mostly these details are represented by textures. However, by generalizing textures, a somewhat clearer appearance could be achieved. Exactly at this point this master thesis starts. The goal is to develop a new appearance for 3D city models, in which each building surface has only a single color value, which was extracted from a texture. Land cover is also taken into account in the new appearance by assigning a matching color value for each land cover class.

For the practical part of this work, the city of Vienna served as an example. In order to be able to process the large number of buildings, a fully automated solution had to be found. This lay in the development of a Python script, which assigns each building surface its own color value based on the respective texture. Thus, a relatively good recognition value can be guaranteed even without textures and without having to accept the disadvantages of textures. These would be shadows, vegetation and other objects which are not part of a building surface in reality. For the ground cover, the color values were set manually for the respective classes, such as traffic areas or water surfaces.

The practical part was developed in cooperation with the Vienna City Surveyor MA41 as part of a job as a seasonal technician.

1. Einleitung

3D-Stadtmodelle sind heutzutage ein wichtiger Bestandteil der Stadtplanung geworden. Dabei spielt neben der Semantik auch das Erscheinungsbild dieser eine wichtige Rolle. Bei vielen 3D-Stadtmodellen konnte vom Autor dieser Masterarbeit beobachtet werden, dass das Erscheinungsbild entweder sehr einfach gehalten ist, oder dass es relativ viele Details enthält. Bei weniger detaillierten Modellen besitzen oft alle Gebäude die gleiche Farbe und bei detaillierteren Modellen werden meist Texturen verwendet. Dabei könnte für die Erhaltung eines guten Wiedererkennungswertes der Einsatz einzelner Farbwerte anstatt von Texturen bereits einen entscheidenden Beitrag leisten.

1.1 Motivation und Zielsetzung

Diese Masterarbeit hat das Ziel, ein neues Erscheinungsbild für 3D-Stadtmodelle zu erarbeiten. Da die Betrachtung aller Komponenten und Aspekte von 3D-Stadtmodellen den Rahmen dieser Arbeit sprengen würde, wurde der Fokus auf die Farbgebung von Gebäuden und Bodenbedeckung gelegt. Konkret soll es dabei um die Ableitung von Farbwerten aus Gebäudetexturen und um den Entwurf einer passenden Farbgebung für die Bodenbedeckung gehen.

Als Motivation dient das besondere Interesse an 3D-Stadtmodellen und deren Erscheinungsbilder. Für das praktische Beispiel wurde die Stadt Wien ausgewählt, da sie bereits ein sehr gut entwickeltes dreidimensionales Stadtmodell besitzt, welches zusätzlich noch Potential zur Erweiterung bietet (vgl. Stadt Wien o.J.a).

1.2 Problemstellung und Relevanz der Arbeit

In CityGML werden Objekte von 3D-Stadtmodellen in Module eingeteilt, welche eine semantische Differenzierung dieser Modelle ermöglichen. Das Appearance-Modul nimmt dabei eine besondere Stellung ein, da es Objekte nicht in Klassen zusammenfasst, sondern unabhängig von den restlichen Modulen das individuelle Erscheinungsbild der Objekte in Form von Farben und Texturen bestimmt. (vgl. LÖWNER et al. 2012: 342)

Im 3D-Modell der Stadt Wien (vgl. Stadt Wien o.J.) sind zum gegenwärtigen Zeitpunkt keine individuellen Farbwerte für Fassaden und Dächer erfasst. Daher wird im Appearance-Modul jeweils ein einheitlicher Farbwert für alle Dachflächen und ein einheitlicher Farbwert für alle Fassadenflächen definiert. Die Bodenbedeckung wird von der offiziellen Grundkarte der Stadt Wien gebildet, welche auf der sogenannten „*Flächen-Mehrzweckkarte*“ basiert.

Zusätzlich existieren für einige Gebäude auch Texturen der Dach- und Fassadenflächen, welche ebenfalls – wie bereits beschrieben – im Appearance-Modul definiert sind. Diese wurden von der Firma UVM Systems (vgl. UVM Systems o.J.a) auf Basis von Senkrecht-Luftbildern hergestellt. In BILJECKI 2017: 43 ist erwähnt, dass Texturen in CityGML nicht Teil der LOD-Definition sind. Demnach können zum Beispiel Objekte in LOD1 voll texturiert sein, während LOD3 Objekte nur mit gleichen Farben ausgestattet sind.

Laut HAKE und GRÜNREICH 1994 sind Modelle eine Vereinfachung der Wirklichkeit und ermöglichen erst dadurch, die Fülle an Umweltinformationen zu verarbeiten. 3D-Stadtmodelle würden demnach dieser Definition entsprechen, wenn man – unter Berücksichtigung des Level of Detail (vgl. LÖWNER et al. 2012: 342) – bewusst auf nicht relevante Informationen verzichtet. Das Weglassen von Texturen würde laut HAKE und GRÜNREICH 1994 nun einer solchen Vereinfachung entsprechen.

Benötigt man nun aber ein 3D-Stadtmodell, in dem man Gebäude nicht nur anhand ihrer Form identifizieren kann, bei dem aber Texturen schon einen zu hohen Detaillierungsgrad aufweisen, so müsste man die Dach- und Fassadenflächen mit einem Farbwert versehen, welcher in etwa dem durchschnittlichen Farbwert einer Textur selbiger Flächen entspricht.

Die Stadt Posen in Polen hat dies bereits mit den Dachflächen durchgeführt und konnte dadurch den Wiedererkennungswert von Gebäude steigern, ohne den Detaillierungsgrad und damit auch die Informationsdichte durch Texturen massiv zu erhöhen. Die Farbwerte der Dachflächen wurden dabei aus einem Orthofoto mit der Software FME abgeleitet. (vgl. Stadt Posen o.J.)

So wie in Posen sollen auch in Wien die Dachflächen und zusätzlich auch die Fassadenflächen mit Farbwerten versehen werden.

Da die Stadt Wien diesen Ansatz bis jetzt noch nicht verfolgt hat, soll diese Masterarbeit den entscheidenden Beitrag dazu leisten, damit dieser auch in Wien verfolgt und umgesetzt werden kann. Damit soll einerseits ein zusätzliches Erscheinungsbild für 3D-Stadtmodelle in Wien umgesetzt werden, aber auch vor allem die Performanz bei großflächigen Visualisierungen gesteigert werden, indem keine Texturen mehr geladen müssen, um einen realitätsnahen Eindruck zu erzeugen.

Bei Erscheinungsbildern mit Texturen kommt als Bodenbedeckung meist ein Orthofoto zum Einsatz. Ansonsten versucht man, die Gebäude möglichst passend zur Grundkarte zu visualisieren. Daher soll auch für das neue Erscheinungsbild der Gebäude eine dazu passende Bodenbedeckung entworfen werden.

1.3 Forschungsfragen

- Wie kann man Farbwerte von Dach- und Fassadenflächen aus Texturen ableiten?
- Wie soll die Bodenbedeckung aussehen, damit sie optimal zu den mit Farbwerten versehenen Gebäuden passt?

1.4 Arbeitsfragen

- Welche Software beziehungsweise Programmiersprache eignet sich am besten, um eine automatisierte Extraktion von Farbwerten aus Texturen durchzuführen?
- Wie lässt sich die Farbwertberechnung optimieren, um die bestmögliche Performanz zu erreichen?
- Welche Datenquelle eignet sich am besten für die Visualisierung der Bodenbedeckung?

1.5 Methodische Vorgehensweise

Zu Beginn der Masterarbeit sollen im theoretischen Teil die grundlegenden theoretischen Aspekte von dreidimensionalen Stadtmodellen erarbeitet werden, um einen soliden Grundstock für die praktische Arbeit zu bekommen. Es sollen dabei neben CityGML auch andere Standards für 3D-Stadtmodelle erwähnt werden, wobei der Fokus klar auf CityGML liegt, da dieser bei der Stadt Wien Verwendung findet. Neben der Erläuterung von 3D-Stadtmodellen soll der Theorie-Teil auch Themen behandeln, welche nicht direkt Teil von 3D-Stadtmodellen sind, jedoch über die praktische Arbeit damit in Verbindung gebracht werden. So würde man zum Beispiel bei der Erklärung von CityGML nicht gleich über die Entstehung von Farben oder über Statistik schreiben, jedoch haben diese theoretischen Themen wohl ihre Relevanz für das bessere Verständnis des praktischen Teils und auch direkt für dessen Durchführung.

Am Beginn des Praktischen Teils steht die Suche nach geeigneten Programmbibliotheken, um nicht jede einzelne Funktion selbst programmieren zu müssen. Danach gilt es, Schritt für Schritt ein Skript aufzubauen und dieses auch laufend zu testen und dessen Ergebnisse zu evaluieren. Es wird dabei nach einzelnen Meilensteinen vorgegangen. So wird beispielsweise zunächst das 3D-Stadtmodell von Wien vom Skript ausgelesen und überprüft, ob diese Daten von Skript einwandfrei verarbeitet werden können. Danach kann dazu übergegangen werden, Farbwerte aus Texturen auszulesen und diese zu verarbeiten. Erst am Ende werden die berechneten Farbwerte wieder in das originale Stadtmodell geschrieben.

Nach dem Aufbau der automatisierten Farbwertextraktion gilt es - je nach Ergebnis - diverse Feinabstimmungen in den Algorithmen zur Farbwertberechnung vorzunehmen.

Zum Schluss wird eine Methode im Skript implementiert, mit der das gesamte dreidimensionale Stadtmodell von Wien in einem Zug mit neuen Farbwerten versehen werden kann.

Im Anschluss an die Farbwertberechnung wird nun ein passendes Design für die Bodenbedeckung entwickelt, um das in dieser Arbeit zu entwerfende Erscheinungsbild zu vervollständigen.

Da die Farbwerte der Gebäudeflächen von Texturen abgeleitet werden, soll auch bei der Bodenbedeckung Bezug auf Luftbilder genommen werden. Damit wird auch für die einzelnen Flächen der Bodenbedeckung ein passender Farbwert auf Basis eines Luftbildes gesucht. Im Unterschied zu den Gebäudeflächen werden die Flächen der Bodenbedeckung jedoch nicht individuell eingefärbt, sondern nach Klassen, wie zum Beispiel Wasserfläche, Straße, usw., mit dazu passenden Farbwerten versehen.

2. 3D-Stadtmodelle

Dieses Kapitel soll dem Leser dieser Masterarbeit 3D-Stadtmodelle etwas näherbringen und auch ein theoretisches Grundgerüst bilden, um den praktischen Teil dieser Arbeit besser verstehen zu können.

Zu Beginn werden verschiedene Standards für 3D-Stadtmodelle aufgezählt und erläutert, bevor dann CityGML als wichtigster Standard für diese Arbeit im Detail erklärt wird. Gegen Ende dieses Kapitels wird noch im Speziellen auf das 3D-Stadtmodell der Stadt Wien eingegangen und dabei dessen Inhalte und Funktionen beschrieben.

2.1. Standards und Formate

Weltweit gibt es zahlreiche Standards und Formate für 3D-Stadtmodelle für die unterschiedlichsten Anwendungsgebiete. Da für diese Arbeit nur der Standard CityGML relevant ist, dient dieses Unterkapitel nur als kurze Übersicht, welche Formate neben CityGML existieren. Da es vom Anwendungsgebiet abhängt, welche Formate und Standards relevant und welche weniger relevant sind, wurde hier vom Autor eine Auswahl der - seiner Meinung nach - bekanntesten Formate getroffen.

2.1.1. Standard versus Format

„Das Dateiformat legt fest, wie die innere Struktur einer Datei aussehen muss“ (E-teaching.org o.J.). Das heißt, dass ein Format (Dateiformat) nur Aufschluss über technische Details gibt. Der Begriff Standard dagegen müsste eher in den Bereich der Organisation eingeordnet werden, da hier Dateiformate vereinheitlicht werden oder zumindest bestimmt wird, welche Formate allgemein anerkannt werden. Oft werden auch beide Begriffe synonym verwendet. (vgl. Vogel IT-Medien GmbH 2021a)

2.1.2. Standards für 3D-Stadtmodelle

Warum existieren überhaupt so viele Standards für 3D-Stadtmodelle? Aufgrund der zahlreichen Anwendungsgebiete von 3D-Stadtmodellen und der Tatsache, dass kein Format perfekt für alle Anwendungen geeignet ist, gibt es auch zahlreiche Möglichkeiten, wie man Formate für die jeweiligen Anwendungsbereiche optimieren kann. Von verschiedenen Organisationen wird

dann in weiterer Folge versucht, die vielen Formate durch Standards, wie zum Beispiel CityGML, zu vereinheitlichen. Zusätzlich gibt es auch noch Firmen, welche ihre eigenen Formate entwickeln. Werden solche Formate oft genutzt, so werden diese auch de facto zu einem Standard. (vgl. ZLATANOVA et al. 2012: 3)

2.1.2.1. CityGML

CityGML ist ein Standard für 3D-Stadtmodelle, welcher von der SIG 3D 2005 entwickelt wurde und seit 2008 ein offizieller Standard des OGC ist. Der Standard basiert auf XML und ist ein Anwendungsprofil des XSD-Schemas GML. (vgl. COORS et al. 2016: 17)

Innerhalb von CityGML existieren mehrere Module, welche die verschiedenen Arten von Objekten (Gebäude, Vegetation, Gelände, etc.) in Klassen zusammenfassen. CityGML erlaubt nicht nur die Speicherung von geometrischen Informationen, sondern auch die Ablage semantischer Informationen. Daher wird CityGML vor allem auch dort verwendet, wo die Anwendungen über die bloße Visualisierung hinausgehen. (vgl. ZLATANOVA et al. 2012: 4)

2.1.2.2. CityJSON

CityJSON wurde entwickelt, um CityGML einfacher in JavaScript verwenden zu können. Es basiert auf JSON und kann über diverse Tools aus CityGML erzeugt werden. Nach Angaben der Entwickler ist CityJSON sechsmal kompakter als CityGML. Aus Gründen der Einfachheit wurden nur jene Objekte, welche am häufigsten verwendet werden, in die Definition von CityJSON von CityGML übernommen. (vgl. 3D geoinformation group o.J.)

2.1.2.3. Multipatch

Multipatch ist ein Typ von 3D-Objekten der Software ArcGIS von ESRI. Als Multipatch werden komplexere 3D-Objekte, wie zum Beispiel Gebäude oder Bäume, abgespeichert, welche über eine einfache Extrusion hinausgehen. (vgl. ESRI 2020a)

2.1.2.4. COLLADA

COLLADA ist ein XML-basiertes Format zum Transport von 3D-Objekten zwischen verschiedenen Anwendungen, welches von der Khronos Group entwickelt wurde. Es dient nicht nur zur Abspeicherung von Geometrien und Texturen, sondern auch zur Speicherung von Beleuchtung, Animationen und Einstellungen. (vgl. The Khronos Group Inc. 2021)

2.1.2.5. OBJ

Das OBJ Format wurde von der Firma Wavefront Technologies für die Anwendung Advanced Visualizer entwickelt. Es kann nur Geometriedaten und Texturkoordinaten speichern. Die Texturen selbst und Materialeigenschaften werden in einer separaten Datei mit der Endung MTL (Material Template Library) abgespeichert. (vgl. Aspose Pty Ltd 2021)

2.1.2.6. KML

Die Keyhole Markup Language ist ein von der Firma Keyhole entwickeltes Format, welches ebenfalls auf XML basiert. Es findet hauptsächlich bei Google Maps und Google Earth Verwendung und kann keine semantischen Informationen abspeichern. Es dient daher hauptsächlich der Visualisierung und Georeferenzierung von Objekten. (vgl. ZLATANOVA et al. 2012: 3)

Da eine Recherche ergeben hat, dass (zumindest in Europa) von vielen Städten das Format CityGML für 3D-Stadtmodelle verwendet wird, geht der Autor davon aus, dass es sich dabei um den führenden Standard für 3D-Stadtmodelle handelt. Diese subjektive Annahme stützt sich darauf, dass keine Informationen zu finden waren, welche ein Format als führendes oder meist genütztes deklarieren würden. Dahingehend wurde die Tatsache, dass viele europäische Städte, welche über ein öffentlich erwähntes 3D-Stadtmodell verfügen, CityGML verwenden, als Indiz gewertet, dass CityGML zumindest eines der am häufigsten verwendeten Formate ist. Daher wird auch in dieser Arbeit nur auf CityGML im Detail eingegangen, da dieses Format zudem auch von der Stadt Wien und somit auch vom Autor im praktischen Teil verwendet wird.

Es sei noch darauf hingewiesen, dass sich diese Arbeit auf die CityGML-Version 2.0 bezieht!

2.2. CityGML

2.2.1. Grundlagen

2.2.1.1. XML

XML steht für Extensible Markup Language und ist im Gegensatz zu den Programmiersprachen, welche bestimmte Abläufe durchführen, eine Auszeichnungssprache. Das bedeutet, dass mit XML keine Arbeitsabläufe durchgeführt, sondern lediglich Informationen gespeichert werden. Im Unterschied zu HTML, welches beschreibt, wie Daten aussehen, ist XML nur für die strukturierte und standardisierte Datenspeicherung zuständig. (vgl. Refsnes Data 2021a; Refsnes Data 2021b)

XML-Dokumente verfügen über eine einfache Baumstruktur. Zu Beginn kann optional ein Prolog stehen, welcher die XML-Version und die Zeichenkodierung angibt. Fehlt dieser Prolog, so wird UTF-8 als Default-Zeichenkodierung angenommen. Jedes XML-Dokument muss ein Wurzelement enthalten. Innerhalb dieses Elementes können sich weitere befinden. Beinhaltet diese Elemente wiederum andere Elemente, so sind dies die Eltern und die Elemente innerhalb Kinder. Mehrere Kinder sind zueinander Geschwister. In XML können Elemente beliebig ineinander verschachtelt werden. (vgl. Refsnes Data 2021c; Refsnes Data 2021d)

Folgendes Beispiel zeigt die Grundstruktur von XML-Dokumenten:

```
<?xml version="1.0" encoding="UTF-8"?>
<wurzel>
  <eltern>
    <kind></kind>
    <kind></kind>
    ...
  </eltern>
</wurzel>
```

Mit sogenannten Attributen können Eigenschaften eines Objektes festgelegt werden (vgl. Refsnes Data 2021e):

```
<element attribut="wert">Elementinhalt</element>
```

Das Auslesen von XML-Dokumenten wird *parsen* genannt. Ein *XML-Parser* liest Daten aus einer XML-Datei und wandelt diese in das gewünschte Format um, sodass sie mit einem anderen Programm weiterverarbeitet werden können. (vgl. Refsnes Data 2021f)

In Python kann man mit *minidom* oder *ElementTree* XML-Dateien parsen (vgl. Brain4ce Education Solutions Pvt. Ltd. 2021).

2.2.1.2. XSD

XSD steht für XML-Schema Definition und beschreibt die Struktur eines XML-Dokuments. Mit XSD kann man Angaben darüber machen, welche Objekte in welcher Form im XML-Dokument vorhanden sein dürfen. Somit kann man ein XML auf Validität prüfen.

Ein XML-Dokument gilt nämlich bereits als wohlgeformt, wenn dessen Syntax korrekt ist. Es können aber trotzdem noch Fehler enthalten sein, welche mitunter weitreichende Konsequenzen nach sich ziehen können. (vgl. Refsnes Data 2021g)

XSD-Schemata müssen im XML-Dokument referenziert werden. Die Syntax von XSD entspricht der von XML. Im Unterschied zu XML gibt es hier vorgeschriebene Bezeichnungen. (vgl. Refsnes Data 2021h)

`xmlns:xs="http://www.w3.org/2001/XMLSchema"` bedeutet, dass Elementen oder Datentypen, welche von *http://www.w3.org/2001/XMLSchema* stammen ein `xs:` vorangestellt werden muss. Somit kann man andere Schemata von denen von *w3* unterscheiden. Steht beim Datentyp zum Beispiel `xs:string`, so weiß man, dass es sich um den Datentyp *string* handelt, welcher von *w3* vorgegeben wird. Steht beim Datentyp allerdings zum Beispiel `TexCoordListType`, so erkennt man, dass es sich um einen speziellen Datentyp handelt, welcher vom Ersteller des Schemas definiert wurde. (vgl. Refsnes Data 2021i)

Simple Elemente enthalten nur Text, keine Attribute oder andere Elemente. In der XSD-Syntax sehen sie folgendermaßen aus: `<xs:element name="Elementname" type="Datentyp"/>` (vgl. Refsnes Data 2021i)

2.2.1.3. GML

Die *Geography Markup Language* ist ein Kodierungsstandard für XML-Dokumente, sozusagen ein XSD-Schema für geographische Daten (vgl. LAKE R. o.J.; COORS et al. 2016: 39). Dabei dient GML nicht dazu, um anzugeben, wie geographische Daten auf einer Karte aussehen sollen, sondern um Informationen über die Geometrie (Punkt, Linie, Fläche, usw.) und sonstige Eigenschaften abzuspeichern. Später kann aus solchen Daten dann automatisch eine Karte erzeugt werden. (vgl. LAKE R. o.J.)

CityGML ist nun eine Spezialform von GML, welche den Aufbau von dreidimensionalen Daten beschreibt. (vgl. COORS et al. 2016: 39)

2.2.2. Levels of Detail

In CityGML können mit den Levels of Detail verschiedene Detaillierungsgrade von 3D-Stadtmodellen abgebildet werden (vgl. COORS et al. 2016: 69). Oft werden die Levels of Detail von CityGML als generelles Synonym für Detaillierungsgrade von 3D-Stadtmodellen verwendet. Es gibt auch diverse Publikationen, wie zum Beispiel jene von BILJECKI 2017, welche sich mit Levels of Detail außerhalb von CityGML befassen. In *Kapitel 6.2. Vergleich verschiedener LOD-Konzepte für Gebäude* werden Levels of Detail in Bezug auf Texturen noch genauer behandelt. In diesem Abschnitt wird nur auf die Definition von Levels of Detail in CityGML Bezug genommen. Da sich das Level of Detail (LOD) in CityGML hauptsächlich auf Gebäude bezieht und auch dahingehend ausführlicher dokumentiert ist, wird später bei der Beschreibung der Gebäude in CityGML noch genauer auf das LOD von Gebäuden Bezug genommen.

CityGML unterscheidet fünf verschiedene Stufen des Level of Detail, LOD0, LOD1, LOD2, LOD3 und LOD4. Man muss sich dabei nicht entscheiden, in welchem LOD man ein Objekt darstellen möchte, da man in einem Modell verschiedene Versionen davon abspeichern kann. Je nach Anwendung kann man sich schlussendlich für die passendste Variante entscheiden. (vgl. COORS et al. 2016: 69)

Zurzeit können jedoch nur Modelle in den Stufen LOD 0, 1, und 2 flächendeckend für größere Gebiete angefertigt werden, da nur für diese automatische Verfahren zur Datenerfassung möglich sind. Für mehr Details müssen die Daten manuell erfasst werden und dies ist nicht mit vertretbarem Kostenaufwand für ganze Großstädte möglich. (vgl. COORS et al. 2016: 70)

2.2.2.1. LOD0 Landschaftsmodell

Beim LOD0 kommt nur ein Geländemodell (*siehe 2.2.3.3. Gelände*) als einziges dreidimensionales Objekt zur Anwendung. Alle anderen Objekte, wie zum Beispiel Gebäude, werden nur mit ihrem Grundriss flach dargestellt. Es geht beim LOD0 daher mehr um die Darstellung der reinen Lage von Objekten innerhalb einer Landschaft in dreidimensionaler Darstellung. (vgl. Open Geospatial Consortium 2012: 11; COORS et al. 2016: 70)

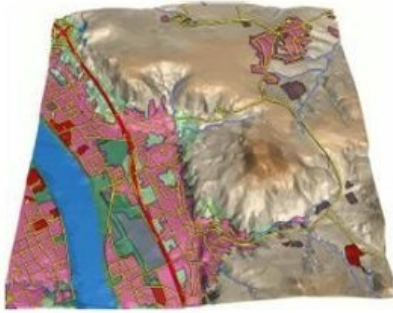


Abbildung 1: Level of Detail 0 (Quelle: nach IGG Uni Bonn o.J.).

2.2.2.2. LOD1 Klötzchenmodell

Das Level of Detail 1 bietet auch für andere Objekte als das Geländemodell eine Ausdehnung in die dritte Dimension. Gebäude werden hier nach ihrer Höhe extrudiert und es entsteht ein sogenanntes Klötzchenmodell. Stadtmöbel und Vegetation kommen im LOD1 auch vereinzelt vor, wenn diese von größerer Bedeutung sind. Für einige Analysezwecke ist dieser Detaillierungsgrad bereits ausreichend.

(vgl. Open Geospatial Consortium 2012: 11f; COORS et. al 2016: 70)

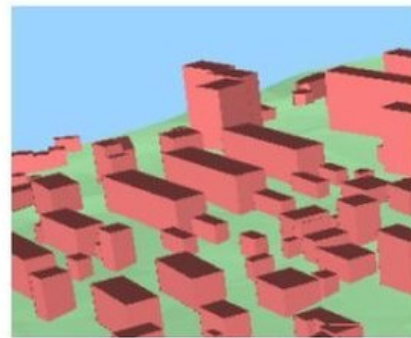


Abbildung 2: Level of Detail 1 (Quelle: nach IGG Uni Bonn o.J.).

2.2.2.3. LOD2 Dachmodell

LOD2 bietet mehr Details als LOD1, vor allem in Bezug auf die Dachflächen. So gibt es bei LOD2 bereits generalisierte Dachstrukturen. Es wird auch erwähnt, dass hier bereits Texturen zur Anwendung kommen können. Auch wird die Grundrissform detaillierter dargestellt als in LOD1. Vegetation und Stadtmöbel können laut dieser LOD-Variante bereits vollständig als Prototypen vorhanden sein.

(vgl. Open Geospatial Consortium 2012: 11f; COORS et al. 2016: 70)

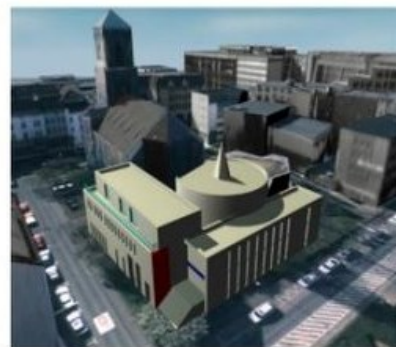


Abbildung 3: Level of Detail 2 (Quelle: nach IGG Uni Bonn o.J.).

2.2.2.4. LOD3 Detailmodell

Bei LOD3 kommen weitere Details, wie Türen, Fenster, Rauchfänge und Antennen, bei Gebäuden hinzu. Auch bei der Vegetation können mehr Details modelliert werden.

Aufgrund des hohen Detaillierungsgrades kommen Objekte in LOD3 eher in kleinräumigen Modellen zum Einsatz, da dessen Modellierung momentan nur von Hand möglich ist. (vgl. Open Geospatial

Consortium 2012: 11f; COORS et al. 2016: 70)



Abbildung 4: Level of Detail 3 (Quelle: nach IGG Uni Bonn o.J.).

2.2.2.5. LOD4 Innenraummodell

Beim Level of Detail 4 kommen bei den Gebäuden noch Innenräume als zusätzliche Teile hinzu. Diese können auch bereits mit Einrichtungsgegenständen ausgestattet sein. Auch andere Objekte wie Bäume oder Gewässer können in LOD4 detaillierter ausfallen als in LOD3. Allerdings kommen zu diesen keine weiteren Elemente hinzu – es wird nur die Geometrie detaillierter dargestellt. (vgl. Open Geospatial Consortium 2012: 11f; COORS et al. 2016: 70f)



Abbildung 5: Level of Detail 4 (Quelle: nach IGG Uni Bonn o.J.).

2.2.3. Module

Um CityGML-Dokumente übersichtlicher zu gestalten, bestehen diese aus einzelnen Modulen. Dadurch müssen nur jene Komponenten eines Stadtmodells in CityGML implementiert werden, welche auch tatsächlich benötigt werden. (vgl. Open Geospatial Consortium 2012: 11)

Es wird unter anderem zwischen folgenden Modulen unterschieden (Auswahl - der nach Meinung des Autors - wichtigsten Module):

- Geometrie

- Erscheinungsbild
- Gelände
- Landnutzung
- Gewässer
- Gebäude
- Vegetation
- Tunnel
- Brücken
- Verkehr
- Stadtmöbel

(vgl. Open Geospatial Consortium 2012: 17)

2.2.3.1. Geometrie

Da CityGML auf GML basiert, kommt in CityGML auch das Geometriemodell von GML zur Anwendung. Trotzdem können nicht alle Geometrien von GML 1:1 auf CityGML übertragen werden. Zunächst wird auf die Geometrietypen von GML eingegangen, bevor die Situation und mögliche Einschränkungen in CityGML beschrieben werden. (vgl. COORS et al. 2016: 62f)

Der grundlegendste Geometrietyp ist der Typ *DirectPosition*, welcher durch einen Tupel, bestehend aus zwei oder drei Koordinaten (zwei- oder dreidimensional), repräsentiert wird. Auf diesen Typ bauen alle weiteren Geometrietypen auf. (vgl. COORS et al. 2016: 45)

Im Folgenden werden die grundlegendsten Geometrietypen aufgelistet und kurz beschrieben:

Punkte:

- ***Point***: dieser Geometrietyp besteht aus genau einem Koordinatenelement *DirectPosition*.
- ***PointGrid***: bei diesem Typ handelt es sich um die Anordnung von Punkten in einem Gitter, wie es beispielsweise bei einem Geländemodell in Form eines TIN der Fall ist.

(vgl. COORS et al. 2016: 45)

Linien:

- ***LineString***: wird eine Folge von Stützpunkten vom Typ *Point* mit Linien zwischen den Punkten verbunden, so handelt es sich um einem *LineString*.

- ***Arc***: dieser Typ stellt eine Kurve dar, welche aus drei Punkten konstruiert wurde.
- ***ArcByBulge***: im Unterschied zum Typ *Arc* wird diese Kurve aus Radius und Mittelpunkt konstruiert.
- ***ArcString***: stellt ein Segment dar, welches aus mehreren Kurven zusammengesetzt wurde.
- ***Geodesic***: Linien vom Typ *Geodesic* bilden den kürzesten Weg zwischen zwei Punkten auf der gekrümmten Erdoberfläche.
- ***GeodesicString***: verbindet man mehrere Linien vom Typ *Geodesic*, so erhält man einen *GeodesicString*.
- ***OffsetCurve***: um eine parallel zu einer Kurve verlaufende Linie zu erstellen, benötigt man eine *OffsetCurve*.
- ***OrientableCurve***: eine *OrientableCurve* ist eine Kopie einer Kurve, welche durch ein positives Vorzeichen zu einer identischen Kopie und durch ein negatives Vorzeichen zu einer Kopie mit umgekehrter Richtung wird.

(vgl. COORS et al. 2016: 46)

Flächen:

- ***SurfaceType***: dieser Geometrietyp ist das flächenhafte Pendant zum *LineString*.
- ***AbstractSurfacePatch***: entsteht, wenn man mehrere Elemente vom Typ *SurfaceType* zusammensetzt.
- ***AbstractParametricCurveSurface***: mit diesem Typ kann man gekrümmte Oberflächen darstellen.
- ***OrientableSurface***: mit einer *OrientableSurface* kann man durch Vorzeichen wie bei der *OrientableCurve* die Richtung einer Oberfläche umkehren.
- ***TriangulatedSurface***: erstellt man aus einem *PointGrid* eine Oberfläche mit Dreiecksvermaschung, so entsteht eine *TriangulatedSurface*.
- ***TIN***: dies ist ein Spezialfall der *TriangulatedSurface* mit dem Unterschied, dass hier die Delaunay-Triangulation zum Einsatz kommt.

(vgl. COORS et al. 2016: 47ff)

Körper:

- **Solid:** ist der einfachste voluminöse Typ und setzt sich aus mehreren Oberflächen zusammen, welche allerdings zwingend einen geschlossenen Körper ergeben müssen und mit ihrer Oberseite nach außen orientiert sein müssen.

(vgl. COORS et al. 2016: 50f)

Die Kombination aus mehreren Objekten der vorhin genannten Geometrietypen kann entweder *Composite*-Objekte oder *Multi*-Objekte enthalten. Wichtig bei den Typen *CompositeCurve*, *CompositeSurface*, *CompositeSolid* ist, dass sie direkt aneinander anschließen müssen. Das heißt, dass sich bei einer *CompositeCurve* die benachbarten Kurven einen Punkt teilen müssen. Das Gleiche gilt für *CompositeSurface* mit einer gemeinsamen Kurve und *CompositeSolid* mit einer gemeinsamen Fläche. (vgl. COORS et al. 2016: 52)

Im Gegensatz zu *Composite*-Objekten müssen *Multi*-Objekte nicht direkt miteinander verbunden sein und können auch ohne Inhalt bestehen. Daraus ergeben sich dann die Geometrietypen *MultiPoint*, *MultiSurface* und *MultiSolid*. (vgl. COORS et al. 2016: 52f)

Nun gibt - es wie bereits angekündigt - gewisse Einschränkungen, was die Verwendung von GML-Geometrien in CityGML betrifft. Konkret bedeutet das, dass in CityGML nur geradlinige Objekte zur Anwendung kommen dürfen. Das heißt, es dürfen keine Kurven und gekrümmten Flächen verwendet werden. (vgl. COORS et al. 2016: 62f)

2.2.3.2. Erscheinungsbild

In CityGML wird das Erscheinungsbild durch das *Appearance*-Modul festgelegt. Es bietet die Möglichkeit, das Aussehen von Objekten oder ganzen Modellen unabhängig von deren Geometrie festzulegen. Dies bietet den Vorteil, dass für ein und dasselbe Modell verschiedene Szenen je nach Anwendungsgebiet erstellt werden können. So können mit denselben Geometrien zum Beispiel das Solarpotential und die Lärmausbreitung neben einer fotorealistischen Darstellung visualisiert werden. (vgl. COORS et al. 2016: 72f)

In CityGML gibt es zwei grundlegende Arten der Darstellung:

- X3D-Material
- Textur

(vgl. COORS et al. 2016: 74)

X3D-Material

Beim X3D-Material sind für das Erscheinungsbild eines Objektes hauptsächlich das Material, welches die Oberflächeneigenschaften beinhaltet, und die Farbe verantwortlich (vgl. COORS et al. 2016: 76).

Die Eigenschaften der Oberfläche geben an, wie das auftreffende Licht reflektiert oder ob sogar eigenes Licht ausgesendet wird (vgl. COORS et al. 2016: 76).

Folgende Eigenschaften der Oberfläche werden unterschieden:

- *Ambient Light*: Licht, das von keiner bestimmten Lichtquelle kommt, vergleichbar mit den Lichtverhältnissen bei bewölktem Himmel.
- *Diffuse Light*: Licht von einer bestimmten Lichtquelle und einem Material, das das Licht zerstreut (normale Oberfläche).
- *Specular Light*: Licht von einer bestimmten Lichtquelle, welches auf eine spiegelnde Oberfläche trifft.

(vgl. COORS et al. 2016: 77)

In der folgenden Tabelle sind alle weiteren Eigenschaften aufgelistet, welche ein Material neben der Oberflächenbeschaffenheit und der Farbe noch annehmen kann:

Tabelle 1: Mögliche Materialeigenschaften für das X3D-Material (Quelle: Coors et al. 2016: 81).

Element	Bedeutung
EmissiveColor	Grundfarbe der Oberfläche
AmbientIntensity	Mindestanteil der Grundfarbe
DiffuseColor	Farbe des zerstreut reflektierten Lichtes
SpecularColor	Farbe des direkt reflektierten Lichtes
Shininess	Schärfe des Highlights
IsSmooth	True = Gouraud Shading, False = Flat Shading
Transparency	Transparenzgrad

Textur

Zusätzlich zum Material gibt es auch noch die Möglichkeit, einem CityGML-Modell eine Textur zuzuweisen. Dabei wird jeder Fläche ein eigenes Bild zugeordnet. In der CityGML-Datei selbst wird nur die Referenz zu einer Textur und nicht die Textur selbst abgespeichert. (vgl. COORS et al. 2016: 83)

Das Zuordnen eines Bildes (Textur) zu einem 3D-Objekt nennt man *Texture Mapping*. Beim Texture Mapping wird eine zweidimensionale Oberfläche sozusagen um ein dreidimensionales Objekt gewickelt. Bei 3D-Stadtmodellen kann das Texture Mapping dazu dienen, eine Oberfläche mit Details zu versehen, die man sonst nur mit viel Modellierungsaufwand erzeugen könnte. (vgl. Tech Target 2021)

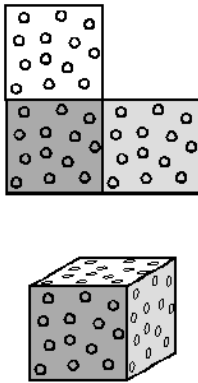


Abbildung 6: Texture Mapping (Quelle: Tech Target 2021).

Abbildung 7 zeigt die verschiedenen Möglichkeiten, wie Texturen in CityGML auf Objekten abgebildet werden können:

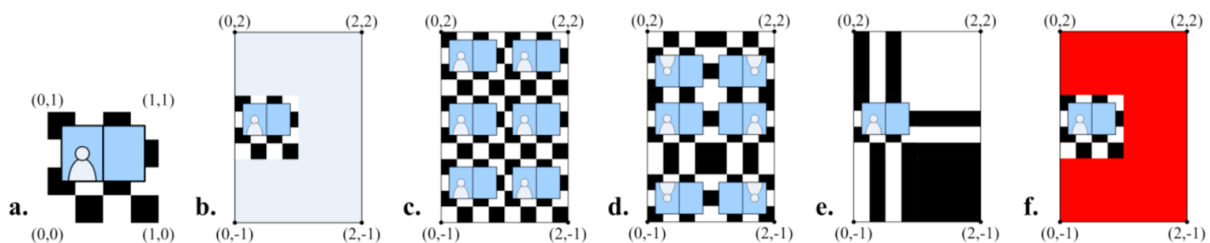


Abbildung 7: Wrap-Mode: Textur (a), none (b), wrap (c), mirror (d), clamp (e), border (f) (Quelle: Open Geospatial Consortium 2012: 38).

Grundsätzlich gibt es in CityGML zwei Arten von Texturen:

- *georeferenzierte Textur*:

Die georeferenzierte Textur wird gleichmäßig über das gesamte Modell gelegt und benötigt eine Georeferenzierung (zum Beispiel georeferenziertes Luftbild) (vgl. Open Geospatial Consortium 2012: 39).

- *parametrisierte Textur*:

Bei der *parametrisierten Textur* wird nur ein Teil der Textur auf einen definierten Teil des zu texturierenden Objektes gelegt. Für diese Auswahl werden in der CityGML-Datei Koordinaten abgespeichert, welche die Auswahl des gewünschten Bereichs eines Bildes ermöglichen. (vgl.

Open Geospatial Consortium 2012: 40f) Dieses Texturierungsverfahren kommt auch bei der Stadt Wien zum Einsatz und wird im Kapitel 8.2. *Ausgangsdaten* noch genauer beschrieben.

2.2.3.3. Gelände

In den meisten Fällen reicht bei 3D-Stadtmodellen ein flaches Gelände nicht aus, um eine Stadt realitätsnahe darstellen zu können. In solchen Fällen wird dem Stadtmodell ein Gelände-Modul hinzugefügt. (vgl. COORS et al. 2016: 101)

Beim Gelände-Modul gibt es in CityGML vier verschiedene Typen (mit dazugehörigen Datentypen):

- TIN Relief: *TriangulatedSurfaceType*
- Raster Relief: *RectifiedGridCoverage*
- Mass Point Relief: *MultiPointType*
- Breakline Relief: *MultiCurveType*

(vgl. COORS et al. 2016: 101)

TIN Relief

TIN ist die Abkürzung für Triangulated Irregular Network und bedeutet so viel wie unregelmäßiges Netzwerk aus Dreiecken. Bei der Delaunay-Triangulation werden lose Höhenpunkte so miteinander verbunden, sodass in den Umkreisen der Dreiecke keine anderen Punkte zu liegen kommen. Der Vorteil eines TIN liegt darin, dass die Dichte der verwendeten Höhenpunkte je nach Komplexität des Geländes variiert werden kann. (vgl. COORS et al. 2016: 102)

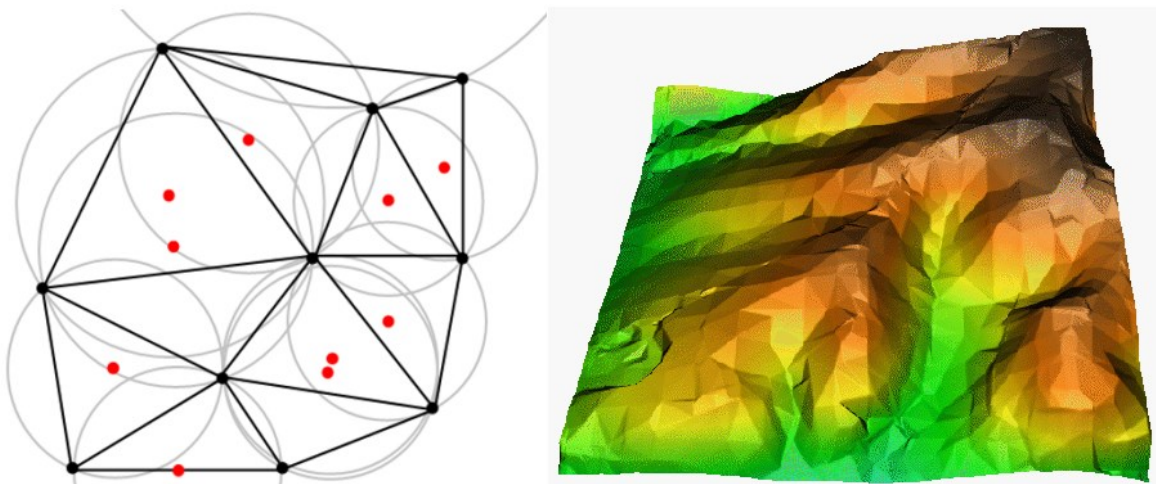


Abbildung 8: TIN: links in 2D mit Umkreisen der Dreiecke und rechts in 3D mit höhenabhängiger Einfärbung (QGIS o.J.).

Raster Relief

Bei diesem Relief-Typ sind die Höhenpunkte in Form eines regelmäßigen Rasters angeordnet. Es können „handelsübliche“ Rasterdateien verwendet werden, wie man sie zum Beispiel beim Download der SRTM-Daten bekommt. Diese sind häufig im Format GeoTIFF vorhanden. (vgl. COORS et al. 2016: 101f)

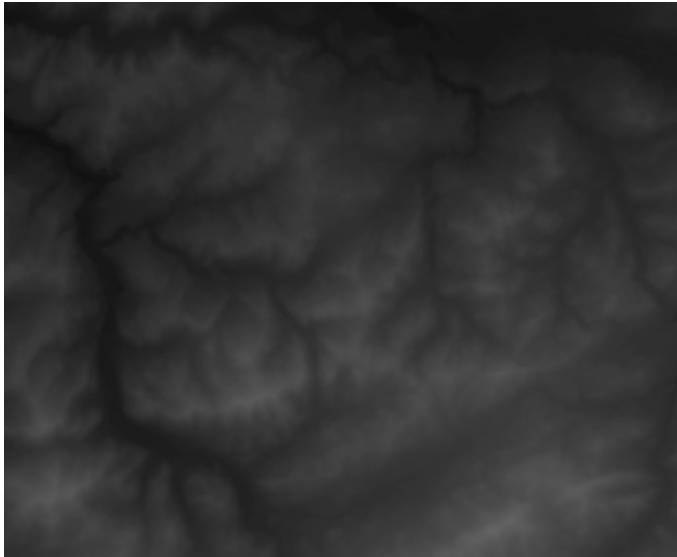


Abbildung 9: Raster Höhenmodell aus SRTM-Daten (Quelle: USGS o.J.).

Mass Point Relief

Ein Mass Point Relief besteht aus einer unregelmäßigen Ansammlung von sogenannten Masspunkten. Diese können zum Beispiel von einer Laserscan-Befliegung stammen. (vgl. COORS et al. 2016: 102)

Break Line Relief

Das Break Line Relief wird normalerweise nicht als eigenständiges Geländemodell verwendet. Es dient meist nur der Verfeinerung von bestehenden Geländemodellen wie Raster oder TIN. Es bestehen zwei Varianten des Break Line Relief:

- Harte Geländekanten
- Weiche Geländekanten oder meist Flusstäler

(vgl. COORS et al. 2016: 102)

2.2.3.4. Bodenbedeckung

Die Bodenbedeckung dient dazu, dem Gelände ein realitätsnahes Aussehen zu verleihen, falls dazu keine Textur (georeferenzierte Textur) verwendet wird. Etwas verwirrend scheint die Originalbezeichnung *landuse*-Modul, was wörtlich übersetzt nur auf die Nutzung und nicht auf die tatsächliche Bodenbedeckung hinweisen würde. Für die Darstellung selbst ist nur die tatsächliche Bodenbedeckung von Relevanz. Da CityGML auch dafür gedacht ist, ein semantisches 3D-Stadtmodell zu bieten, kann die Nutzung eines Stückes Land trotzdem von Interesse sein. (vgl. COORS et al. 2016: 115)

Grundsätzlich ist das *landuse*-Modul für eine flächendeckende Verwendung vorgesehen. Da jedoch auch Module für Gewässer und Vegetation existieren, kann man das *landuse*-Modul auch mit anderen Modulen kombinieren. (vgl. COORS et al. 2016: 115)

Die Bodenbedeckungsflächen haben alle die Geometrie *MultiSurface* und müssen zwingend dreidimensionale Koordinaten aufweisen. Es ist auch möglich, den einzelnen Dreieckssegmenten eines *TIN* Bodenbedeckungsklassen zuzuweisen. (vgl. COORS et al. 2016: 115)

Bei der Bodenbedeckung wird zwischen folgenden Attributen unterschieden:

- *class* (Siedlungsareal, Verkehrsfläche, Vegetationsfläche, Wasser, nicht entwickelte fläche)
- *function* (Wohnbebauung, Industrie-/Gewerbegebiete, Grünland, Ackerland, Moor, Heide und Gehölz)
- *usage* (siehe *function*)

Zu beachten ist, dass für *function* und *usage* die gleichen Werte vorgesehen sind. *Class* bezieht sich dabei eher auf die Art der Bodenbedeckung selbst und *function* beziehungsweise *usage* eher auf die Nutzung. (vgl. COORS et al. 2016: 115f)

2.2.3.5. Gewässer

Obwohl Gewässer auch in die Bodenbedeckung integriert werden können, gibt es für diese trotzdem ein eigenes Modul. Mit diesem werden alle Gewässer (Flüsse, Kanäle, Bäche, Seen, etc.) in der Klasse *AbstractWaterObjectType* zusammengefasst und als dreidimensionale Körper und Oberflächen modelliert. Das Gewässermanual in CityGML dient hauptsächlich der Visualisierung von verschiedenen Wasserständen. Hydrologische Abflusssimulationen sind damit zurzeit leider noch nicht möglich. (vgl. COORS et al. 2016: 293f)

2.2.3.6. Gebäude

Das Gebäudemodell zählt zu den umfangreichsten Modulen von CityGML. Besonders hier kommen die vier Levels of Detail zur Anwendung und können auch als Qualitätsmerkmal von 3D-Stadtmodellen herangezogen werden, wobei Modelle mit geringerem LOD nicht immer schlechter sind, vor allem dann nicht, wenn sie ihren Anwendungszweck einwandfrei erfüllen. (vgl. Open Geospatial Consortium 2012: 62)

Gebäude bestehen in CityGML grundsätzlich aus der Oberklasse *AbstractBuilding*, welche die Klasse *Building* beinhaltet. Die Klasse *Building* kann dann weiter in einzelne Gebäudeteile (*BuildingPart*) unterteilt werden. Jene Eigenschaften (Attribute), welche alle Teile eines Gebäudes betreffen, werden in der Klasse *AbstractBuilding* gespeichert. (vgl. COORS et al. 2016: 131)

Folgende Attribute kann ein Gebäude in der Klasse *AbstractBuilding* enthalten:

- *class*: Art eines Gebäudes, wie zum Beispiel administrativ, Kultur, Sport, usw.
- *function*: bezeichnet den ursprünglichen Zweck, für welchen ein Gebäude gebaut wurde.
- *usage*: im Gegensatz zu *function* bezieht sich *usage* auf die tatsächliche Nutzung.
- *yearOfConstruction*: gibt das Baujahr eines Gebäudes an.
- *yearOfDemolition*: für historische Analysen kann auch das Jahr des Abrisses von Bedeutung sein.
- *roofType*: dieses Attribut gibt an, welche Form das Dach eines Gebäudes besitzt. Durch diese Angabe kann ein Dach aus einer Liste von 16 prototypischen Dächern gewählt und daraufhin automatisch erzeugt werden.
- *measuredHeight*: gemessene Gesamthöhe eines Gebäudes.
- *storeysAboveGround*: Anzahl der oberirdischen Stockwerke.
- *storeysBelowGround*: Anzahl der unterirdischen Stockwerke.
- *storeyHeightsAboveGround*: Höhe der oberirdischen Stockwerke.
- *storeyHeightsBelowGround*: Höhe der unterirdischen Stockwerke.

(vgl. COORS et al. 2016: 132f)

Die nachfolgende Abbildung (*Abbildung 10*) zeigt verschiedene Dachformen, wie sie auch in ähnlicher Form in CityGML definiert sind.

DACHFORMEN

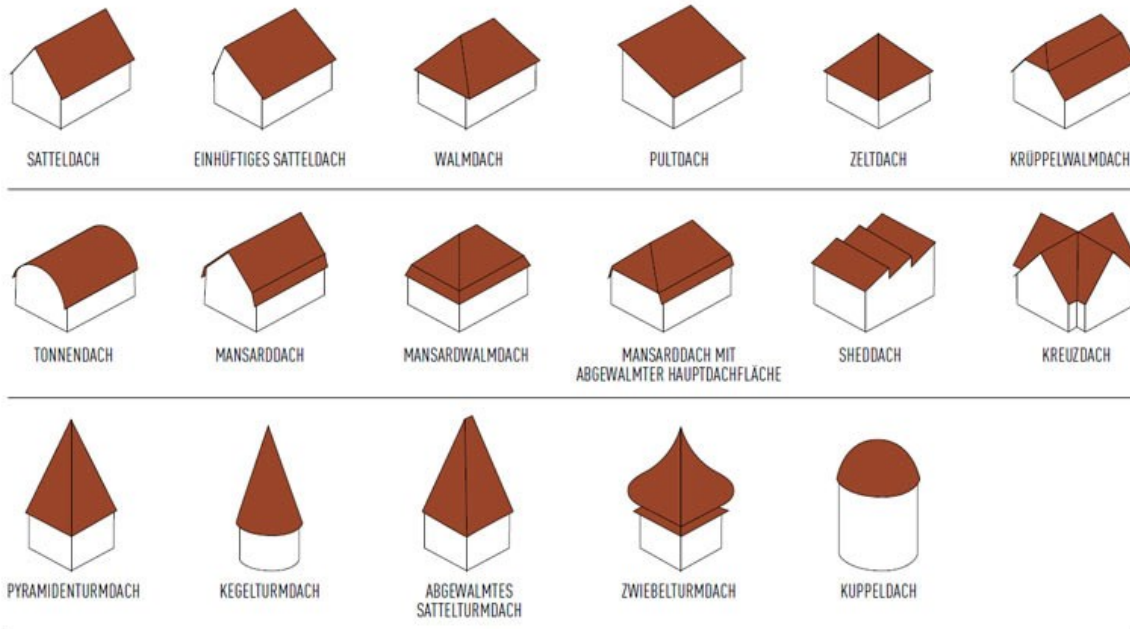


Abbildung 10: Arten von Dachformen (Quelle: Prefa 2021).

Gebäude in LOD0

Das LOD0 stellt, wie bereits in 2.2.2.1. *LOD 0 Landschaftsmodell* erwähnt, nur das Gelände in dreidimensionaler Form dar. Gebäude werden entweder als Fußabdruck am Boden oder mit der Umrissskante aus der Luft dargestellt. (vgl. COORS et al. 2016: 138)

Gebäude in LOD1

Im Level of Detail 1 werden die zweidimensionalen Gebäudeumrisse durch einen Höhenwert oder die Kenntnis über die Anzahl der Geschosse in die dritte Dimension erweitert, sozusagen extrudiert. Das Dach bleibt dabei flach. Oft werden in dieser Detailstufe kleinere Details des Gebäudeumrisses wie Kanten oder Erker nicht dargestellt. (vgl. COORS et al. 2016: 140)

Als Geometrie kommen bei den LOD1 Gebäuden *MultiSurface* und *Solid* in Frage. Da *MultiSurface*-Geometrien nicht garantieren, dass ein Gebäude eine geschlossene Geometrie aufweist, ist es jedoch ratsam, nur *Solid*-Geometrien zu verwenden. Diese setzen sich wiederum aus einer *CompositeSurface*-Geometrie zusammen, welche keine Überlappungen aufweisen

darf. Um als *Solid* gültig zu sein, darf diese als zusätzliche Bedingung keine Lücken aufweisen (siehe 2.2.3.1. *Geometrie*). (vgl. COORS et al. 2016: 141)

Wird bei einem LOD1-Stadtmodell ein Relief verwendet, so müssen die Gebäude an das Relief angepasst werden. In der Regel geschieht dies automatisch. Möchte man bei Gebäuden eine genauere Anpassung an das Gelände erzielen, so kann man auch explizit eine sogenannte *TerrainIntersectionCurve* modellieren. Diese gibt genau an, wo das Gebäude in das Gelände übergehen soll. (vgl. COORS et al. 2016: 141f)

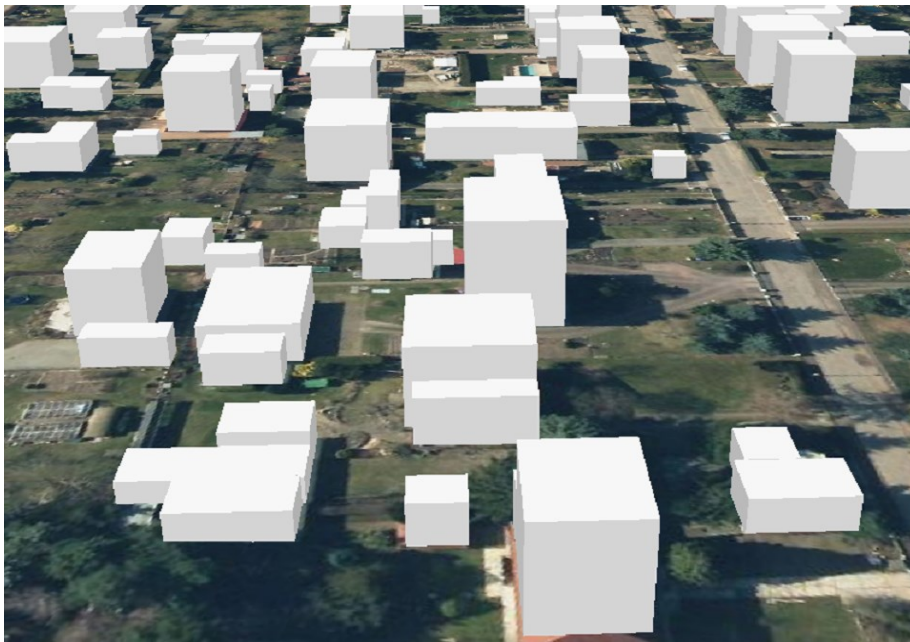


Abbildung 11: LOD1 Stadtmodell Leipzig (Quelle: Stadt Leipzig 2021a).

Gebäude in LOD2

Gebäudemodelle in LOD2 weisen bereits Elemente wie Erker oder Balkone auf, welche in LOD1 noch nicht berücksichtigt wurden. Auch die Dachformen aus [Abbildung 10](#) kommen hier zum Einsatz. (vgl. COORS et al. 2016: 167)

Trotz der zunehmenden Details sind Stadtmodelle in LOD2 noch automatisiert ableitbar. Darum können diese auch (im Gegensatz zu LOD3 und LOD4) für ganze Großstädte flächendeckend erzeugt werden. Dies geschieht meistens über Laserscanbefliegungen in Kombination mit einem Kataster, aus dem die Grundflächen entnommen werden. Bei der Datenerfassung mittels Laserscan entstehen Punkte mit dreidimensionalen Koordinaten. Diese ergeben ein Modell der Erdoberfläche, welches auch Gebäude und Bäume beinhaltet. Durch Weiterverarbeitung des Oberflächenmodells kann ein Geländemodell erzeugt werden, bei welchem alle

Objekte, die aus dem Gelände herausragen, entfernt werden. Aus der Differenz zwischen Oberflächenmodell und Geländemodell können in weiterer Folge Gebäudehöhen abgeleitet werden. Generell liefert das Oberflächenmodell auch Daten, aus denen für LOD2 relevante Gebäudedetails extrahiert werden können. (vgl. COORS et al. 2016: 167f)

Für LOD2 Gebäude können neben den Geometrischen Details auch semantische Details über den Aufbau des Gebäudes definiert werden. Diese sind die Randflächen Dach, Fassade und Grundfläche. Sind Gebäude an einer Seite offen, so können sie durch *ClosureSurface*-Objekte geschlossen werden. (vgl. COORS et al. 2016: 170)



Abbildung 12: LOD2 Stadtmodell Leipzig (Quelle: Stadt Leipzig 2021b).

Gebäude in LOD3

Die Detailstufe LOD3 ist die höchste Detailstufe für den Außenbereich von 3D-Stadtmodellen. Mehr Details bietet nur mehr LOD4 mit der Modellierung des Innenraums. Charakteristisch für LOD3 ist, dass im Unterschied zu LOD2 auch Gebäudeöffnungen, wie Fenster und Türen, modelliert werden. Neben diesen können je nach Belieben noch mehr Details hinzugefügt werden. (vgl. COORS et al. 2016: 200)

Fenster, Türen und andere Gebäudeöffnungen dienen dabei nicht nur als optische Gestaltungsmittel, sondern können beispielsweise für die Berechnung der Energiebilanz eines Gebäudes von entscheidender Bedeutung sein. (vgl. COORS et al. 2016: 201)

Aufgrund des hohen Detaillierungsgrades von LOD3-Stadtmodellen ist eine flächendeckende Modellierung nur mit erheblichem Arbeits- und daher auch Kostenaufwand möglich. Daher findet man LOD3-Stadtmodelle bis jetzt hauptsächlich in kleinräumigen, meist projektbezogenen Visualisierungen. (vgl. COORS et al. 2016: 200)



Abbildung 13: LOD3 Stadtmodell Leipzig (Quelle: Stadt Leipzig 2021c).

Gebäude in LOD4

Das Gebäudemodell in LOD4 bildet schließlich auch Innenräume eines Gebäudes ab. Bei dieser Detaillierungsstufe liegt der Anwendungszweck klar im semantischen Bereich, da für die Visualisierung von Innenräumen hauptsächlich CAD-Software verwendet wird. Ein Beispiel für eine Anwendung wäre die Ausbreitung von Rauchgasen bei einem Brand. Für eine solche Anwendung ist die Modellierung von Öffnungen und Verbindungen innerhalb eines Gebäudes besonders wichtig. (vgl. COORS et al. 2016: 218)

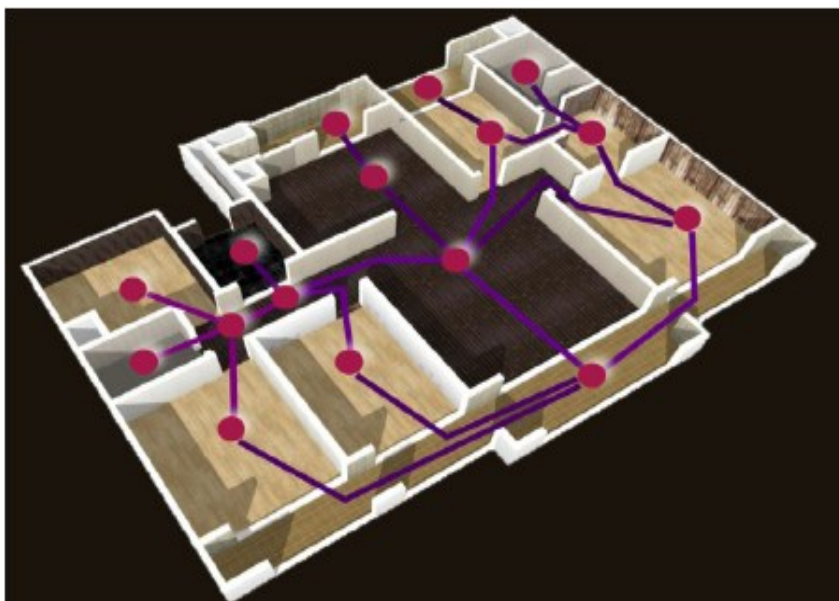


Abbildung 14: IndoorGML (Quelle: Open Geospatial Consortium o.J.).

Zusätzlich zu den vorhin beschriebenen LODs gibt es noch eine erweiterte LOD-Definition von BILJECKI 2017, welche den LODs 1 bis 3 noch weitere Abstufungen verleiht:

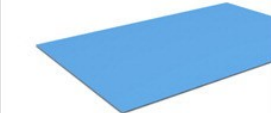
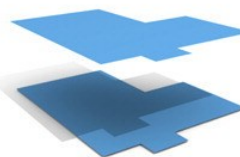
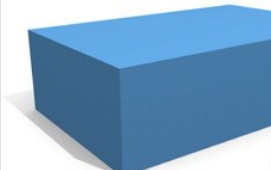
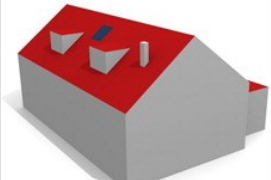
	LOD x.0	LOD x.1	LOD x.2	LOD x.3
LOD0	 LOD0.0	 LOD0.1	 LOD0.2	 LOD0.3
LOD1	 LOD1.0	 LOD1.1	 LOD1.2	 LOD1.3
LOD2	 LOD2.0	 LOD2.1	 LOD2.2	 LOD2.3
LOD3	 LOD3.0	 LOD3.1	 LOD3.2	 LOD3.3

Abbildung 15: LOD-Matrix von Biljecki (Quelle: Biljecki 2017: 76).

2.2.3.7. Vegetation

Wie alle anderen Objektarten in CityGML dient auch die Vegetation nicht nur dem Erscheinungsbild von 3D-Stadtmodellen, sondern hat auch wichtige Aufgaben für diverse Analysen und bietet zudem semantische Informationen (vgl. COORS et al. 2016: 229).

Die Oberklasse *AbstractVegetationObject* beinhaltet zwei verschiedene Klassen:

- *SolitaryVegetationObject*: einzelne Vegetationsobjekte
- *PlantCover*: flächendeckende Vegetation

(vgl. Open Geospatial Consortium 2012: 132)

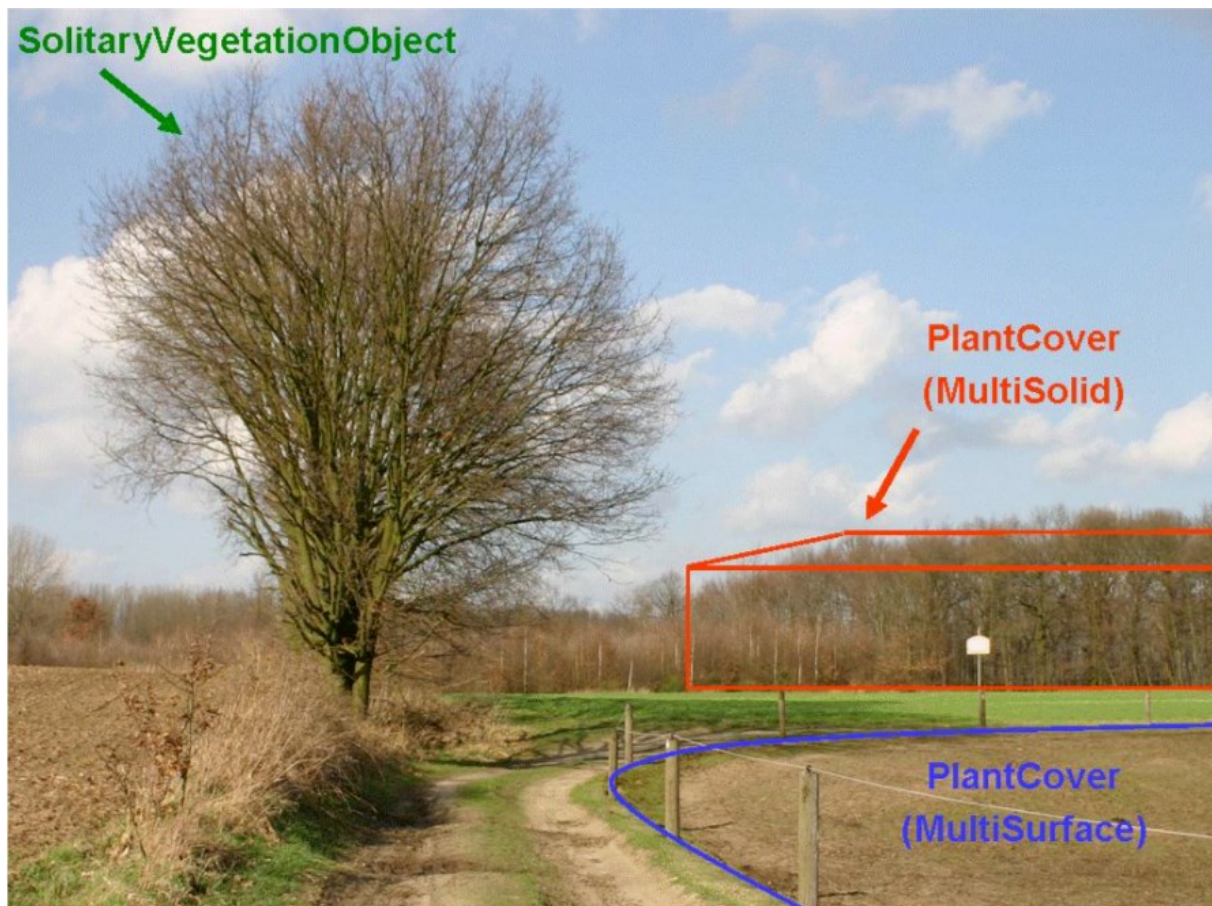


Abbildung 16: Arten von Vegetation (Quelle: Open Geospatial Consortium 2012: 132).

Einzelne Vegetationsobjekte

Einzelne Vegetationsobjekte, wie zum Beispiel Bäume oder Sträucher, werden durch folgende Attribute näher beschrieben:

- *class*: generelle Art von Vegetation, wie Baum, Strauch, Gras, usw.
- *height*: Höhe des Vegetationsobjektes.
- *trunkDiameter*: Stammdurchmesser.
- *crownDiameter*: Kronendurchmesser.
- *species*: genaue botanische Pflanzenart.

(vgl. COORS et al. 2016: 243)

Für das Erscheinungsbild werden folgende Arten von Geometrien unterschieden:

- Implizite Geometrie:

Als implizite Geometrien werden Objekte bezeichnet, welche eine fixe Form aufweisen und nur mehr in ihrer Größe und Ausrichtung verändert werden können. Solche Geometrien werden häufig von großen Modelldatenbanken bezogen. (vgl. COORS et al. 2016: 242f)



Abbildung 17: Detailliertes Baummodell als implizite Geometrie (Quelle: Evermotion o.J.).

- Fächergeometrie:

Eine Fächergeometrie ist eine Geometrie, welche von mehreren Einzelflächen gebildet wird. Von der Seite aus betrachtet entsteht dabei ein dreidimensionaler Eindruck. (vgl. COORS et al. 2016: 245, 248)

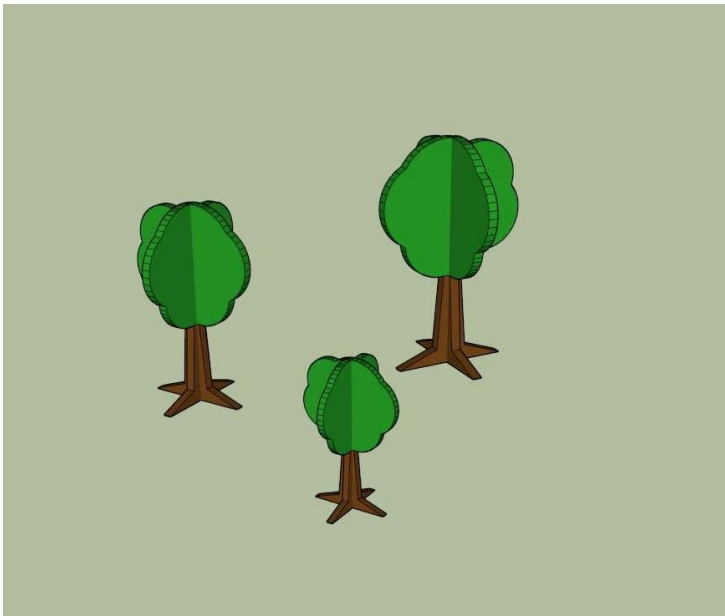


Abbildung 18: Schematische Darstellung von Fächergeometrien (Quelle: SACH 2018).

- Analytische Geometrie:

Eine Analytische Geometrie zeichnet sich dadurch aus, dass für ihre Erstellung Attribute über Form und Größe verwendet werden. So kann zum Beispiel für einen Nadelbaum ein Kegel verwendet werden, dessen Höhe und Durchmesser sich aus der Kronenhöhe und dem Kronendurchmesser ergeben. (vgl. COORS et al. 2016: 251)

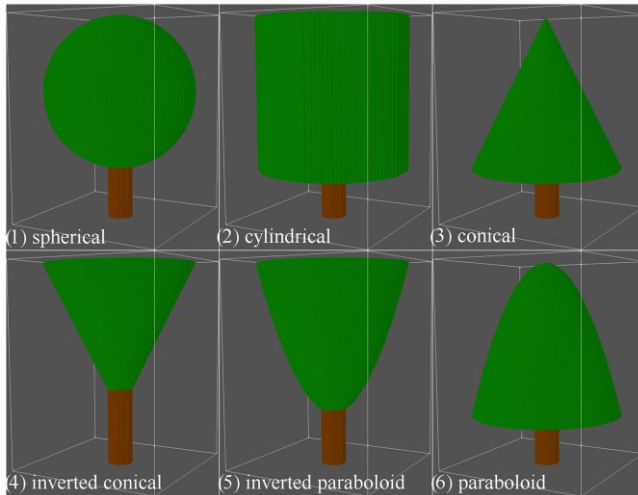


Abbildung 19: Analytische Geometrien (Quelle: HELDENS et. al. 2020).

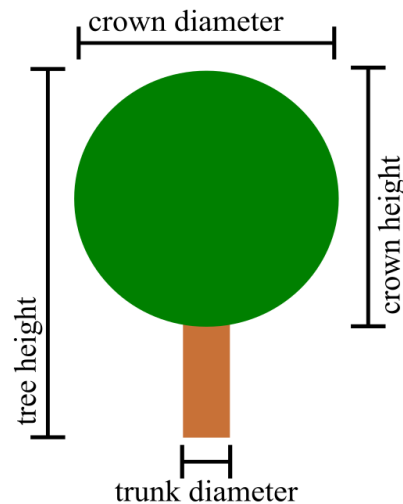


Abbildung 20: Parameter Analytischer Geometrien (Quelle: HELDENS et. al. 2020).

Flächendeckende Vegetation

Die Bodenbedeckung bezeichnet Vegetation, welche flächenhaft in Erscheinung tritt. Dazu gehören zum Beispiel Wiesen und Äcker sowie in Form von Waldflächen auftretende Bäume. Je nachdem, wie hoch die Vegetationsfläche im Durchschnitt ist, kann sie als Fläche oder Körper dargestellt werden. (vgl. Open Geospatial Consortium 2012: 132)

Des Weiteren kann die Geometrie der Bodenbedeckung dem jeweiligen Level of Detail angepasst werden (vgl. COORS et al. 2016: 231).

Folgende Attribute können für die Bodenbedeckung vergeben werden:

- *class*
- *function*
- *usage*
- *averageHeight*

2.2.3.8. Tunnel

Bei Tunnels gibt es - genauso wie bei den Gebäuden - eine Einteilung in vier Levels of Detail (vgl. Open Geospatial Consortium 2012: 82). Im Gegensatz zu den Gebäuden ist bei Tunnels das LOD0 nicht nutzbar. Daher können Tunnels erst mit einfachen dreidimensionalen Geometrien in LOD1 modelliert werden. (vgl. COORS et al. 2016: 288)

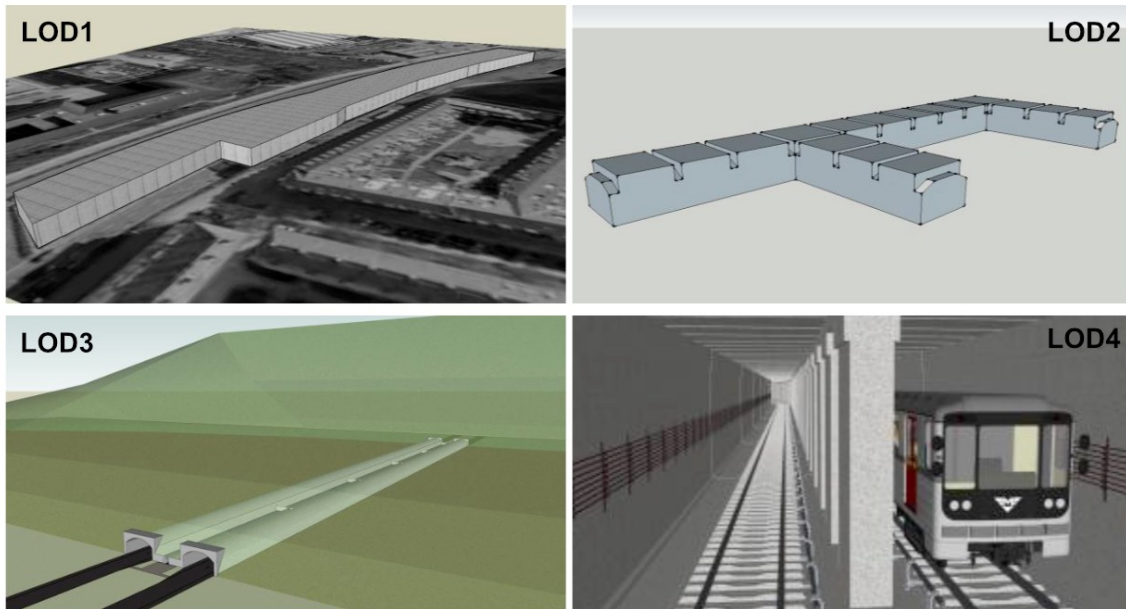


Abbildung 21: Tunnel in den Levels of Detail 1 bis 4 (Quelle: nach Open Geospatial Consortium 2012: 82).

Folgende Attribute können für Tunnels vergeben werden:

- *class*: Art des Tunnels
 - *traffic*: Verkehrstunnel
 - *supply*: Versorgungstunnel
 - *historical*: historischer Tunnel
- *function*: ursprünglicher Zweck, für den ein Tunnel gebaut wurde
 - *railway tunnel*: Eisenbahntunnel
 - *roadway tunnel*: Straßentunnel
 - *canal tunnel*: Kanaltunnel
- *usage*: tatsächliche Nutzung eines Tunnels (gleiche Kategorien wie bei *function*)
 - *railway tunnel*
 - *roadway tunnel*
 - *canal tunnel*

(vgl. COORS et al. 2016: 289)

2.2.3.9. Brücken

Auch bei den Brücken kommen nur die Levels of Detail 1 bis 4 zur Anwendung (vgl. Open Geospatial Consortium 2012: 99).

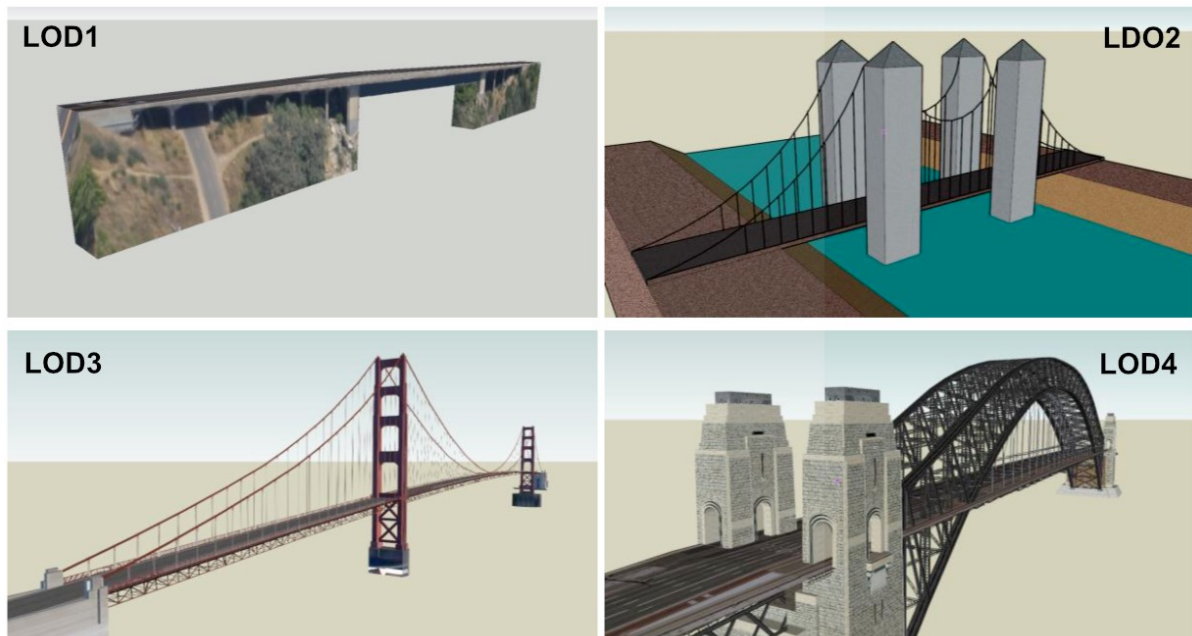


Abbildung 22: Brücken in den Levels of Detail 1 bis 4 (Quelle: nach Open Geospatial Consortium 2012: 99).

Folgende Attribute können für Brücken vergeben werden:

- *class*: Art der Brücke
 - *arced bridge*: Bogenbrücke
 - *cable-stayed bridge*: Schrägseilbrücke
 - *deck bridge*: Deckbrücke
 - *cable-stayed overpass*: Schrägseilüberführung
 - *truss bridge*: Fachwerkbrücke
 - *pontoonbridge*: Pontonbrücke
 - *suspension bridge*: Hängebrücke
- *function*: ursprünglicher Zweck, für den eine Brücke gebaut wurde:
 - *railway bridge*: Eisenbahnbrücke
 - *roadway bridge*: Straßenbrücke
 - *cable link*: Leitungsbrücke
 - *canal bridge*: Kanalbrücke
 - *aqueduct*: Aquädukt
 - *foot bridge*: Fußgängerbrücke

- *usage*: tatsächliche Nutzung einer Brücke (gleiche Kategorien wie bei *function*)
 - *railway bridge*
 - *roadway bridge*
 - *cable link*
 - *canal bridge*
 - *aqueduct*
 - *foot bridge*
- *yearOfConstruction*: Baujahr
- *yearOfDemolition*: Abrissjahr
- *isMoveable*: bewegliche Brücke

(vgl. COORS et al. 2016: 282f, 443)

2.2.3.10. Verkehr

Im Verkehrsmodell von CityGML können Verkehrswege als Linien und Flächen modelliert werden. Die Linien weisen dabei einen topologischen Zusammenhang auf und können daher für Routinganwendungen verwendet werden. Mit dem Attribut *class* wird zwischen den verschiedenen Verkehrsarten wie Schienenverkehr, Straßenverkehr und Schiffsverkehr unterschieden. Hier findet sich auch eine Unterscheidung in privat, öffentlich, zivil und militärisch. (vgl. COORS et al. 2016: 296ff)

Das Verkehrsmodell unterscheidet sich auch je nach verwendetem Level of Detail. Bei LOD0 werden Verkehrswege nur als Linien modelliert. In LOD1 können sie bereits als Flächen modelliert werden und ab LOD2 sind auch Hilfsflächen, wie zum Beispiel Verkehrsinseln vorgesehen. (vgl. COORS et al. 2016: 298f)

2.2.3.11. Stadtmöbel

Als Stadtmöbel werden fest installierte Gegenstände im Urbanen Raum bezeichnet. Dazu gehören Laternen, Bänke, Verkehrszeichen usw. (vgl. COORS et al. 2016: 302)

Die Hauptanwendung solcher Elemente ist die Erhöhung der Detaillichte von detaillierten 3D-Stadtmodellen und dient somit dem Erscheinungsbild. Neben dieser Anwendung können Stadtmöbel auch beispielsweise dazu dienen, um als Hindernisse für Schwertransporte sichtbar zu sein. Für selbstfahrende Autos sind Verkehrszeichen sogar ein sehr wichtiger Bestandteil. (vgl. COORS et al. 2016: 302f)

2.3. Das 3D-Stadtmodell von Wien

In diesem Unterkapitel wird die Ausgangssituation des 3D-Stadtmodells von Wien beschrieben, bevor es im Zuge dieser Masterarbeit erweitert wird.

Die in diesem Abschnitt gezeigten Visualisierungen des 3D-Stadtmodells von Wien stammen von einem Viewer der Firma *Virtual City Systems* (vgl. Stadt Wien o.J.a; Virtual City Systems o.J.a).

2.3.1. LOD-Matrix der Stadt Wien

Die Stadt Wien besitzt ihre eigene LOD-Matrix, welche eine Erweiterung der bestehenden Matrix von BILJECKI 2017 darstellt. Konkret bezieht sich diese Erweiterung auf überbaute Flächen.

In der folgenden Abbildung sind in dieser alle von der Stadt Wien verwendeten Levels of Detail hervorgehoben:

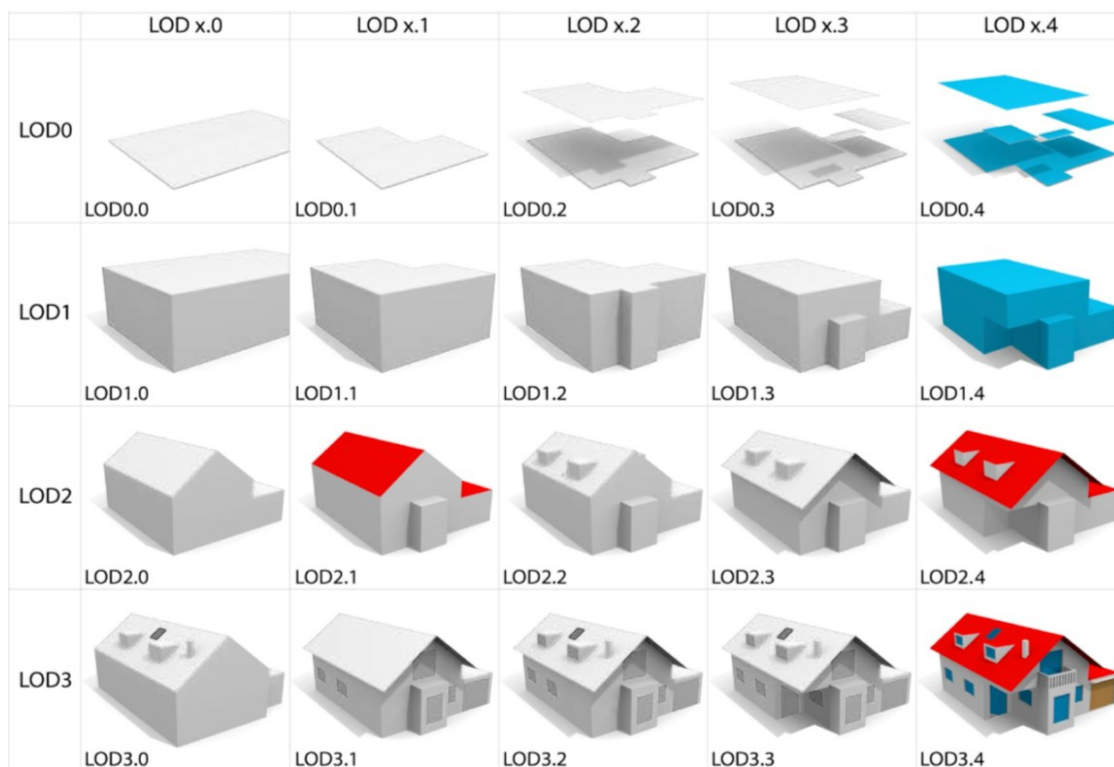


Abbildung 23: LOD-Matrix der Stadt Wien (Quelle: nach LEHNER UND DORFFNER 2020).

2.3.2. Bestandteile

Dieser Abschnitt beinhaltet alle in Wien vorhandenen Komponenten eines 3D-Stadtmodells – unabhängig davon, ob sie öffentlich verfügbar sind oder nicht. In den Fällen, in denen auch die Informationen nicht öffentlich verfügbar sind, werden diese mit *Stadt Wien intern* gekennzeichnet.

2.3.2.1. LOD1.4 Baukörpermodell

Das Baukörpermodell basiert auf den photogrammetrisch und terrestrisch erfassten Gebäuden von Wien, welche auch mit Höheninformationen ausgestattet sind. Diese Höheninformation kann dazu benützt werden, um die Gebäude je nach Höhe zu extrudieren. Dadurch entsteht ein dreidimensionales Gebäudemodell aus Prismen, welches dem in CityGML definierten LOD1 entspricht. (vgl. Stadt Wien o.J.b)

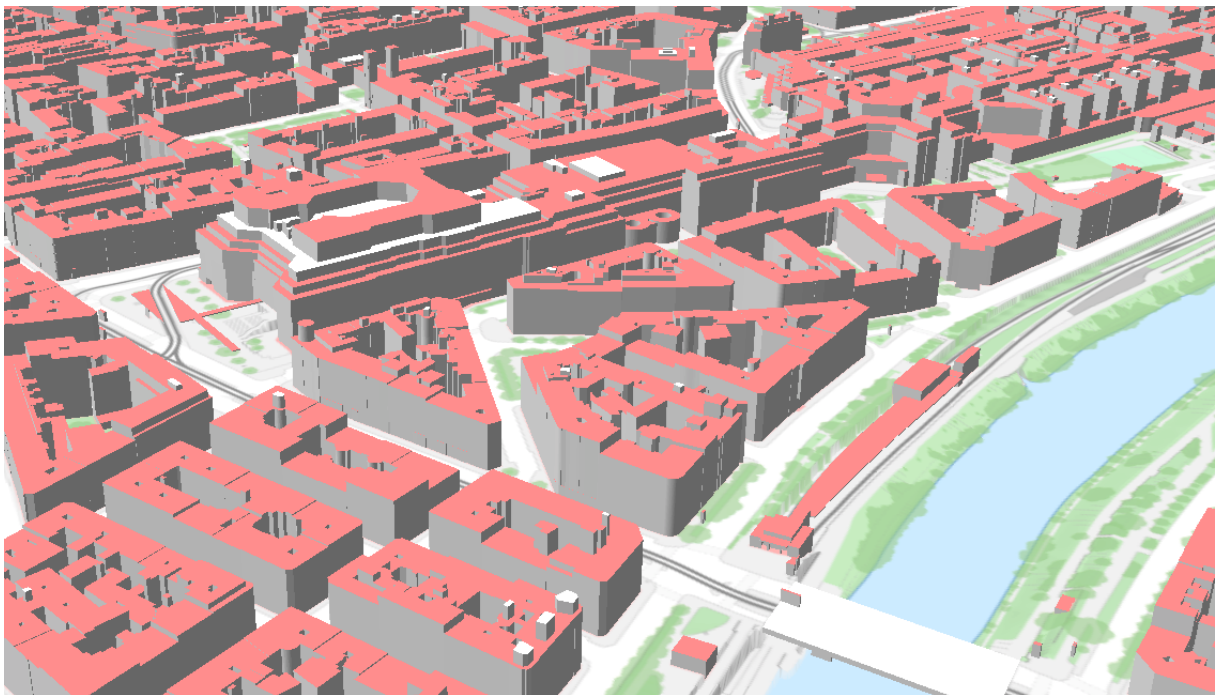


Abbildung 24: Baukörpermodell (Quelle: Stadt Wien o.J.a).

2.3.2.2. LOD2.1 Gebäudemodell

Das LOD2.1 Gebäudemodell beinhaltet alle Gebäude Wiens mit generalisierten Dachformen. Es wurde von der Firma *Virtual City Systems* mit ihrer eigenen Software *Building Reconstruction* erstellt. Diese Software benützt als Ausgangsdaten die Ergebnisse von 3D-Laserscan-Befliegungen und eine Datenbank mit allen gängigen Dachformen. Daraus wird automatisiert ein

Gebäudemodell erstellt, welches dem Detaillierungsgrad LOD2 von CityGML entspricht. (vgl. Virtual City Systems o.J.b)

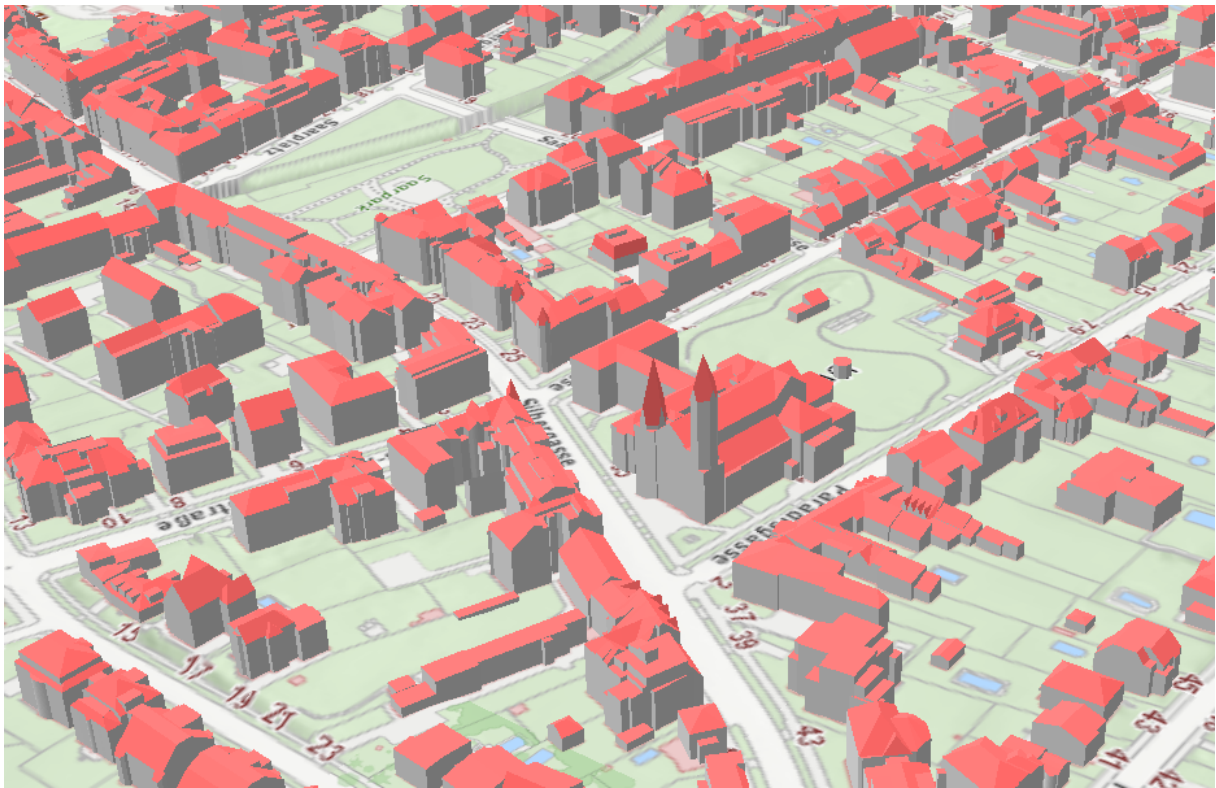


Abbildung 25: LOD2.1 Gebäudemodell (Quelle: Stadt Wien o.J.f).

2.3.2.3. LOD2.4 Gebäudemodell

Für die Wiener Innenstadt wurden die Dachflächen der Gebäude in detaillierterer Form modelliert. Dies geschieht manuell mit der Software *City Grid* der Firma *UVM-Systems* (vgl. UVM Systems o.J.a). Das LOD2.4 Gebäudemodell kann auch im offiziellen 3D-Stadtplan von Wien betrachtet werden (vgl. Stadt Wien o.J.a).

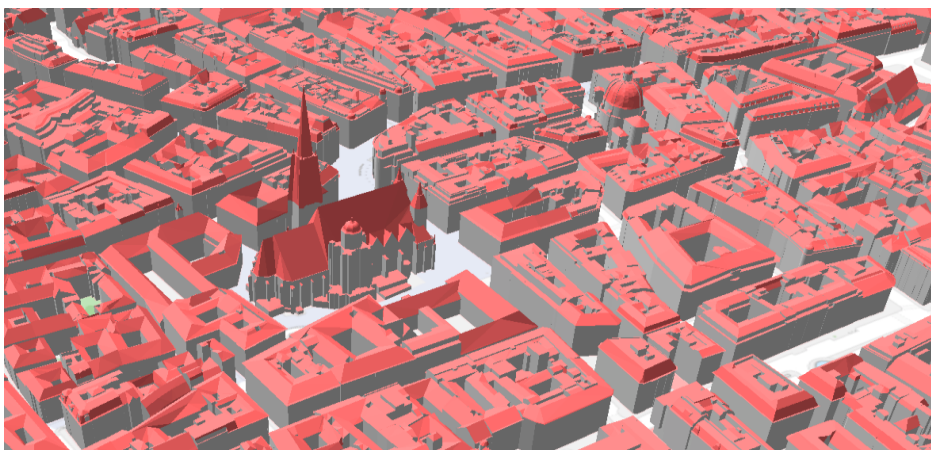


Abbildung 26: LOD2.4 Gebäudemodell (Quelle: Stadt Wien o.J.a).

2.3.2.4. LOD3.4 Gebäudemodell

Projektbezogen wurde für manche Bereiche Wiens ein detailliertes Gebäudemodell in LOD3.4 angefertigt. Aktuell existieren solche Modelle nur für die Bereiche rund um den Karlsplatz und entlang des Donaukanals beim Schwedenplatz. (vgl. Stadt Wien o.J.g)



Abbildung 27: LOD3.4 Gebäudemodell (Quelle: Stadt Wien o.J.g).

2.3.2.5. Brücken

Auch Brücken wurden manuell für das 3D-Stadtmodell modelliert. (vgl. Stadt Wien o.J.a)



Abbildung 28: Brücken im 3D-Stadtmodell von Wien (Quelle: Stadt Wien o.J.a).

2.3.2.6. Bodenbedeckung

Für die Darstellung der Bodenbedeckung existieren momentan ein Layer mit einem Orthofoto und ein Layer, welcher vom Design her der Standarddarstellung der *basemap.at* gleicht, jedoch keine Brücken aufweist, da diese in dreidimensionaler Form dargestellt werden. (vgl. Stadt Wien o.J.a; Stadt Wien und Österreichische Länder 2016)



Abbildung 29: Bodenbedeckung mit Orthofoto (Quelle: Stadt Wien o.J.a).



Abbildung 30: Bodenbedeckung mit Grundkarte, welche ein ähnliches Design wie die Standardkarte von *basemap.at* aufweist (Quelle: Stadt Wien o.J.a).

2.3.2.7. Geländemodell

Das digitale Geländemodell von Wien besteht aus Airborne Laserscanning Daten und anderen, photogrammetrisch erfassten Geländeobjekten. Es bildet die Basis für alle weiteren Komponenten des 3D-Stadtmodells, welche auf dieses aufgesetzt werden. (vgl. Stadt Wien o.J.h)

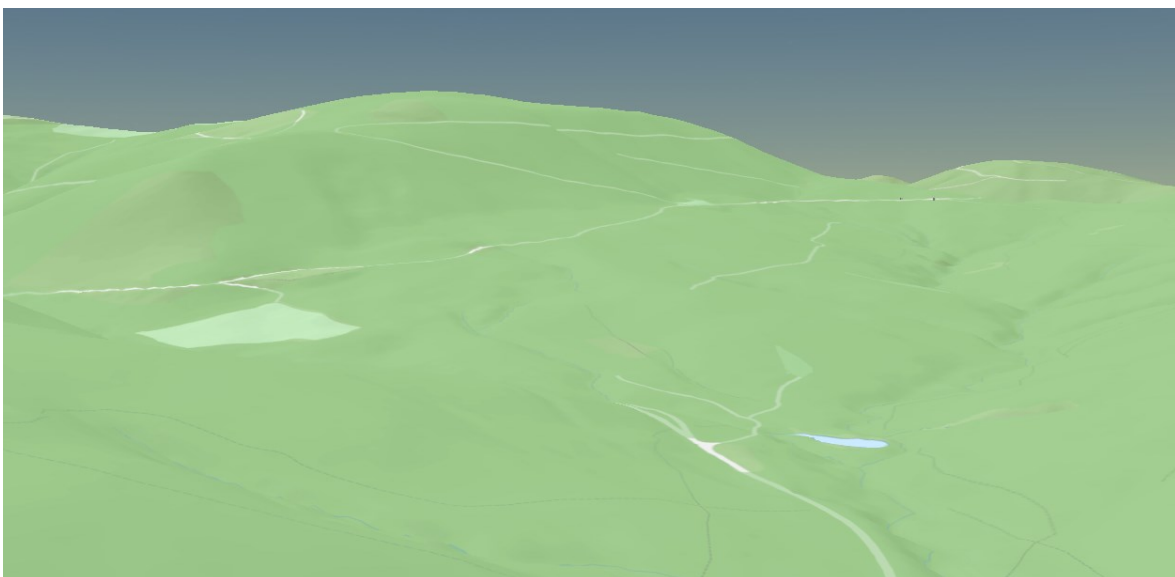


Abbildung 31: 3D-Geländemodell von Wien (Quelle: Stadt Wien o.J.a).

2.3.2.8. Tunnel der Wiener Linien

Für die Wiener Linien wurde von der Firma *UVM Systems* ein 3D-Modell aller U-Bahn-Tunnel auf Grundlage von Planunterlagen erstellt (vgl. UVM Systems o.J.d).

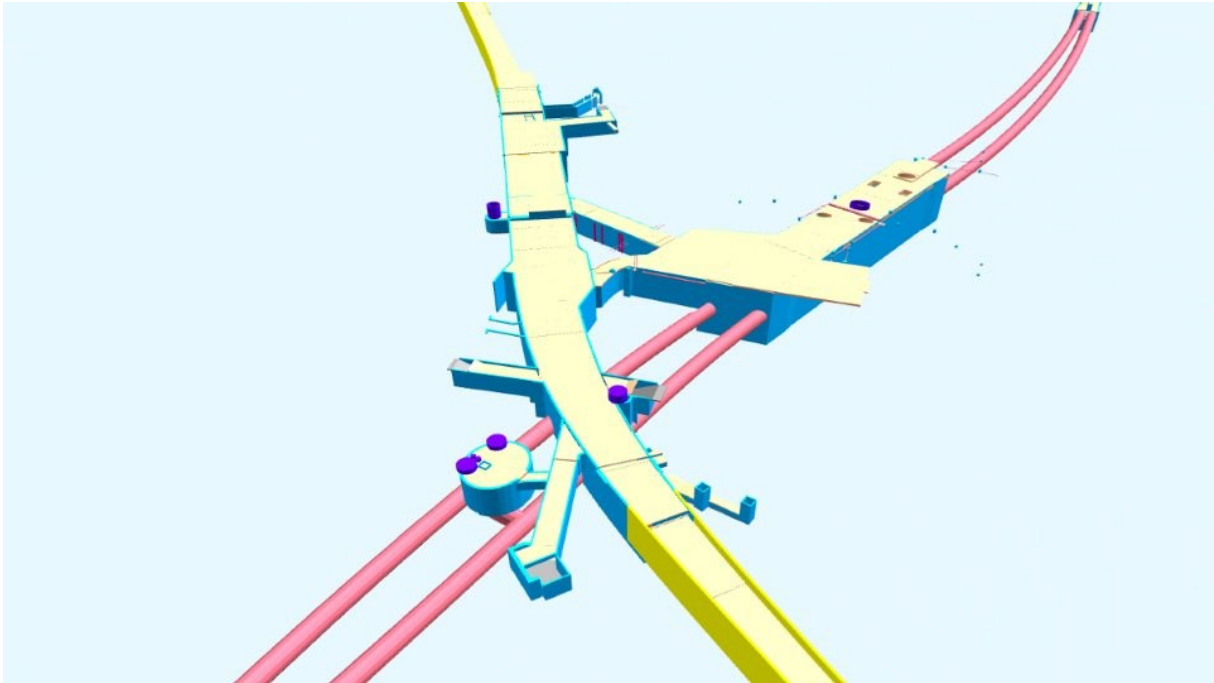


Abbildung 32: 3D-Modell einer U-Bahnstation in Wien (Quelle: UVM-Systems o.J.d).

2.3.3. Software

Dieser Abschnitt beschreibt kurz die wichtigsten Softwareprodukte, welche zur Erstellung und Bearbeitung des 3D-Stadtmodells von Wien zum Einsatz kommen.

2.3.3.1. FME

FME (Feature Manipulation Engine) ist eine Software der Firma *Safe Software* zur Konvertierung und Verarbeitung von Geodaten. Besonders gut geeignet ist sie für die Erstellung von automatisierten Arbeitsabläufen. (vgl. Safe Software Inc 2021).

Von der Stadt Wien wird diese Software für viele Vorgänge in der Erstellung und Bearbeitung von 3D-Geodaten genutzt. Unter anderem werden damit auch die als OGD frei verfügbaren Daten wie zum Beispiel das Baukörpermodell aufbereitet. (vgl. Stadt Wien intern)

2.3.3.2. Building Reconstruction

Building Reconstruction ist eine Software der Firma *Virtual City Systems*, welche es erlaubt, aus 3D-Laserscan-Daten und einer Datenbank mit gängigen Dachformen generalisierte 3D-Gebäudemodelle zu erzeugen, welche in etwa einem Detaillierungsgrad von LOD2 entsprechen (vgl. Virtual City Systems o.J.b).

Von der Stadt Wien wird diese Software nicht direkt eingesetzt, da das Gebäudemodell direkt von *Virtual City Systems* im Auftrag der Stadt Wien erstellt wird (vgl. LEHNER UND DORFFNER 2020: 66).

2.3.3.3. Dt Master

Die Software Dt Master (digital terrain master) dient zur manuellen Modellierung und Bearbeitung von digitalen Geländemodellen (vgl. Trimble Inc. 2021). Mit ihr werden in Wien Geländekanten, Dämme, Einschnitte, etc. manuell nachbearbeitet, da diese im originalen Geländemodell nicht so gut zur Geltung kommen (vgl. Stadt Wien intern).

2.3.3.4. City Grid

City Grid ist ein Softwarepaket der Firma *UVM Systems*, welches laut Angaben des Herstellers den gesamten Softwarebedarf für 3D-Stadtmodelle abdeckt (vgl. UVM Systems o.J.c).

Die Stadt Wien verwendet *City Grid* für die Modellierung der LOD3 Dachflächen (vgl. LEHNER UND DORFFNER 2020: 66).

3. Farben

In diesem Kapitel werden die wichtigsten Aspekte zum Thema Farbe behandelt, da Farben eine wichtige Rolle in dieser Masterarbeit einnehmen.

Zu Beginn wird die Entstehung von Farben erklärt. Im Anschluss daran wird erläutert, wie man Farben in Farbsysteme einteilen und somit reproduzierbar machen kann.

3.1. Entstehung von Farben

Damit Farben überhaupt existieren können, benötigen sie Licht. Genauer gesagt sind Farben eine Erscheinungsform von Licht. (vgl. WÄGER 2017: 63)

Licht kann auf zwei verschiedene Arten beschrieben werden: in Teilchen- und in Wellenform (vgl. LIEBMANN und WELSCH 2018: 278). Der genaue Unterschied zwischen diesen beiden Betrachtungsformen von Licht ist für diese Arbeit nicht von Relevanz. Wichtig für die Entstehung von Farben ist die Betrachtung von Licht als elektromagnetische Wellen.

Alle in der Natur vorkommenden elektromagnetischen Wellen werden als elektromagnetisches Spektrum bezeichnet (vgl. Sun.org GmbH o.J.). Ein Teil dieses elektromagnetischen Spektrums ist das sichtbare Licht mit Wellenlängen von zirka 380 bis 780 Nanometern. Diese Farben werden als Spektralfarben bezeichnet. (vgl. WÄGER 2017: 66f; Heinze GmbH o.J.).

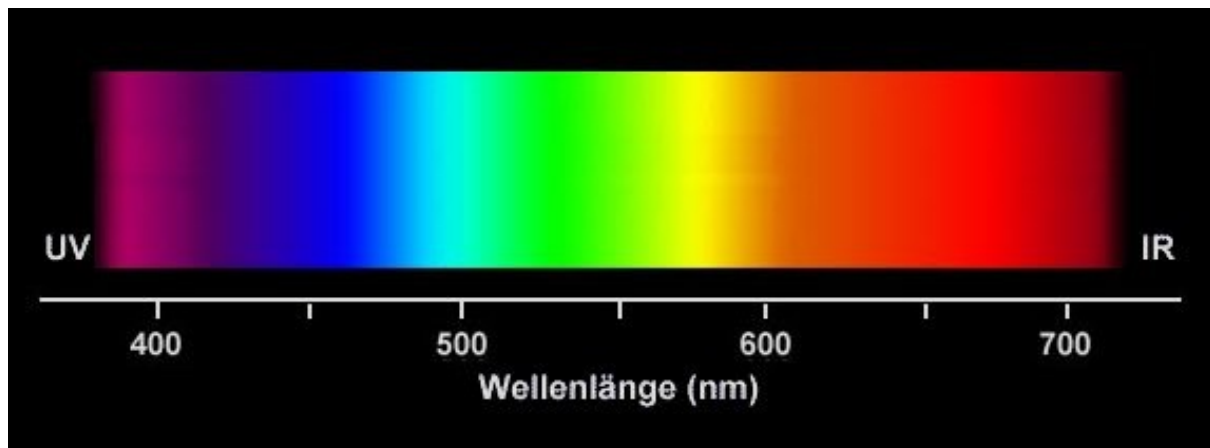


Abbildung 33: Spektrum des sichtbaren Lichts (Quelle: SCHÜNGEL 1999).

Das natürliche Sonnenlicht wird auch als weißes Licht bezeichnet. Es erscheint unserer Wahrnehmung so, da sich unterschiedliche Wellenlängen (daher unterschiedliche Farben) zusammen zu Weiß vermischen (siehe 3.2. Lichtfarben). Wären die Wellenlängen nicht so ausgewogen verteilt und würden in einem gewissen Bereich häufiger vorkommen, so würde das Licht in einer bestimmten Farbe erscheinen. Kerzenlicht zum Beispiel wirkt auf uns Menschen gelblich,

da hier jene Wellenlängen dominieren, welche von Menschen als gelb wahrgenommen werden.
(vgl. WÄGER 2017: 70f; Heinze GmbH o.J.)



Abbildung 34: Sonnenlicht (Quelle: Pxhere o.J.a).



Abbildung 35: Kerzenlicht (Quelle: Pxhere o.J.b).

Farbiges Licht kann man auch bewusst herbeiführen, wenn man weißes Licht durch einen farbigen Filter scheinen lässt. Ein Farbfilter ist ein lichtdurchlässiger Gegenstand, welcher uns Menschen in der zu erzeugenden Farbe erscheint. Nicht nur der Gegenstand erscheint uns farbig, sondern auch das weiße Licht nimmt dessen Farbe an, nachdem es ihn durchschienen hat. Ist der Filter beispielsweise rot, so werden durch ihn hauptsächlich rote Wellenlängen hindurch gelassen. (vgl. WÄGER 2017: 70f; Heinze GmbH o.J.)

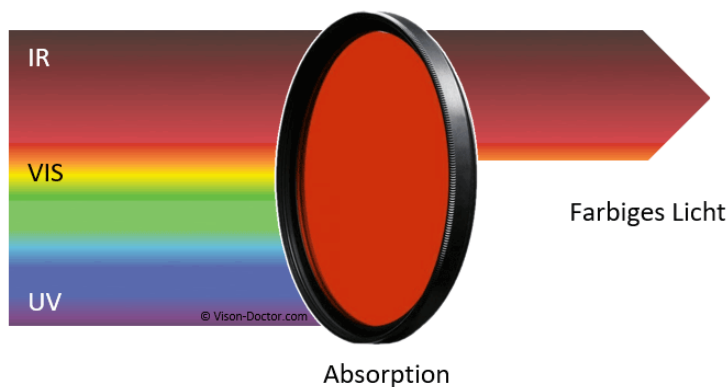


Abbildung 36: roter Farbfilter (Quelle: FERMUM o.J.).

Um Licht zu erzeugen, welches nur aus einer Wellenlänge besteht, benötigt man einen Laser. Diese in der Natur nicht vorkommende Lichtquelle kann Licht erzeugen, das wirklich nur aus einer Wellenlänge besteht. (vgl. WÄGER 2017: 70f)

Häufig wird bei der Beschreibung von Farben zwischen Lichtfarben und Körperfarben unterschieden. Im Grunde sind jedoch alle Farben Lichtfarben, da Farbe über Licht transportiert

wird. Trotzdem kann man Farben in Lichtfarben und Körperfarben unterteilen, indem man Lichtfarben als farbiges Licht bezeichnet, welches aktiv von einer Lichtquelle ausgesendet wird, und Körperfarben als Farben bezeichnet, welche erst durch das Anstrahlen eines Körpers mit weißem Licht entstehen. (vgl. WÄGER 2017: 70f)

3.2. Lichtfarben

Lichtfarben sind besonders im Bereich der Technik bei digitalen Monitoren und Projektoren von Bedeutung, da hier Farben, welche in der Natur vorkommen, reproduziert werden müssen. Hierfür sind drei Farben von entscheidender Bedeutung: Rot, Grün und Blau. Aus diesen drei Farben lassen sich beinahe alle Farben mischen und werden daher auch Primärfarben genannt. Diese Farben werden mit drei verschieden, farbigen Lichtquellen erzeugt. (vgl. WÄGER 2017: 72f)

Mischt man alle drei Primärfarben in voller Intensität zusammen, so erhält man Weiß. Daher spricht man hier auch von der sogenannten *additiven Farbmischung*. (vgl. Heinze GmbH o.J.) Um Schwarz zu erhalten, müssen alle drei Lichtquellen ausgeschaltet werden.

Dieses aus den drei Primärfarben Rot, Grün und Blau bestehende Farbsystem ist unter dem Namen *RGB* bekannt (*siehe 3.4.2. RGB*). (vgl. WÄGER 2017: 72f, 204)

Die Anzahl der durch Monitore wie in *Abbildung 38* darstellbaren Farben wird Farbtiefe genannt (*siehe 4.3.2. Farbtiefe*).

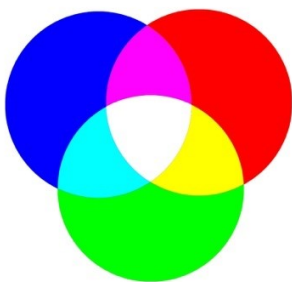


Abbildung 37: additive Farbmischung (Quelle: WECKNER o.J.).

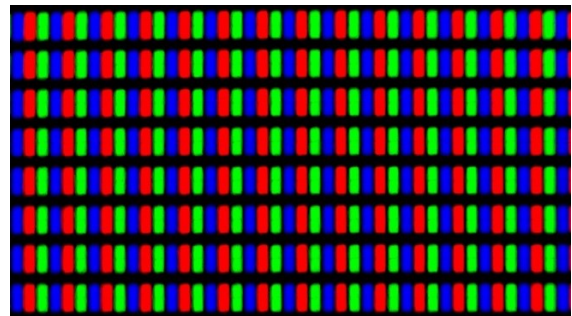


Abbildung 38: Einzelpixel eines Monitors bestehend aus den Farben Rot, Grün und Blau (Quelle: Imaging Arts Printing 2015).

3.3. Körperfarben

Körperfarben spielen mehr in der analogen Welt und in der Drucktechnik eine Rolle. Sie entstehen nicht durch einen Filterprozess beim Durchscheinen eines Körpers, wie dies bei Lichtfarben der Fall ist, sondern durch die Reflexion von bestimmten Wellenlängen eines Körpers. (vgl. WÄGER 2017: 75, 78; PCC Gruppe 2019)

Trifft weißes Licht auf einen Körper, so kann dieses auf verschiedene Art und Weise von diesem reflektiert und/oder absorbiert werden. Ein Spiegel zum Beispiel reflektiert alle Wellenlängen des eintreffenden weißen Lichts (vgl. Südwestrundfunk 2020). Daher kommt es annähernd in der gleichen Form wieder zurück und bleibt weiß, da sich die Wellenlängen durch additive Farbmischung wieder zu weißem Licht vereinen (vgl. Heinze GmbH o.J.). Werden jedoch bestimmte Wellenlängen von einem Körper absorbiert, so werden nur noch die restlichen Wellenlängen reflektiert. Durch den Umstand, dass nicht mehr alle Wellenlängen reflektiert werden, kann durch die additive Farbmischung auch kein Weiß mehr entstehen – der betreffende Körper erscheint uns in einer von den reflektierten Wellenlängen abhängigen Farbe. (vgl. WÄGER 2017: 70f) Mischt man Körperfarben miteinander, so verringert sich die Anzahl der reflektierten Wellenlängen – man spricht daher auch von der subtraktiven Farbmischung (vgl. Heinze GmbH o.J.).

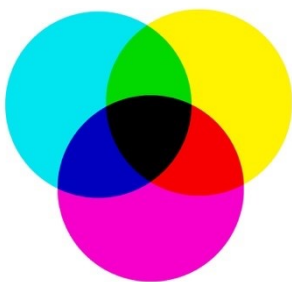


Abbildung 39: subtraktive Farbmischung (Quelle: WECKNER o.J.).

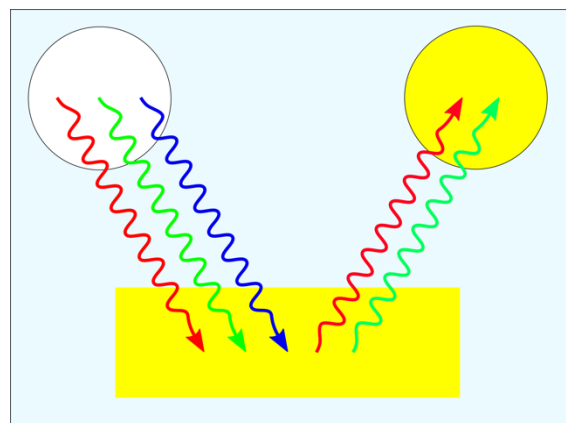


Abbildung 40: Entstehung von Körperfarben (Quelle: eigene Abbildung).

3.4. Farbsysteme

Um Farben zu vereinheitlichen und somit reproduzierbar zu machen, ist es notwendig, sie in Farbsystemen zu ordnen. Man unterscheidet dabei Farbkreise und die etwas komplexeren Farbmodelle. (vgl. WÄGER 2017: 180)

Farbkreise gehen auf die ersten Versuche zurück, Farben in Systeme einzuordnen (vgl. LIEBMANN und WELSCH 2018: 115). Sie basieren auf den sogenannten Grundfarben. Diese Farben sind keine exakte physikalische Größe, sondern eine willkürliche Unterteilung des sichtbaren Lichtes. (vgl. WÄGER 2017: 64) Heute werden Farbkreise meist nur mehr im historischen Kontext und in der Kunst verwendet. In technischen Anwendungen werden Farbmodelle eingesetzt. (vgl. LIEBMANN und WELSCH 2018: 115)

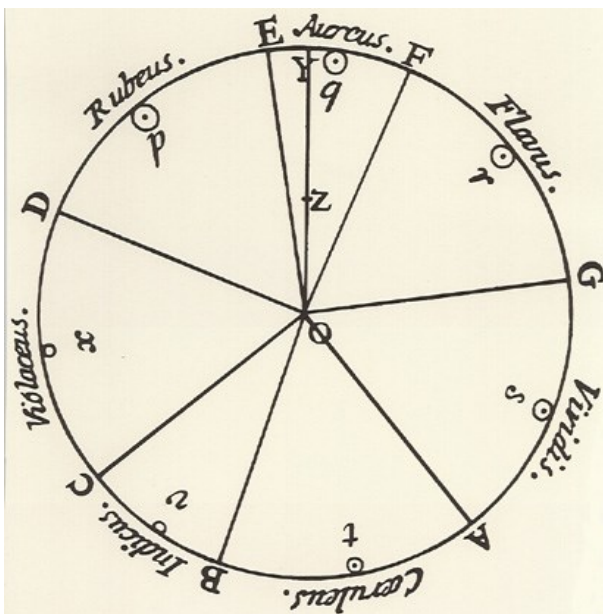


Abbildung 41: Farbkreis von Isaac Newton (Quelle: BAUMANN 2011).



Abbildung 42: Farbkreis von Johannes Itten (Quelle: KÖSTNER o.J.).

3.4.1. CIE-Normvalenzsystem

Das CIE-Normvalenzsystem ist ein Modell, welches 1931 von der internationalen Beleuchtungskommission CIE (*Commission internationale de l'éclairage*) zur Vereinheitlichung von Farben entwickelt wurde. Es ist das erste Farbsystem, welches Farben quantifizierbar macht, und somit die Grundlage für alle heute bekannten Farbmodelle wie RGB, HSV oder CMYK. Von nun an benötigt man für die Reproduktion von Farben keine Muster mehr und kann die Information über Farben auch mit Hilfe von Koordinaten in dem durch das CIE-Normvalenzsystem festgelegten, dreidimensionalen Farbraum kommunizieren. Dieses System beinhaltet

alle vom menschlichen Sehsinn wahrnehmbaren Farben und wurde durch Experimente mit sogenannten *Normbeobachtern* erstellt. Das Ergebnis ist wie bereits erwähnt ein dreidimensionaler Farbraum, welcher jedoch aus Übersichtsgründen meist nur in zweidimensionaler Darstellung gezeigt wird. Diese schuhsohlenförmige, zweidimensionale Darstellung ist auch als *CIE-Normfarbtafel* bekannt. (vgl. LIEBMANN und WELSCH 2018: 122)

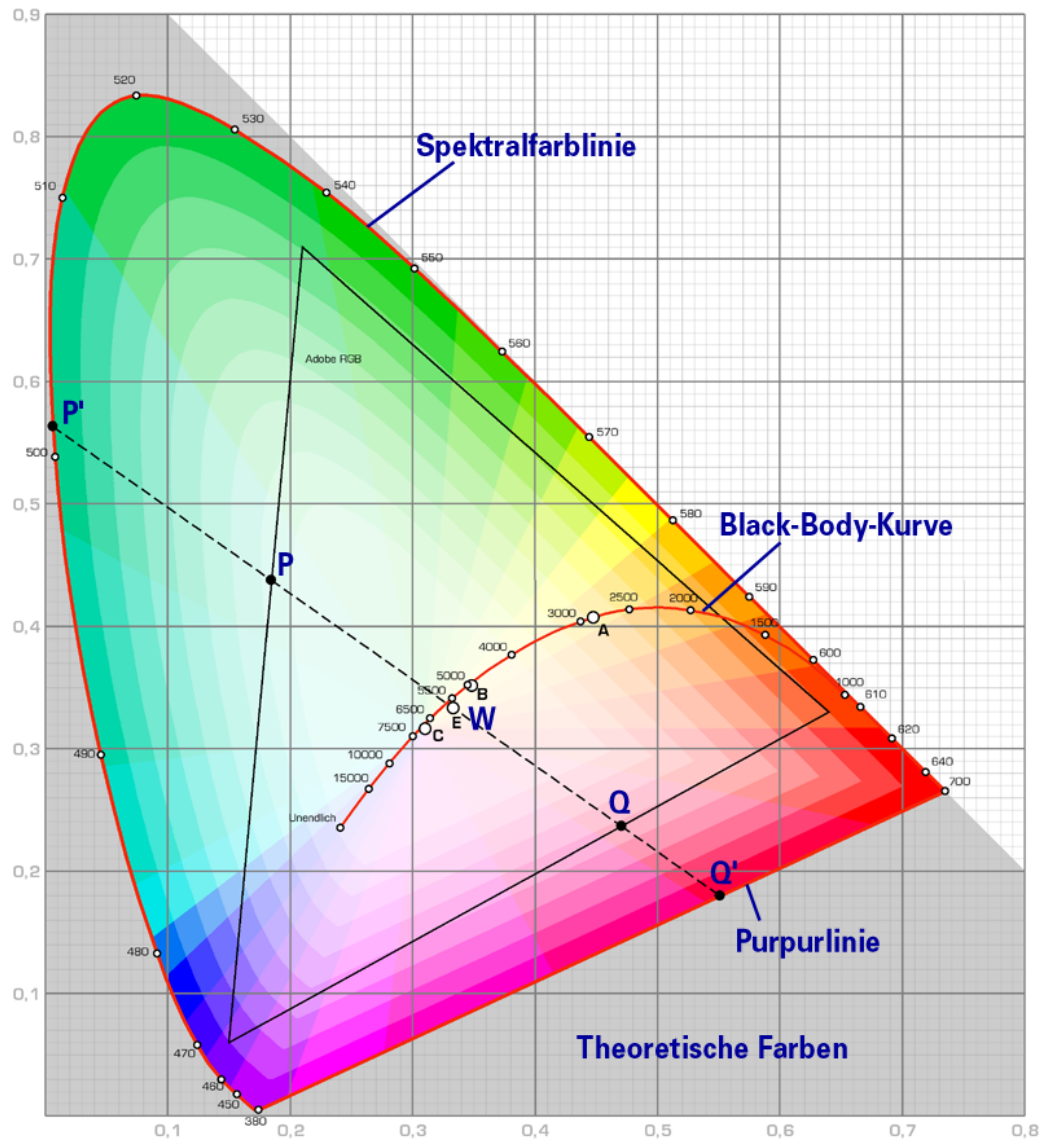


Abbildung 43: CIE-Normfarbtafel (Quelle: ANDERS 2005).

Die CIE-Normfarbtafel wird außen durch die *Spektralfarblinie* und durch die *Purpurlinie* begrenzt. Entlang der *Spektralfarblinie* befinden sich alle Spektralfarben. Die *Purpurlinie* verbindet die beiden Enden des Spektrums miteinander und schließt somit die Fläche. (vgl. LIEBMANN und WELSCH 2018: 122) In der Mitte der Farbtafel befindet sich der *Weißpunkt* W. Dies ist der Punkt, an dem alle drei Farben die gleiche Intensität aufweisen. Durch den *Weißpunkt* verläuft

die *Black-Body-Kurve*. Sie verläuft entlang der Farbtemperaturen eines idealen schwarzen Körpers. (vgl. X-Rite 2020) Als Farbtemperatur wird jene Farbe bezeichnet, welche ein idealer Schwarzer Körper bei einer bestimmten Temperatur aufweist (vgl. Max-Planck-Gesellschaft 2020).

Da die *CIE-Normfarbtafel* alle vom Menschen wahrnehmbaren Farben umfasst, eignet sie sich auch gut, um verschiedene Farbmodelle miteinander zu vergleichen. In *Abbildung 43* ist zum Beispiel ersichtlich, welchen Anteil an den vom Menschen wahrnehmbaren Farben Adobe RGB darstellen kann. (vgl. WÄGER 2017: 224f)

3.4.2. RGB

Das RGB-Farbmodell ist das wichtigste seiner Art in der digitalen Farbreproduktionstechnik. RGB steht für Rot, Grün und Blau. Diese Farben werden auch als Primärfarben bezeichnet, da sie als Grundlage für alle weiteren Farben dienen. Als Sekundärfarben werden in weiterer Folge jene Farben bezeichnet, welche sich aus der Mischung zweier Farben ergeben. Bei RGB sind dies die Primärfarben des *CMYK-Modells* *Cyan*, *Magenta* und *Gelb*. (vgl. WÄGER 2017: 204)

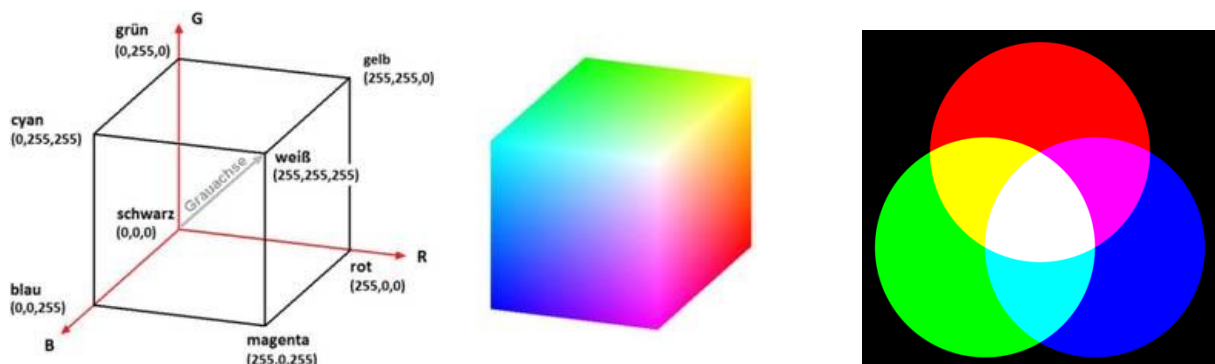


Abbildung 44: RGB-Farbwürfel und subtraktive Farbmischung (Quelle: GONZALEZ UND WOODS o.J.; QUARK67 2006).

Bei der Mischung aller drei Primärfarben in voller Intensität ergibt sich Weiß. Bei gleichen Mischverhältnissen ergibt sich immer ein Grauton, bis schließlich Schwarz beim Wegfall der Lichtquellen entsteht. Die Intensität der einzelnen Farben kann in Prozent von 0 bis 1 oder von 0 bis 255 angegeben werden. Am gebräuchlichsten ist die Angabe mit Werten zwischen 0 und 255. (vgl. LIEBMANN und WELSCH 2018: 129; WÄGER 2017: 204f)

3.4.3. CMYK

Das *CMYK-Farbmodell* kann als analoges Äquivalent zum *RGB-Farbmodell* betrachtet werden. *CMYK* besteht ebenfalls aus drei Primärfarben. Die Mischung zweier Primärfarben ergibt als Sekundärfarben die Primärfarben des *RGB-Modells*, so wie dies bei *RGB* mit *CMYK* der Fall ist. Wie bereits in 3.4.2. *RGB* angeführt, sind die drei Primärfarben von *CMYK* Cyan, Magenta und Yellow (Gelb). Mischt man diese drei Farben in voller Intensität zusammen, so erhält man in der Theorie Schwarz. In der Praxis entsteht jedoch bedingt durch mangelnde Reinheit der Farben und durch andere Faktoren nur ein schmutziges Grau. Um ein reines Schwarz zu erhalten und um manchen Farben mehr Tiefe zu verleihen, benötigt man zusätzlich zu den drei Primärfarben noch Schwarz, welches das *K* bei *CMYK* ausmacht und für *Keycolor* steht. (vgl. WÄGER 2017: 200ff)



Abbildung 45: CMYK Farben (Quelle: Soremba GmbH 2020).

3.4.4. HSV und HSL

Während *RGB* und *CMYK* notwendig sind, um Farben durch Mischung zu erzeugen, sind HSV und HSL eher auf der Seite der Gestalter angesiedelt. Es ist nämlich nicht sehr einfach möglich, zum Beispiel in *RGB* die Helligkeit einer Farbe zu ändern. Dazu müssen mehrere „Stellrädchen“ gedreht werden und es wird dadurch auch schwierig, diese eine Änderung auch genauso auf andere Farben anzuwenden. (vgl. WÄGER 2017: 208)

Tabelle 2 veranschaulicht, wie sich die *RGB-Werte* verändern, wenn man in *HSV* den Helligkeitswert *V* verändert. Man erkennt deutlich, dass es in *HSV* viel einfacher ist, Parameter wie Farbton, Sättigung oder Helligkeit zu verändern. In *RGB* müsste man für die Änderung einer dieser Farbeigenschaften alle drei Werte verändern. Somit ist *HSV* oder *HSL* für gestalterische Zwecke das Farbmodell der Wahl.

Tabelle 2: Auswirkung der Veränderung der Helligkeit auf die Werte von RGB und HSV (Quelle: eigene Tabelle).

Resedagrün						
R	G	B		H	S	V
108	124	89		87°	28%	49%
R	G	B		H	S	V
178	204	147		87°	28%	80%
R	G	B		H	S	V
67	77	55		87°	28%	30%
Verkehrsrot						
R	G	B		H	S	V
204	6	5		0°	98%	80%
R	G	B		H	S	V
128	3	3		0°	98%	50%
R	G	B		H	S	V
77	2	2		0°	98%	30%

Aber was bedeutet nun *HSV* und *HSL*? – *H* steht in beiden Modellen für *Hue* (dt.: Farbton), *S* steht für *Saturation* (dt.: Sättigung) und *V* beziehungsweise *L* stehen beide für die Helligkeit. Der Helligkeitswert hat hier mit *V* (*Value*) und *L* (*Lightness*) nicht nur eine unterschiedliche Bezeichnung, sondern auch eine unterschiedliche Bedeutung. Bei *HSV* hat eine Farbe die größte Sättigung bei 100% *S* und 100% *V*. Bei *HSL* hingegen liegt die größte Sättigung einer Farbe bei 100% *S* und 50% *L*. (vgl. HÄSSLER 2018)

Bei *HSL* ist es daher leichter, die Helligkeit einer Farbe zu erhöhen, da man bei der Veränderung dieser nicht automatisch auch die Sättigung mitverändert (vgl. WÄGER 2017: 214).

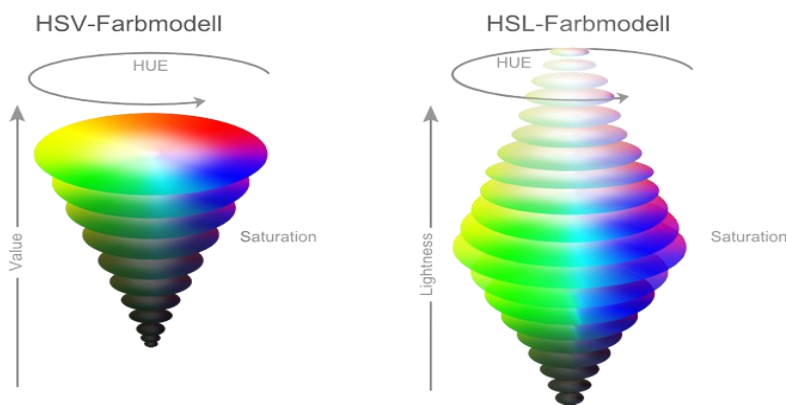


Abbildung 46: Unterschied zwischen HSV und HSL (Quelle: (vgl. HÄSSLER 2018).

4. Luftbildfotografie

Luftbilder können sowohl mit Flugzeugen, als auch mit Quadrokoptern, Hubschraubern und Satelliten aufgenommen werden. In dieser Arbeit wird nur auf die Aufnahme von Luftbildern mit Hilfe von Flugzeugen eingegangen.

Dabei werden zunächst Aufbau und Funktionsweise einer Digitalkamera erklärt. Danach wird mit dem Bildflug die Aufnahme von Luftbildern kurz beschrieben, bevor auf die Eigenschaften von Luftbildern eingegangen wird.

4.1. Aufbau und Funktionsweise einer Digitalkamera

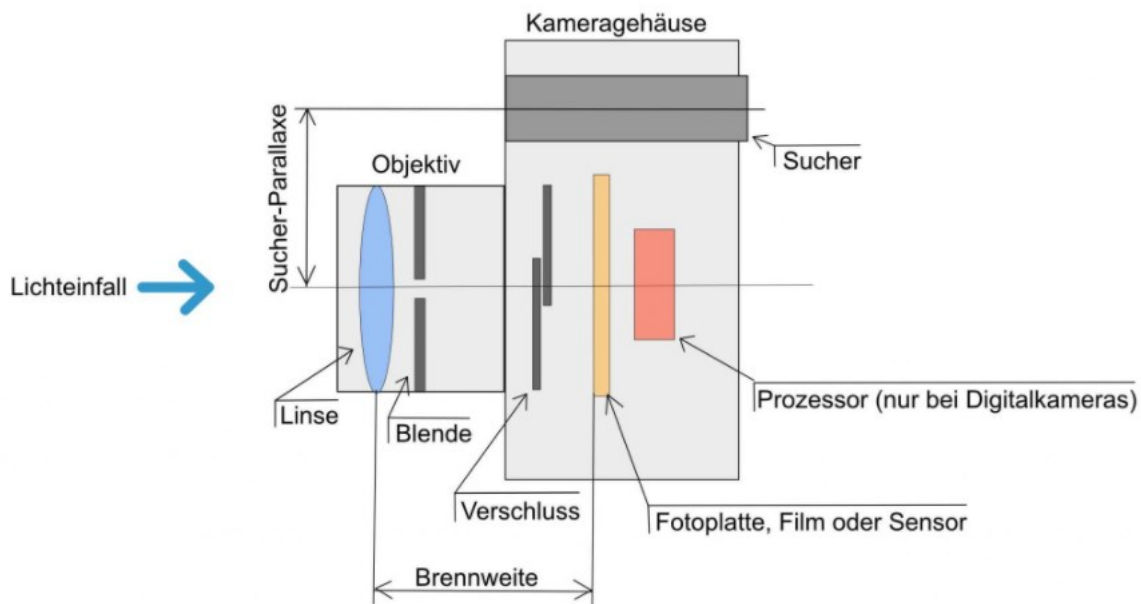


Abbildung 47: Aufbau einer (Digital-) Kamera (Quelle: Creative Dragon Works 2020).

Jede Kamera benötigt ein Objektiv, um die eingefangenen Lichtstrahlen korrekt auf den Sensor projizieren zu können. Im Objektiv sind mehrere Linsen hintereinander angeordnet und sind für die Regulierung der Schärfe und der Brennweite zuständig. Nachdem das Licht das Objektiv erfolgreich passiert hat, gelangt es in das Kameragehäuse, wo es schlussendlich auf den Bildsensor trifft. Dieser wandelt das Licht in digitale Signale um, welche vom Prozessor weiterverarbeitet und auf einem Speichermedium abgespeichert werden. (vgl. Creative Dragon Works 2020)

4.2. Bildflug

Als Bildflug wird jener Flug bezeichnet, bei dem Luftbilder aufgenommen werden (vgl. Spektrum der Wissenschaft Verlagsgesellschaft mbH 2001a).

Vor jedem Bildflug ist eine ausführliche Planung erforderlich. Neben der Berücksichtigung von Vorschriften, welche den Flug an sich betreffen, müssen auch Wetter und gewünschte Flugroute beachtet werden. (vgl. Spektrum der Wissenschaft Verlagsgesellschaft mbH 2001c)

Bei der Planung einer Flugroute spielt auch die Bildüberdeckung eine wichtige Rolle. Als Bildüberdeckung wird die Überlappung benachbarter Luftbilder bezeichnet. Diese wird in Prozent zur gesamten Bildformatseite angegeben. Man unterscheidet dabei in Längsüberdeckung (in Flugrichtung) und Querüberdeckung (quer zur Flugrichtung). (vgl. Spektrum der Wissenschaft Verlagsgesellschaft mbH 2001d) Die Bildüberdeckung dient primär dazu, um die Luftbilder über Passpunkte korrekt im Raum positionieren zu können. Bei höherer Bildüberdeckung ist auch eine dreidimensionale Bildauswertung möglich. (vgl. GeoFly GmbH o.J.)

Für einen Bildflug kommen meist speziell ausgerüstete Flugzeuge zum Einsatz. Diese müssen vor allem über spezielle Öffnungen für Kamerasysteme am Boden des Rumpfes verfügen. Zusätzlich sind - über die Notwendigkeit des normalen Flugbetriebs hinausgehend - Geräte zur exakten Positionsbestimmung erforderlich. (vgl. Spektrum der Wissenschaft Verlagsgesellschaft mbH 2001b)



Abbildung 48: Luftbildflugzeug Piper PA 31 Chieftain der Firma GeoFly (Quelle: CT Aeropics o.J.).

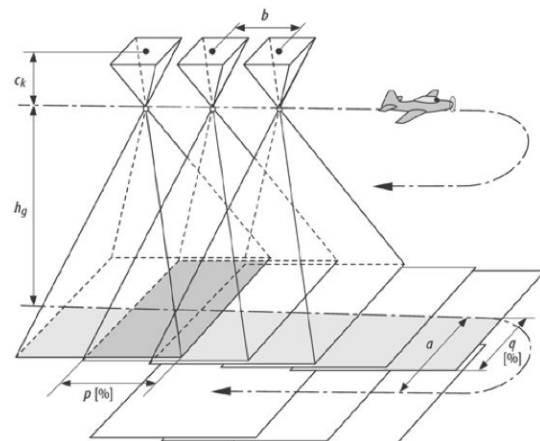


Abbildung 49: Bildüberdeckung (Quelle: Spektrum der Wissenschaft Verlagsgesellschaft mbH 2001d).

4.3. Eigenschaften von Luftbildern

4.3.1. Auflösung

„Als Auflösung bezeichnet man im technischen Verständnis die Angabe der horizontalen und vertikalen Punkte in einem Punktraster“ (Fuchs Media Solutions o. J.). Umgelegt auf die digitale Fotografie wären diese Punkte die Pixel in einem Bild. In der Fernerkundung wird der Begriff Auflösung jedoch ein wenig anders verwendet:

- **Räumliche Auflösung:**

Als räumliche Auflösung wird die Größe des Gebiets, welches mit einem Pixel aufgenommen wird, bezeichnet. Diese ist in Metern angegeben. (vgl. HODAM H. o.J.a)

- **Spektrale Auflösung:**

Ein Fernerkundungssensor nimmt meistens nicht nur sichtbares Licht, sondern auch für Menschen unsichtbare Infrarotstrahlung auf. Dazu ist der Sensor in sogenannte Kanäle aufgegliedert, von denen jeder für einen bestimmten Wellenlängenbereich des elektromagnetischen Spektrums zuständig ist. Das sichtbare Licht beispielsweise wird über die drei Kanäle Rot, Grün und Blau aufgenommen. Die Anzahl der Kanäle, welche von einem Fernerkundungssystem aufgenommen werden können, wird als spektrale Auflösung bezeichnet. (vgl. HODAM H. o.J.b)

- **Radiometrische Auflösung:**

Die radiometrische Auflösung gibt an, wie viele verschiedene Werte pro Kanal aufgenommen werden können. Grundsätzlich handelt es sich dabei um Grauwerte, welche erst in Kombination mit anderen Kanälen zu Farben werden können. So werden die Kanäle Rot, Grün und Blau durch additive Farbmischung zu Farben zusammengemischt (*siehe 3.4.2. RGB*). (vgl. HODAM H. o.J.c)

- **Zeitliche Auflösung:**

Als Zeitliche Auflösung bezeichnet man den Abstand zwischen zwei Luftbildaufnahmen. Je größer die Zeitliche Auflösung, desto geringer ist der zeitliche Abstand von zwei Aufnahmen. (vgl. HODAM H. o.J.d)

4.3.2. Farbtiefe

Die Farbtiefe gibt bei digitalen Bildern an, wie viele Farben pro Kanal gespeichert werden können. Schwarz-Weiß-Bilder haben nur einen Kanal, welcher aus Graustufen besteht. Farbbilder hingegen bestehen aus drei Kanälen – Rot, Grün und Blau (*siehe 3.4.2. RGB*).

Die Farbtiefe wird in Bit angegeben. Bit ist in der Computertechnik die kleinste mögliche Speichereinheit und kann die Werte 0 und 1 annehmen. Auf Bilder umgelegt wären dies Schwarz und Weiß, womit sich noch kein sinnvolles Bild (außer für Spezialanwendungen) darstellen ließe. Will man die Anzahl der speicherbaren Informationen erhöhen, so muss man mehrere Bit aneinanderreihen. 8 Bit werden dann zu einem Byte zusammengefasst und es lassen sich damit bereits 256 verschiedene Werte abspeichern.

Digitale Bilder werden üblicherweise mit 8 Bit pro Farbkanal abgespeichert, was insgesamt 24 Bit entspricht. Für spezielle Anwendungen werden aber auch öfters Bilder mit 16 Bit pro Farbkanal abgespeichert. (vgl. GADE 2019)



Abbildung 50: Vergleich verschiedener Farbtiefen (Quelle: MAIER 2015).

4.3.3. Dateiformate

4.3.3.1. TIFF

Das TIFF-Format (Tagged Image File Format) ist ein Bildformat für die Speicherung von hochwertigen Bilddaten. Neben den eigentlichen Pixeln (unkomprimiert oder verlustfrei komprimiert) können auch Ebenen oder Vektorobjekte von Bildbearbeitungsprogrammen in einer TIFF-Datei abgespeichert werden. Die Farbtiefe kann dabei bis 32 Bit pro Farbkanal reichen. (vgl. Mind Kommunikation GmbH o.J.)

4.3.3.2. GIF

Das GIF-Format wurde ursprünglich für Webanwendungen entwickelt und ist auch eines der ältesten Bildformate. Der große Nachteil dieses Formates ist, dass es insgesamt nur 256 Farben darstellen kann. Es sind daher auf GIF-Bildern deutliche Artefakte zu erkennen.

Trotz dieses erheblichen Nachteils wird dieses Format noch heute verwendet, da es mehrere Bilder hintereinander als Animation abspielen kann. (vgl. Mind Kommunikation GmbH o.J.)

4.3.3.3. JPEG

JPEG steht für „*Joint Photographic Experts Group*“ und ist der Name eines Gremiums, welches den JPEG-Standard entwickelt hat. Es handelt sich dabei um ein komprimiertes Format, das den Vorteil mit sich bringt, dass die Dateigröße teilweise erheblich verringert werden kann. Der Nachteil dieser Komprimierung ist natürlich, dass sich die Qualität verschlechtert.

Bei der JPEG-Komprimierung werden benachbarte Pixel mit ähnlichen Farbwerten zusammengefasst. Je größer die Komprimierung ist, desto weniger müssen sich diese benachbarten Pixel ähneln, um zusammengefasst zu werden. (vgl. Mind Kommunikation GmbH o.J.)

4.3.3.4. PNG

PNG – Portable Network Graphics – wurde als Ersatz für das GIF-Format entwickelt, welches bis 2006 mit Patentforderungen belegt war. Bis auf die fehlende Fähigkeit, mehrere Bilder als Animation abzuspeichern, bietet PNG nur Vorteile gegenüber GIF. Selbst JPEG gegenüber ist PNG überlegen, da es Bilder verlustfrei komprimiert. Einzig die Dateigröße ist bei PNG aufgrund der verlustfreien Kompression etwas höher, als bei JPEG.

Ein großer Vorteil von PNG ist, dass es Transparenzen abspeichern kann. (vgl. Mind Kommunikation GmbH o.J.)

4.3.4. Orientierung

- **Innere Orientierung:**

Die Innere Orientierung beschreibt die Lage eines Punktes auf der Bildebene zum Projektionszentrum im Objektiv. Ihre Parameter sind die Kamerakonstante (Abstand zum Projektionszentrum) und die Bildkoordinaten im Bezug zum Bildmittelpunkt. (vgl. Spektrum der Wissenschaft Verlagsgesellschaft mbH 2001e)

- **Äußere Orientierung:**

Bei der äußeren Orientierung handelt es sich vereinfacht dargestellt um die Lage der Messkamera zum aufzunehmenden Objekt, daher um die Lage eines Objektpunktes zum Projektionszentrum im Objektiv. (vgl. Spektrum der Wissenschaft Verlagsgesellschaft mbH 2001f)

4.3.5. Orthofoto

Ein Orthofoto ist ein entzerrtes Luftbild. Während ein normales Luftbild (wie das menschliche Sehen und Bilder jeder anderen Kamera) eine Zentralprojektion aufweist, ist ein Orthofoto in der Parallelprojektion gehalten. Um ein Luftbild entzerren zu können, ist ein Geländemodell erforderlich, da auf einem normalen Luftbild höher gelegene Bereiche durch die Zentralprojektion größer erscheinen als tiefer gelegene. (vgl. Spektrum der Wissenschaft Verlagsgesellschaft mbH 2001g; Technisches Zeichnen 2008)

Trotz Entzerrung mittels Geländemodell können gewisse „Restverzerrungen“ nicht vollständig eliminiert werden, was sich durch den sogenannten Bildsturz bemerkbar macht. Es wird zwar versucht, den Bildsturz möglichst zu minimieren, was jedoch nur in *True Orthofotos* vollständig möglich ist. (vgl. Spektrum der Wissenschaft Verlagsgesellschaft mbH 2001g; GeoFly GmbH o.J.)

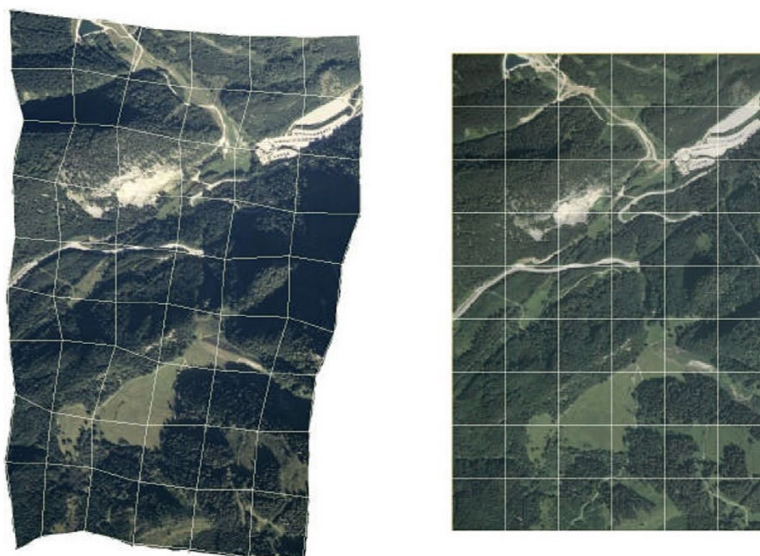


Abbildung 51: Unterschied zwischen Luftbild (re.) und Orthofoto (li.) (Quelle: ÖTK Österreichischer Touristenklub 2018).

4.3.6. Bildsturz

Als letzte, eher unerwünschte Eigenschaft eines Luftbildes sei noch der bereits erwähnte Bildsturz erläutert. Dieser ergibt sich dadurch, dass bei der Entzerrung meist nur das Gelände berücksichtigt wird. Objekte, welche nicht direkt am Boden liegen, werden weiterhin verzerrt dargestellt. Diese Verzerrung macht sich durch das Kippen von Gebäudefassaden, Bäumen, Masten und anderen Objekten bemerkbar. (vgl. Land Tirol o.J.)

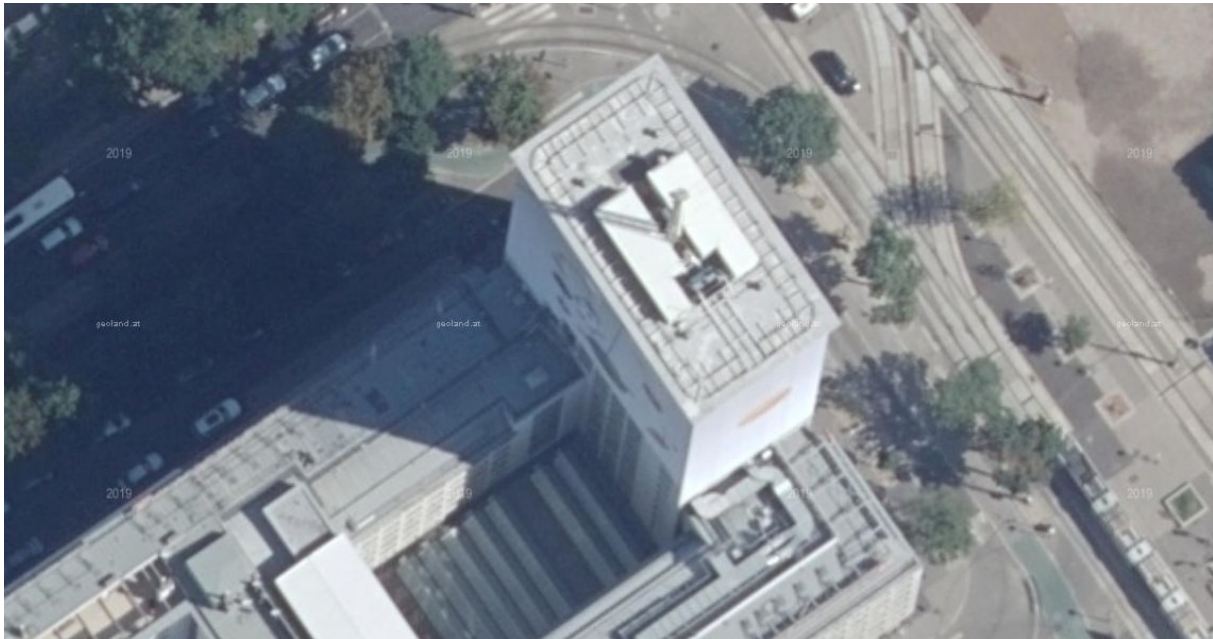


Abbildung 52: Bildsturz beim Ringturm in Wien (Quelle: Stadt Wien und Österreichische Länder 2016).

5. Grundlagen der Statistik

In diesem Kapitel werden jene Grundlagen der Statistik vermittelt, welche die Durchführung des praktischen Teils dieser Arbeit erleichtern beziehungsweise erst ermöglichen. Um Vollständigkeit zu gewährleisten, werden - wenn nötig - auch Inhalte beschrieben, welche im Praktischen Teil keine Verwendung finden.

5.1. Allgemeines

Statistik begegnet einem im Alltag viel öfter, als man denkt. Viele Zeitungsartikel beinhalten interessante und auch unterhaltsame Statistiken, welche die Welt etwas greifbarer machen. So kann man durch sie zum Beispiel einen Blick auf die Zusammensetzung, Denk- und Verhaltensweisen der Gesellschaft werfen. (vgl. Statista GmbH o.J.)

Natürlich ist die Statistik auch in der Kartographie und in der Geoinformation allgegenwärtig. Dabei können Kartographie und Geoinformation aus der Perspektive der Statistik auch als deren Werkzeug angesehen werden. Entscheidend ist die Frage, welches Ziel unter der Verwendung der Statistik erreicht werden soll. Hat eine Karte die Funktion, eine Statistik zu visualisieren, wobei kartographische Gesichtspunkte nicht im Detail berücksichtigt werden, so ist sie eher ein Werkzeug der Statistik. Werden statistische Methoden und Kennzahlen aber verwendet, um eine Karte zu erstellen oder um Geodaten zu verarbeiten, wobei der Fokus auf der Karte beziehungsweise auf den Geodaten liegt, so kann man auch die Statistik als Werkzeug der Kartographie und Geoinformation betrachten.

So ist die Statistik auch in dieser Arbeit ein Werkzeug, um Geodaten zu verarbeiten. Sie hilft dabei, aus einer Vielzahl von Farbwerten jenen auszuwählen, welcher den Anforderungen an das Endergebnis am besten entspricht.

5.2. Arten der Statistik

5.2.1. Deskriptive Statistik

Die deskriptive Statistik (beschreibende Statistik) dient dazu, Aussagen über vorhandene Daten zu treffen (vgl. Mentorium GmbH 2021). Eine Aussage über Daten zu treffen bedeutet, diese auszuwerten und damit greifbarer und überschaubarer zu machen (vgl. SCHÄFER 2016: 47-81).

Die Daten, welche der deskriptiven Statistik zu Grunde liegen, gehen entweder aus einer Stichprobe oder aus einer Vollerhebung hervor. Bei einer Vollerhebung werden alle Daten zu

einem bestimmten Sachverhalt erhoben, bei einer Stichprobe nur ein Teil davon. Es sind vor allem wirtschaftliche Gründe dafür verantwortlich, warum in manchen Fällen statt einer Vollerhebung nur eine Stichprobe entnommen wird. (vgl. KÖNIG 2010)

Bei der Auswertung dieser Daten können Kennzahlen berechnet werden, welche dann eben diese Aussagen über die Daten treffen können. Diese Kennzahlen sind Mittelwert, Modus und Median und werden später noch erläutert.

Auch mit Tabellen und Diagrammen können Aussagen über Daten getroffen werden. (vgl. Novustat 2021)

5.2.2. Inferenzstatistik

Bei der Inferenzstatistik (schließende Statistik) wird versucht, auf der Grundlage einer vorliegenden Stichprobe eine Aussage über die Grundgesamtheit zu treffen. Sie wird dann verwendet, wenn statt einer Vollerhebung nur eine Stichprobe entnommen wird. Die Grundgesamtheit stellt dabei alle Daten zu einem gewissen Sachverhalt dar, während die Stichprobe nur ein Teil davon ist. (vgl. SCHÄFER 2016: 109-119)

Um herauszufinden, ob diese Aussage tatsächlich auf die Grundgesamtheit zutrifft und nicht nur auf die Stichprobe, werden sogenannte „Hypothesentests“ verwendet. Mit diesen Tests kann überprüft werden, zu welcher Wahrscheinlichkeit eine Aussage (Hypothese) über eine Stichprobe auch auf die Grundgesamtheit zutrifft. (vgl. JESUSSEK 2020)

Bei der automatischen Ableitung von Farbwerten aus Texturen kommt die deskriptive Statistik zum Einsatz, da zur Berechnung jede einzelne Textur herangezogen wird und nicht von wenigen berechneten Farbwerten auf andere Farbwerte geschlossen wird. Es handelt sich daher um eine Vollerhebung.

5.3. Statistische Kennzahlen

5.3.1. Arithmetischer Mittelwert

Der arithmetische Mittelwert wird auch als Durchschnitt oder einfach nur Mittelwert bezeichnet. Er wird aus der Summe aller betrachteten Werte, dividiert durch deren Anzahl, berechnet und ist die am häufigsten verwendete statistische Kennzahl. Der Nachteil des Mittelwertes liegt darin, dass er nicht robust gegenüber Ausreißern ist. Wenn zum Beispiel wenige Farbwerte einer

Gebäudefläche besonders dunkel (im Vergleich zu den anderen Farbwerten) sind, so beeinflussen diese den Mittelwert signifikant und das Ergebnis wäre insgesamt dunkler als erwartet. (vgl. STOCKER UND STEINKE 2017: 52-55)

5.3.2. Modus (Modalwert)

Der Modalwert ist der am häufigsten vorkommende Wert innerhalb einer betrachteten Datenmenge. Zu beachten ist, dass es auch mehrere häufigste Werte geben kann! (vgl. STOCKER UND STEINKE 2017: 59)

5.3.3. Median (Zentralwert)

Beim Median werden die zu betrachtenden Werte zunächst der Größe nach geordnet. Von diesen Werten ist dann der in der Mitte liegende Wert der Median. Bei den Zahlen von 1 bis 5 wäre zum Beispiel 3 der Median. Im Gegensatz zum arithmetischen Mittelwert ist der Median robust gegenüber Ausreißern. So wird das Medianeinkommen eines Landes geringer ausfallen, als das Durchschnittseinkommen. (vgl. STOCKER UND STEINKE 2017: 56)

5.3.4. Minimum und Maximum

Das Minimum ist der kleinste Wert und das Maximum der größte Wert einer betrachteten Menge von Werten (vgl. STOCKER UND STEINKE 2017: 69).

5.3.5. Quantile

Das Quantil gibt an, wie viel Prozent der Werte einer vordefinierten Menge einen gewissen Wert über- beziehungsweise unterschreiten. Um Quantile zu erhalten, muss der Datensatz (wie beim Median) nach der Größe der Werte geordnet werden. Der Median ist dabei als 50%-Quantil ein Spezialfall. Weitere Spezialfälle sind die 25%- und 75%-Quantile, welche auch als Quartile bezeichnet werden. (vgl. STOCKER UND STEINKE 2017: 62)

5.3.6. Spannweite

Die Spannweite ist die Differenz zwischen Maximum und Minimum. Will man besonders extreme Ausreißer vernachlässigen, so kann man alternativ den Abstand zwischen zwei Quantilen betrachten. (vgl. STOCKER UND STEINKE 2017: 69)

5.4. Clusteranalyse

Das grundlegende Ziel der Clusteranalyse ist, dass Elemente einer gegebenen Menge verschiedenen Gruppen (Cluster) zugeordnet werden. Diese Zuordnung erfolgt aufgrund deren Ähnlichkeit zueinander. Es soll also innerhalb der Cluster eine möglichst große Ähnlichkeit zwischen den Elementen und zwischen den Clustern selbst eine möglichst geringe Ähnlichkeit bestehen. (vgl. OERTHEL UND TUSCHL 1995: 198)

Da im Verlauf dieser Arbeit die Clusteranalyse im praktischen Teil zur Anwendung kommen wird, seien hier zwei wichtige Verfahren der Clusteranalyse kurz beschrieben:

5.4.1. Partitionierende Verfahren

Partitionierende Verfahren gehen bei der Clusterberechnung im Vorhinein von einer fixen Anzahl an Clustern aus, zu welchen die Elemente per Zufall zugeordnet werden. Durch laufende Umgruppierung der Elemente zwischen den Clustern wird im nächsten Schritt versucht, diesen Grundzustand nach und nach zu verbessern. (vgl. LÜKEN UND SCHIMMELPFENNIG 2017)

Das bekannteste partitionierende Verfahren ist der K-Means-Algorithmus. Bei diesem Algorithmus werden - wie bereits erwähnt - zu Beginn per Zufall Cluster gebildet. Pro Cluster wird der Mittelwert aller Elemente bestimmt und als Clustermittelpunkt bezeichnet. Danach wird das Ergebnis evaluiert, indem alle Elemente der verschiedenen Cluster jenem Clustermittelpunkt zugeordnet werden, welchem sie am nächsten liegen. So können Elemente auch von einem Cluster in einen anderen verschoben werden. Aufgrund der Verschiebung der Elemente müssen nun die Clustermittelpunkte neu berechnet werden – vorausgesetzt, es fand eine Verschiebung von Elementen zu anderen Clustern statt. Diese Vorgänge werden so lange wiederholt, bis keine Elemente mehr verschoben werden müssen, weil sie bereits alle dem Clustermittelpunkt zugeordnet sind, welchem sie am nächsten liegen. (vgl. LÜKEN UND SCHIMMELPFENNIG 2017)

5.4.2. Hierarchische Verfahren

Bei den hierarchischen Verfahren gibt es zwei grundsätzliche Möglichkeiten, nach denen vorgegangen werden kann: die agglomerativen Clusterverfahren und die diversiven Clusterverfahren (vgl. OERTHEL und TUSCHL 1995: 198).

Bei den agglomerativen Verfahren werden nacheinander zwei Elemente oder Cluster so lange zu einem neuen Cluster zusammengefasst, bis nur noch ein Cluster übrigbleibt. Damit diese Methode Sinn macht, muss die Berechnung ab einer gewissen Zahl von Clustern gestoppt werden. (vgl. OERTHEL und TUSCHL 1995: 198)

Bei den diversiven Verfahren läuft der vorhin beschriebene Vorgang genau umgekehrt ab. Man startet mit der Clusteranzahl eins und teilt den Cluster in immer mehr einzelne Cluster auf. Auch hier muss der Vorgang sinnvoller Weise vor dem Ende bei einer gewissen Clusterzahl gestoppt werden (jedes einzelne Element bildet einen eigenen Cluster). (vgl. OERTHEL und TUSCHL 1995: 198)

Neben der gewünschten Anzahl der Cluster kann auch das sogenannte Distanzmaß bei beiden Verfahren (agglomerativ und diversiv) bestimmen, wie weit die Cluster zusammengefasst beziehungsweise geteilt werden. In diesem Fall ist die Anzahl der Cluster im Vorhinein unbekannt und ergibt sich erst während der Berechnung, je nach Distanzmaß. Die hierarchische Clusterberechnung ist auch für die Farbwertextraktion aus Gebäudetexturen interessant, da über das Distanzmaß die maximale Ähnlichkeit der Farbwerte pro Cluster bestimmt werden kann. (vgl. OERTHEL und TUSCHL 1995: 199)

6. Das neue Erscheinungsbild für 3D-Stadtmodelle

In diesem Kapitel soll das in dieser Masterarbeit neu zu entwerfende Erscheinungsbild vorgestellt werden. Zu Beginn werden bestehende 3D-Stadtmodelle hinsichtlich ihres Erscheinungsbildes beschrieben, um daraus später das neue Erscheinungsbild ableiten zu können.

6.1. Beschreibung des Erscheinungsbildes bestehender 3D-Stadtmodelle

Da im praktischen Teil nur Gebäude und die Bodenbedeckung visualisiert werden, beschränkt sich diese Beschreibung auch auf letztgenannte Inhalte. Aufgrund der großen Zahl an weltweit verfügbaren 3D-Stadtmodellen und aufgrund des begrenzten Umfangs dieser Masterarbeit kann nicht garantiert werden, dass alle möglichen bestehenden Varianten von Erscheinungsbildern betrachtet werden.

6.1.1. Wien

Erscheinungsbild des 3D-Stadtmodells von Wien:

- LOD2.1-Gebäude mit roten Dächern und grauen Fassaden mit Grundkarte (hier: digitaler Stadtplan Wien) als Bodenbedeckung. (vgl. Stadt Wien o.J.a)
- Texturierte LOD2.1-Gebäude und Orthofoto als Bodenbedeckung. (vgl. Stadt Wien o.J.f)
- Detailliertes LOD3.4-Architekturmodell ausgewählter Bereiche Wiens. (vgl. Stadt Wien o.J.g)



Abbildung 53: LOD2.1-Gebäude mit roten Dächern und grauen Fassaden (Quelle: Stadt Wien o.J.f).



Abbildung 54: texturierte LOD2.1-Gebäude mit Orthofoto als Bodenbedeckung (Quelle: Stadt Wien intern).

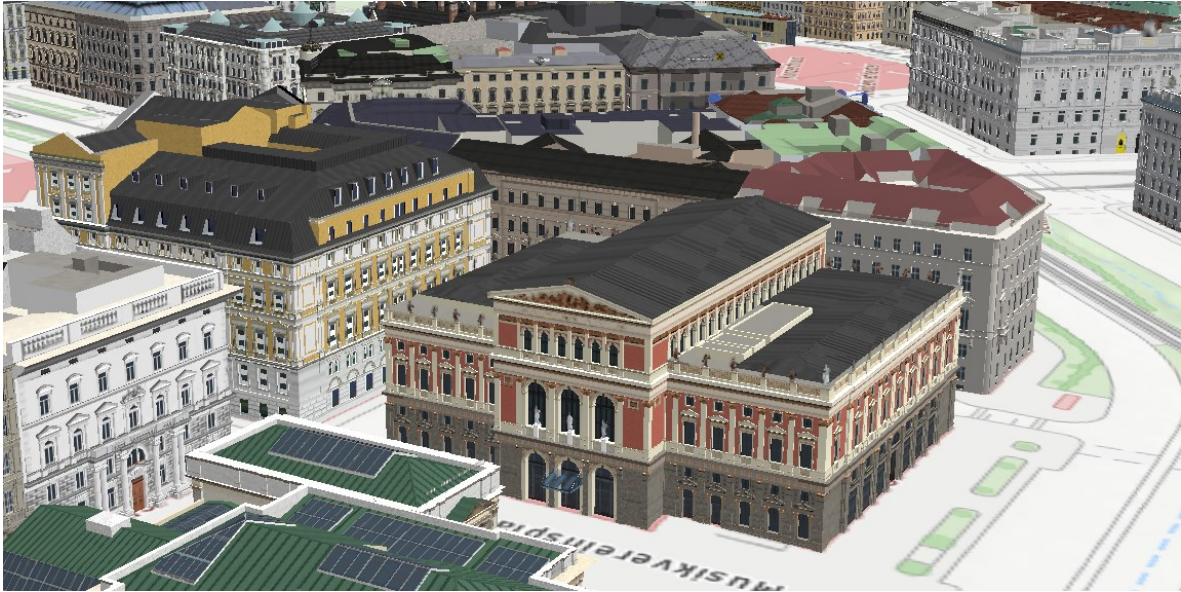


Abbildung 55: detailliertes LOD3-Architekturmodell (Quelle: Stadt Wien o.J.g).

6.1.2. Berlin

Die Stadt Berlin besitzt, im Vergleich zum 3D-Stadtmodell von Wien, zusätzlich ein texturiertes Oberflächenmodell, auch Mesh genannt. (vgl. Virtual City Systems o.J.c)

Erscheinungsbild des 3D-Stadtmodells von Berlin:

- Weiße LOD2-Gebäude und Grundkarte (hier: OpenStreetMap) als Bodenbedeckung.
- Texturierte LOD2-Gebäude und Orthofoto als Bodenbedeckung.
- 3D-Mesh: texturiertes Oberflächenmodell.



Abbildung 56: LOD2-3D-Stadtmodell von Berlin mit texturierten Gebäuden, Bäumen und Orthofoto (Quelle: Virtual City Systems o.J.c).



Abbildung 57: LOD2-3D-Stadtmodell von Berlin mit weißen Gebäuden, Bäumen und OpenStreetMap (Quelle: Virtual City Systems o.J.c).



Abbildung 58: 3D-Stadtmodell von Berlin als 3D-Mesh (Quelle: Virtual City Systems o.J.c).

6.1.3. Bremen

Das 3D-Stadtmodell von Bremen ist ähnlich ausgeprägt wie das von Berlin. Im Gegensatz zu Berlin wurde hier anscheinend das texturierte Gebäudemodell zu Gunsten des texturierten Oberflächenmodells weggelassen. (vgl. Virtual City Systems o.J.d)

Erscheinungsbild des 3D-Stadtmodells von Bremen:

- Graue LOD2-Gebäude und drei verschiedene Grundkarten als Bodenbedeckung.
- 3D-Mesh (texturiertes Oberflächenmodell).

(vgl. Virtual City Systems o.J.d)



Abbildung 59: Texturiertes Oberflächenmodell der Stadt Bremen (Quelle: Virtual City Systems o.J.d).

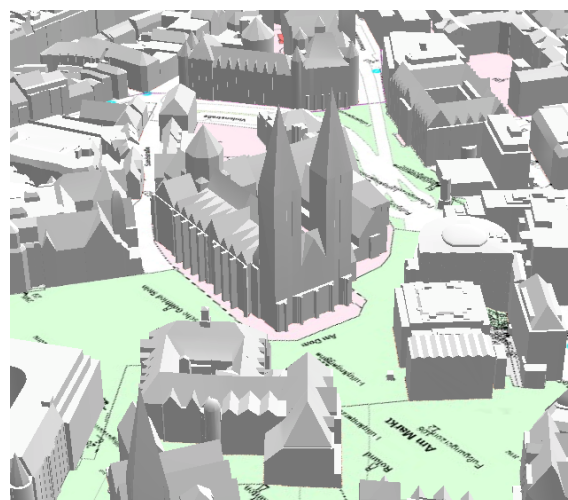


Abbildung 60: Graue LOD2-Gebäude in Bremen mit Grundkarte als Bodenbedeckung (Quelle: Virtual City Systems o.J.d).

6.1.4. Posen

Auch Posen verfügt über ein ähnliches Erscheinungsbild der Gebäude und der Bodenbedeckung.

Ähnlich wie in Wien existieren LOD2-Gebäude mit roten Dächern und texturierte LOD2-Gebäude. Zusätzlich gibt es hier ein Design, welches Dächer in realitätsnahen Farben darstellt. Des Weiteren existiert noch - wie auch in Berlin und Bremen - ein texturiertes Oberflächenmodell (Mesh). (vgl. Stadt Posen o.J.)

Erscheinungsbild des 3D-Stadtmodells von Posen:

- LOD2-Gebäude mit roten Dächern und grauen Fassaden mit Grundkarte als Bodenbedeckung.
- LOD2-Gebäude mit grauen Fassaden und realitätsnahe eingefärbten Dachflächen mit Grundkarte als Bodenbedeckung.
- Texturierte LOD2-Gebäude und Orthofoto als Bodenbedeckung.
- 3D-Mesh (texturiertes Oberflächenmodell).

(vgl. Stadt Posen o.J.)

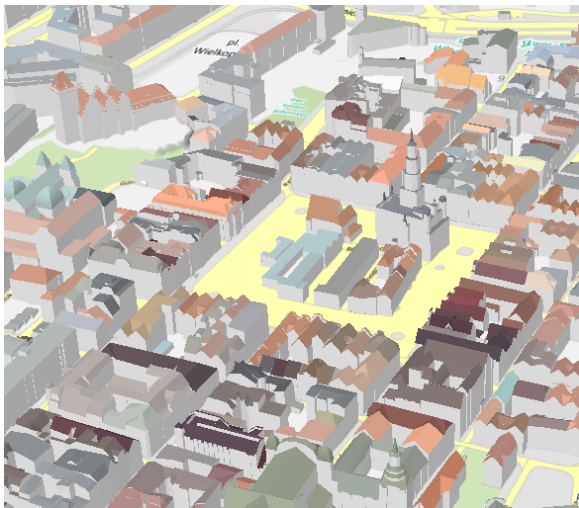


Abbildung 61: 3D-Stadtmodell Posen mit realitätsnahe eingefärbten Dächern und grauen Fassaden, Grundkarte als Bodenbedeckung (Quelle: Stadt Posen o.J.).

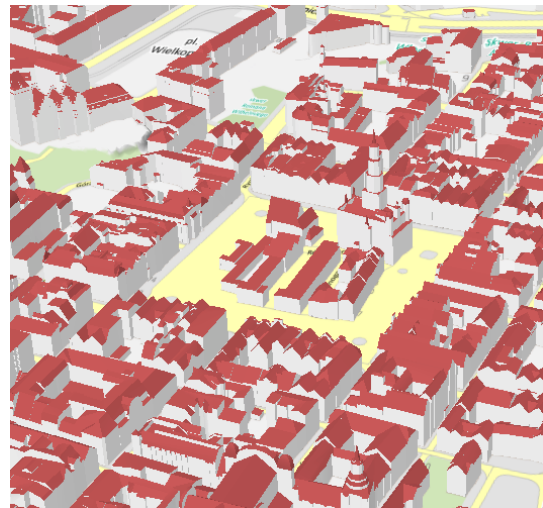


Abbildung 62: 3D-Stadtmodell Posen mit roten Dächern und grauen Fassaden, Grundkarte als Bodenbedeckung (Quelle: Stadt Posen o.J.).



Abbildung 63: 3D-Stadtmodell Posen mit texturiertem Oberflächenmodell (3D-Mesh) (Quelle: Stadt Posen o.J.).



Abbildung 64: 3D-Stadtmodell Posen mit texturierten LOD2-Gebäuden, Orthofoto als Bodenbedeckung (Quelle: Stadt Posen o.J.).

6.1.5. Schweiz

Die Schweiz nimmt bei den hier analysierten Stadtmodellen eine Sonderstellung ein. Anstatt einer Stadt existiert die gesamte Schweiz in 3D. Auf Grundlage der topographischen Karte der Schweiz sind hier Gebäude, Bäume und Brücken als dreidimensionale Elemente enthalten. Vom Design her ähneln die hier zu vergleichenden Inhalte den 3D-Stadtmodellen von Wien und Posen, da auch hier alle Dächer in Rot gehalten sind und eine Grundkarte die Bodenbedeckung bildet. (vgl. Bundesamt für Landestopographie swisstopo o.J.)

Erscheinungsbild des 3D-Stadtmodells der Schweiz:

- LOD2-Gebäude mit roten Dächern, beigen Fassaden und die topographische Karte der Schweiz als Grundkarte.

(vgl. Bundesamt für Landestopographie swisstopo o.J.)

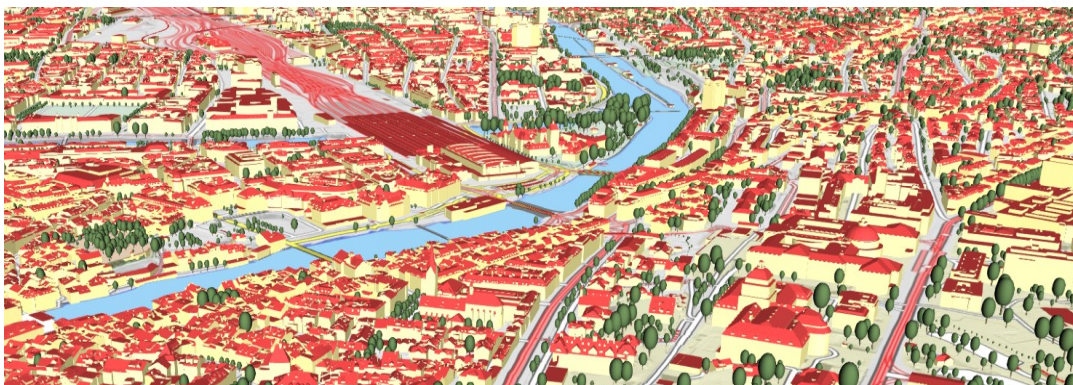


Abbildung 65: 3D-Modell der Schweiz (hier: Zürich) (Quelle: Bundesamt für Landestopographie swisstopo o.J.).

6.1.6. Paris

Von Paris konnte kein offizieller Viewer für das dortige 3D-Stadtmodell gefunden werden. Daher wurde als Beispiel für die durchzuführende Beschreibung eine Visualisierung zur Demonstration der Fähigkeiten von ArcGIS herangezogen, welche anhand von frei verfügbaren Daten ein 3D-Stadtmodell von Paris erzeugt. (vgl. GLAVENU 2020)

Erscheinungsbild des 3D-Stadtmodells von Paris:

- Weiße Gebäude mit Esri-Grundkarte als Bodenbedeckung.

(vgl. GLAVENU 2020)

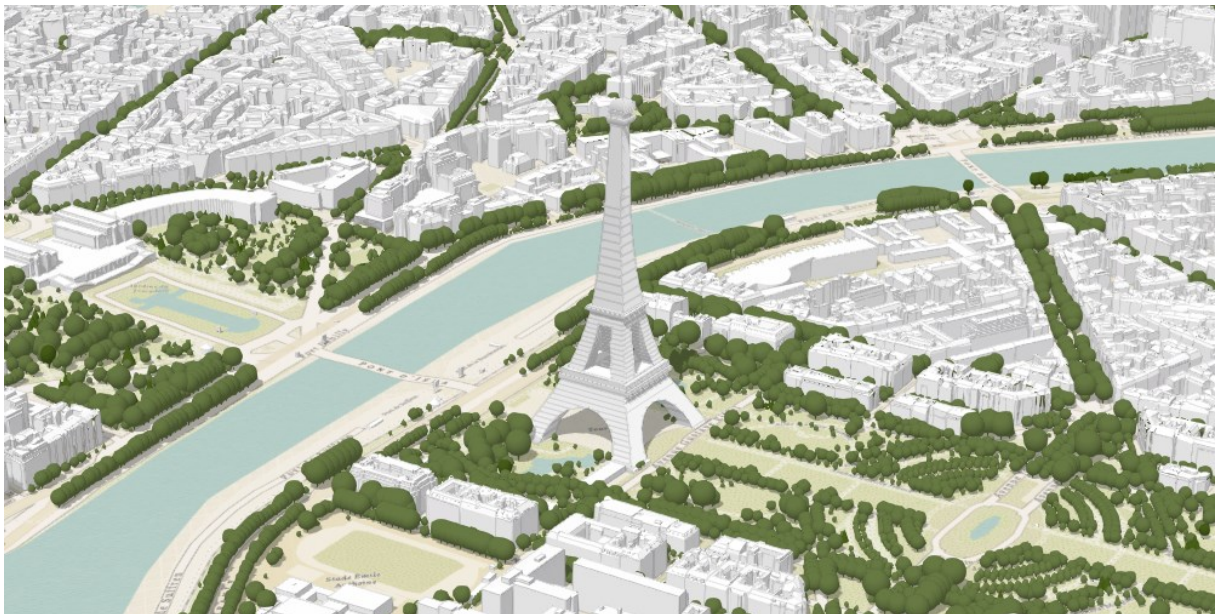


Abbildung 66: Paris in 3D (Quelle: GLAVENU 2020).

6.1.7. New York

Die Stadt New York stellt ihre 3D-Gebäudemodelle als Open Data zur Verfügung. Auf Basis von Daten aus dem Open Data Portal von New York (vgl. City of New York 2017) wurde von der Technischen Universität München ein 3D-Stadtmodell von New York erstellt. (vgl. KOLBE 2016)

Erscheinungsbild des 3D-Stadtmodells von New York:

- Graue LOD1-Gebäude mit Orthofoto als Bodenbedeckung.

(vgl. KOLBE 2016)



Abbildung 67: 3D-Stadtmodell von New York, erstellt von der TU München (Quelle: Kolbe 2016).

6.1.8. Kurzübersicht bestehender Erscheinungsbilder

Zur besseren Übersicht sind die vorhin beschriebenen Erscheinungsbilder noch einmal in einer Tabelle zusammengefasst:

Tabelle 3: Kurzübersicht bestehender Erscheinungsbilder.

Stadt / Land	Gebäudedarstellung	Bodenbedeckung	Vegetation vorhanden
Wien	Rote Dächer, graue Fassaden; Textur	Grundkarte, Orthofoto	nein
Berlin	Weißer Gebäude; Textur; 3D-Mesh	Grundkarte, Orthofoto	ja
Bremen	Graue Gebäude; 3D-Mesh	Grundkarte, Orthofoto	nein
Posen	Rote Dächer, graue Fassaden; Textur; 3D-Mesh	Grundkarte, Orthofoto	ja
Schweiz	Rote Dächer, beige Fassaden	Grundkarte	ja
Paris	Weißer Gebäude	Grundkarte	ja
New York	Graue Gebäude	Orthofoto	nein

6.2. Vergleich verschiedener LOD-Konzepte für Gebäude

Nach der Beschreibung bestehender Erscheinungsbilder von 3D-Stadtmodellen, welche größtenteils auf CityGML basieren, sollen nun verschiedene Konzepte des Level of Detail angeführt und miteinander verglichen werden. Hier werden neben dem LOD-Konzept von CityGML auch zwei weitere Konzepte von BLOM3D™ und NAVTEQ vorgestellt.

Die Beschreibung dieser Konzepte stammt von BILJECKI 2013, welcher im Zuge des Proposals zu seiner Doktorarbeit LOD-Konzepte verschiedener Standards beziehungsweise Firmen miteinander vergleicht.

6.2.1. CityGML

In CityGML sind fünf Levels of Detail definiert (bezogen auf CityGML 2.0):

- LOD0: Gebäude nur als zweidimensionaler Fußabdruck des Umrisses oder der Dachfläche.
- LOD1: Gebäude mit Höheninformation extrudiert als sogenanntes Klötzchenmodell.
- LOD2: Gebäude mit generalisierten Dachstrukturen.
- LOD3: Gebäude mit detaillierten Dachformen und Fassadendetails.
- LOD4: Gebäude mit modellierten Innenräumen.

Bei der Definition der LODs in CityGML ist nicht eindeutig festgelegt, in welchem Ausmaß Texturen bei welchen LODs einzusetzen sind. Es können somit Gebäude in allen LODs texturiert werden, da sich das LOD-Konzept von CityGML vorrangig auf die Geometrie von Gebäuden bezieht.

(vgl. BILJECKI 2013: 14ff)

6.2.2. BLOM3D™

BLOM ist ein Norwegischer Anbieter von 3D-Stadtmodellen und anderen Produkten im Vermessungswesen und in der Fernerkundung (vgl. Blom Norway AS 2021).

Auch bei BLOM3D™ existiert ein eigenes LOD-Konzept, ähnlich wie bei CityGML:

- BlomLOD1™: Gebäude mit Höheninformation extrudiert als sogenanntes Klötzchenmodell und eingefärbt mit der dominanten Farbe.
- BlomLOD2™: Gebäude mit generalisierten Dachstrukturen und eingefärbt mit der dominanten Farbe der Fassaden und Dächer wie in BlomLOD1™.

- BlomLOD3TM: Gebäude wie in BlomLOD2TM, nur mit generischen Texturen anstatt der Farben.
- BlomLOD4TM: Gebäude wie in BlomLOD3TM, nur mit echten Texturen anstatt der Farben, abgeleitet von Luftbildern.

(vgl. BILJECKI 2013: 17)

Das LOD2 von Blom3DTM ähnelt hinsichtlich der Farben sehr stark dem Thema dieser Masterarbeit. Da Blom3DTM jedoch eine proprietäre Software darstellt, sind deren Ressourcen nicht frei verfügbar und müssen somit für diesen Zweck neu entwickelt werden.

6.2.3. NAVTEQ

NAVTEQ war eine US-Amerikanische Firma für Geodaten zu Navigationszwecken (vgl. Here 2021; BILJECKI 2013: 17).

Bei NAVTEQ untergliederten sich die LODs in drei Stufen, jedoch ohne Nummerierung:

- Gebäude mit Höheninformation extrudiert als sogenanntes Klötzchenmodell.
- Zusätzlich zum Klötzchenmodell werden wichtige Gebäude (Landmarks) detaillierter dargestellt.
- Beim höchsten Detailgrad werden Landmarks mit Texturen versehen.

Charakteristisch für die LODs bei NAVTEQ sind die gemischten LOD-Stufen. (vgl. BILJECKI 2013: 17f)

6.3. Entwurf eines neuen Erscheinungsbildes

6.3.1. Allgemeines

Grundsätzlich fällt bei der Betrachtung der verschiedenen 3D-Stadtmodelle auf, dass allen Anschein nach versucht wurde, mit den vorhandenen Daten und Mitteln das größtmögliche Maß an Detailtreue umzusetzen. Es sind keine Ansätze erkennbar, in denen versucht wurde, den Detaillierungsgrad durch Generalisierung bewusst zu reduzieren. Und an dieser Stelle setzt das neue Erscheinungsbild für 3D-Stadtmodelle an. Anstatt beispielsweise das Erscheinungsbild mit Texturen zu verbessern oder LOD3-Gebäudemodelle automatisch zu generieren, werden aus vorhandenen Texturen Farbwerte für Dach- und Fassadenflächen abgeleitet. Zusätzlich wird ein passendes Farbschema für die Bodenbedeckung entworfen. In beiden Fällen werden also die Texturen zu Farbwerten generalisiert. Es wird also durch Generalisierung ein einzelner

Farbwert pro Gebäudefläche ausgewählt und verwendet. Der Sinn dieser Generalisierung ist die einfachere Betrachtung des 3D-Stadtmodells von Wien. (vgl. Spektrum der Wissenschaft Verlagsgesellschaft mbH 2001h)

Vom Detailgrad her entspricht dieser Entwurf in etwa dem BlomLOD2TM, mit dem Unterschied, dass auch die Bodenbedeckung an das Farbschema und den Detailgrad angepasst wird. Zusätzlich ist BlomLOD2TM - wie bereits erwähnt - ein proprietäres Konzept und muss daher, ohnehin neu entwickelt werden, um frei anwendbar zu sein.

Als vollständiges Erscheinungsbild müssten zusätzlich noch Bäume, Tunnels und Brücken visualisiert werden. Um den Rahmen dieser Masterarbeit nicht zu sprengen, beschränkt sich der Autor jedoch auf die Entwicklung eines Erscheinungsbildes für Gebäude und Bodenbedeckung.

6.3.2. Idee

Die Idee, durch Generalisierung bewusst auf Details in einem 3D-Stadtmodell zu verzichten, anstatt dem Trend zu folgen, an immer detaillierteren Modellen zu arbeiten, bekam der Autor durch die Mitarbeit beim OpenStreetMap-Projekt. Hier werden den zweidimensionalen Daten dreidimensionale Informationen wie Gebäudehöhe, Dachform, Dachhöhe und Farbe angefügt, welche dann eine dreidimensionale Darstellung ermöglichen. *Abbildung 68* zeigt die 3D-Visualisierung von OpenStreetMap-Daten mit der Software F4map (vgl. F4 o.J.). Dieses Erscheinungsbild zeigt den zurzeit maximalen Detaillierungsgrad, welcher bei der 3D-Visualisierung von OpenStreetMap-Daten erreicht werden kann. Dass die Gebäude hier in etwa dem Level of Detail 2 entsprechen ist also nur Zufall, da hier (zurzeit) nicht mehr Details möglich sind (*siehe 6.3.3. Versuch einer eigenen LOD-Definition*).

Da - wie die hier bereits durchgeführte Beschreibung zeigt - die bisher existierenden LOD2 Gebäude in diversen Stadtmodellen anscheinend nicht über jene farbige Gestaltung verfügen, wie dies bei der *F4map* der Fall ist, ließ sich der Autor dazu inspirieren, daraus ein neues Erscheinungsbild für 3D-Stadtmodelle abzuleiten.



Abbildung 68: Visualisierung dreidimensionaler OpenStreetMap-Daten durch die F4map (Quelle: F4 o.J.).

6.3.3. Versuch einer eigenen LOD-Definition

Nach der kurzen Beschreibung der LOD-Definitionen von CityGML, BLOM3DTM und NAVTEQ in 6.2. *Vergleich verschiedener LOD-Konzepte für Gebäude* soll nun aus den bestehenden LOD-Konzepten ein neues Konzept für Gebäude entworfen werden, in welchem vor allem Farben und Texturen eine Rolle spielen sollen. Es existieren bereits Versuche, das LOD-Konzept zu verfeinern. Einer davon ist die Dissertation von BILJECKI 2017, welche sich vor allem um die Geometrie von Gebäuden kümmert. Einzig BLOM3DTM und NAVTEQ weisen in ihrer LOD-Definition auch Texturen und Farben auf.

Neue LOD-Definition für diese Masterarbeit:

- LOD0: Nur Geländemodell mit Bodenbedeckung; keine Einzelgebäude, sondern nur bebaute Gebiete.
- LOD1: Nach Höhenattribut extrudierte Gebäude als sogenanntes Klötzchenmodell; je nach Anwendungsfall mit oder ohne Farben, welche von Luftbildtexturen abgeleitet werden können.
- LOD2: Wie LOD1, nur mit generalisierten Dachformen; Gebäudeflächen mit von Luftbildtexturen abgeleiteten Farben.

- LOD3: Farbige Gebäude wie in LOD2, nur mit exakterer Geometrie und Fassadendetails. Der Detailgrad der Geometrie kann auch innerhalb des LOD3 variieren (siehe BILJECKI 2017: 76). Die farbige Gestaltung geschieht hier mittels Farben oder generischen Texturen.

Die Unterscheidung in verschiedene Levels of Detail kann unterschiedlichen Anwendungszwecken dienen. Einerseits könnte man geringere LODs für weiter entfernte Objekte benutzen, andererseits kann man durch die bewusste Reduktion von Details räumliche Zusammenhänge auch besser sichtbar machen.

Im Gegensatz zum LOD0 von CityGML schlägt der Autor vor, keine Gebäudegrundrisse, sondern nur Ortsgebiete als ganze darzustellen, da seiner Meinung nach bei Maßstäben, in denen Einzelgebäude als solche wahrgenommen werden können, bereits LOD1 besser geeignet wäre. Generell ist der Nutzen des bewussten Einsatzes von LOD1 zu hinterfragen, falls ein Modell in LOD2 existiert. Selbst in Erscheinungsbildern, in denen Einzelgebäude gerade noch sichtbar sind, würde die Anwesenheit von einfachen Dachformen die Ansicht vermutlich nicht zu kompliziert erscheinen lassen. Klötzchen hingegen könnten eher ein falsches Bild von Gebäuden vermitteln. Bestehen jedoch nicht die technischen oder finanziellen Möglichkeiten zur automatisierten und damit großflächigen Erfassung von LOD2 Gebäuden, so kann das LOD1 durchaus einen guten Kompromiss darstellen.

Bei LOD3 sollten nach Meinung des Autors auch weitgehend Farben oder Generische Texturen Verwendung finden, da diese frei von störenden Umwelteinflüssen wie Schatten, Vegetation oder Stadtmöbel sind. Für hochdetaillierte Visualisierungen aus der Fußgängerperspektive könnte es jedoch sinnvoll sein, hochwertige Texturen einzusetzen. Diese müssten dann aber von genannten Umwelteinflüssen befreit werden. An einer Software, welche genau solche texturierten Modelle einfach ermöglichen soll, arbeitet die Firma ViewApp (vgl. ViewApp GmbH o.J.). Nur in dieser Qualität wären für den Autor Texturen für LOD3 sinnvoll.

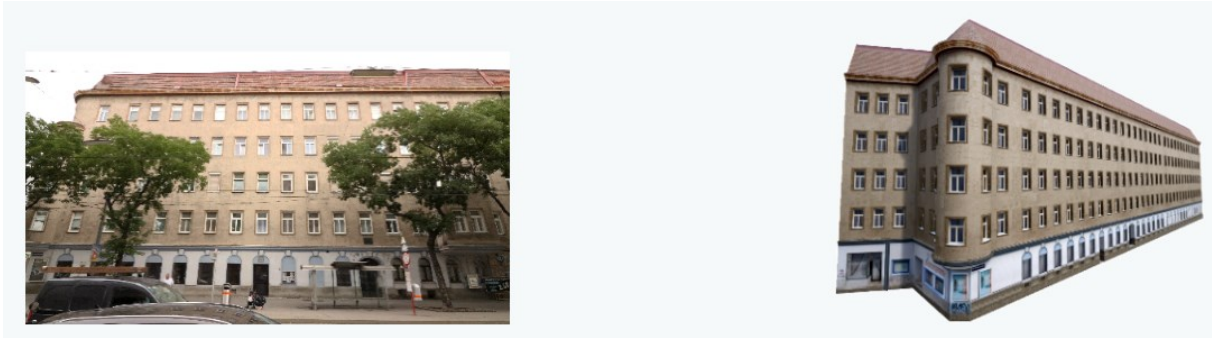


Abbildung 69: Gebäudemodell, texturiert mit einer Software der Firma ViewApp (links: Originalgebäude, rechts: texturiertes Gebäude) (Quelle: ViewApp GmbH o.J.).



Abbildung 70: texturiertes LOD2-Gebäudemodell der Stadt Wien (Quelle: Stadt Wien intern).

Abbildung 69 zeigt, wie sich der Autor die ideale Texturierung vorstellt (im Gegensatz dazu: Abbildung 70). Nur unter diesen Bedingungen würde der Autor Texturen bei 3D-Stadtmodellen einsetzen.

Die vorhergehende Argumentation mag nun den Eindruck vermitteln, der Autor halte nichts von der Verwendung von Texturen bei weniger detaillierten Modellen. Für die reine Visualisierung von LOD2 Gebäuden mag dies zutreffen. Für die realitätsnahe Einfärbung von Gebäuden, wie sie in dieser Arbeit für LOD2 Gebäude durchgeführt wird, sind Texturen jedoch essentiell, da aus ihnen die Farbwerte extrahiert werden.

7. Softwarewahl zur Farbwertextraktion aus Gebäudetexturen

Dieses Kapitel dient dazu, die Auswahl einer geeigneten Software für die automatisierte Farbwertextraktion aus Gebäudetexturen zu treffen und zu begründen. Dabei soll zunächst die Frage geklärt werden, ob es denn nun besser sei, eine Anwendungsapplikation mit graphischer Benutzeroberfläche oder doch lieber eine Programmiersprache zu verwenden. Danach sollen die nach Meinung des Autors bedeutendsten Programmiersprachen miteinander verglichen werden, um damit schließlich die Auswahl von Python zu begründen.

7.1. Anwendungssoftware oder Programmiersprache?

Bevor zwischen diesen beiden Konzepten abgewogen werden kann, müssen die Begriffe Anwendungssoftware und Programmiersprache zunächst einmal definiert und deren Vor- und Nachteile dargelegt werden:

7.1.1. Anwendungssoftware

7.1.1.1. Definition

Anwendungssoftware ist (im Gegensatz zum Betriebssystem, welches den Computer überhaupt erst zum Laufen bringt) Software, welche es dem Endbenutzer ermöglicht, bestimmte Aufgaben, wie zum Beispiel Textverarbeitung oder Bildbearbeitung, durchzuführen (vgl. WEKA MEDIA GmbH & Co. KG o.J.).

Der Autor dieser Arbeit teilt Anwendersoftware in zwei Bereiche:

- Programme mit grafischer Benutzeroberfläche.
- Programme ohne grafische Benutzeroberfläche.

Programme mit grafischer Benutzeroberfläche erlauben es dem Nutzer, einen Computer ohne tiefgreifende Informatikkenntnisse zu benutzen. Funktionen sind größtenteils über leicht verständliche Symbole abrufbar und man muss sich keine Textbefehle merken. (vgl. 1 & 1 IONOS SE 2021)

Programme ohne graphische Benutzeroberfläche hingegen funktionieren meist rein textbasiert. Der Benutzer muss dabei in die sogenannte Konsole (auch Befehlszeile, Terminal oder Interpreter) Befehle in Textform eingeben, anstatt einfach Symbole anklicken zu können. (vgl. ACADEMIC o.J.)

7.1.1.2. Vorteile

Die Vorteile einer graphischen Benutzeroberfläche liegen auf der Hand: Sie sieht meist gut aus und ist im Idealfall auch noch bequem zu bedienen. Programme ohne graphische Benutzeroberfläche kommen heutzutage nur noch selten vor. Es handelt sich hier oft um sehr spezialisierte Anwendungen, welche vom Otto Normalverbraucher eher selten benötigt werden. Auch sie bieten den Vorteil (wenn auch nicht ganz so bequem wie graphische Benutzeroberflächen), dass man die Befehle zumindest in Textform „serviert“ bekommt und man keine logischen Zusammenhänge selbst entwickeln muss.

7.1.1.3. Nachteile

Der Vorteil der einfachen Bedienung und vorgefertigten Funktionen kann sich jedoch für manche Anwender beziehungsweise Anwendungen auch als Nachteil herausstellen. So besitzt man weniger Flexibilität und ist darauf angewiesen, dass benötigte Funktionen vom Entwickler implementiert wurden.

7.1.2. Programmiersprache

7.1.2.1. Definition

„Programmiersprachen sind formale Sprachen, mit denen sich Rechenvorschriften formulieren lassen. Diese Vorschriften können dann von einem Computer ausgeführt werden.“ (Vogel IT-Medien GmbH 2021b)

Solche Rechenvorschriften werden in Form eines Textes, dem sogenannten Quellcode, festgehalten. Damit ein Computer diesen Quellcode ausführen kann, muss er zunächst in eine Sprache umgewandelt werden, welche von der Computerhardware lesbar ist: die sogenannte Maschinensprache. Der Prozess der Umwandlung von Quellcode in Maschinensprache wird Kompilierung genannt. Bei manchen Programmiersprachen geschieht die Kompilierung vor der Ausführung, bei manchen erst während dieser. Letztere benötigen für die Ausführung einen sogenannten Interpreter. (vgl. Vogel IT-Medien GmbH 2021b)

7.1.2.2. Vorteile

Die Vorteile von Programmiersprachen gleichen die Nachteile von Anwendungssoftware aus, indem sie dem Benutzer nahezu alle Möglichkeiten offenlassen. Man ist nicht durch das Vor-

handensein gewisser Funktionen beschränkt, kann jedoch durch sogenannte Programmbibliotheken auf vorgefertigte Bausteine zugreifen. Somit kann man selbst entscheiden, wie viel man selbst programmieren und wie viel man wiederverwenden möchte. Darum sind die klaren Vorteile von Programmiersprachen die hohe Flexibilität und die vielen Möglichkeiten.

7.1.2.3. Nachteile

Die Nachteile von Programmiersprachen liegen auch ganz klar auf der Hand: Man benötigt nämlich im Vergleich zu Anwendungssoftware (im Besonderen mit grafischen Benutzeroberflächen) viel mehr Einarbeitungszeit. Um nur annähernd komplexe Funktionen umzusetzen, wie sie bei Anwendungssoftware üblich sind, werden meist relativ gute Programmierkenntnisse benötigt.

7.1.3. Wahl zwischen Anwendungssoftware und Programmiersprache

Nach Betrachtung der Eigenschaften, Vor- und Nachteile von Anwendungssoftware und Programmiersprachen kann durch Abwägung vorhin genannter Kriterien eine Entscheidung zwischen den beiden getroffen werden.

Bereits im Vorhinein stand fest, dass die Berechnung von Farbwerten auf Grundlage von Texturen zumindest teilautomatisch erfolgen muss, da man nur so ein derart großes Gebiet wie Wien mit so vielen Gebäuden mit vertretbarem Aufwand bearbeiten kann. Hätte die Farbwertextraktion nur für Dächer und rein auf Grundlage von Orthofotos stattgefunden (ohne vorhergehende Texturierung von Gebäuden), so hätte man sie noch gut mit Geoinformationssystemen und den dort integrierten Automatisierungstools durchführen können. Da aber auf Basis einer XML-Datei gearbeitet wird, so ist zumindest eine Software wie FME (vgl. Safe Software Inc 2021) erforderlich, um Daten aus einer komplexen XML-Struktur zu lesen und zu schreiben. Da der Autor über keine Kenntnisse in der Anwendung mit FME verfügt und um die Vorteile der Flexibilität zu nutzen, fiel die Entscheidung gegen Anwendungssoftware und für Programmiersprachen.

7.2. Vergleich ausgewählter Programmiersprachen

Nachdem geklärt wurde, warum eine Programmiersprache zum Einsatz kommen soll, muss noch geklärt werden, welche Programmiersprache am besten für diese Anwendung geeignet ist. Dazu sollen die nach Meinung des Autors gängigsten Programmiersprachen miteinander verglichen werden, um so leichter eine Entscheidung treffen zu können.

7.2.1. Auswahl zu vergleichender Programmiersprachen

Durch Recherche auf vier Internetseiten konnte festgestellt werden, dass Python, Java und C immer unter den Top 5 der dort angeführten Rankings sind (vgl. Hackr.io 2021; Medium 2020; Northeastern University 2019; TIOBE Software BV 2020). In Hackr.io 2021, Medium 2020 und Northeastern University 2019 führt Python das Ranking an.

7.2.2. Kriterien für Vergleich

Die drei ausgewählten Programmiersprachen sollen aufgrund folgender vier Kriterien miteinander verglichen werden:

- **Erscheinungsjahr:** Jahr, in dem die Programmiersprache erfunden wurde.

(vgl. Refsnes Data 2021j; Refsnes Data 2021l; Tutorialspoint 2020a)

- **Anzahl Datentypen:** um hier einen aussagekräftigen Vergleich zu ermöglichen, werden Datentypen zur Speicherung von mehr als einem Objekt (welche besonders in Python zahlreich vorhanden sind) nicht berücksichtigt. Dadurch kann man auch eine Aussage über die Komplexität einer Programmiersprache treffen. Folgende Datentypen, welche laut Refsnes Data 2021k als Python Datentypen gelten, wurden daher vom Autor für den Vergleich nicht berücksichtigt: list, tuple, range, dict, set, frozenset, bytearray und memoryview.

(vgl. Refsnes Data 2021k; Refsnes Data 2021m; Tutorialspoint 2020b)

- **Komplexität:** gemessen an der Anzahl an Zeichen, welche für das berühmte „Hello World“ Programm notwendig sind.

(vgl. DigitalOcean, LLC 2021; PETRI und PETRI 2021; Parewa Labs Pvt. Ltd. o.J.a)

- **Objektorientierung:** dieses Kriterium gibt an, ob es sich um eine objektorientierte Programmiersprache handelt.

(vgl. Refsnes Data 2021n; Refsnes Data 2021o; Parewa Labs Pvt. Ltd. o.J.b)

7.2.3. Vergleich

Tabelle 4: Vergleich der Programmiersprachen Python, Java und C.

	Python	Java	C
Erscheinungsjahr	1991	1995	1972
Anzahl Datentypen	6	8	12
Komplexität	20	100	62
Objektorientierung	Ja	Ja	Nein

7.3. Warum Python?

Aus welchem Grund hat sich der Autor nun dazu entschlossen, für die automatisierte Farbwertextraktion aus Gebäudetexturen Python als Programmiersprache zu verwenden?

Zunächst sei gesagt, dass es einige subjektive Kriterien gibt, welche für die Verwendung von Python sprechen. Bei der Recherche für objektive Kriterien (welche im Anschluss behandelt werden) konnte festgestellt werden, dass Python zu den beliebtesten und einfachsten Programmiersprachen zählt. Zudem kann Python auch zur Automatisierung von Arbeitsabläufen in ArcGIS und QGIS eingesetzt werden und wird auch im Studium Kartographie und Geoinformation gelehrt. Durch letzteren Umstand bestanden auch bereits gewisse Vorkenntnisse, was eindeutig für Python spricht.

Tabelle 4 vergleicht nun objektive Kriterien der drei betrachteten Programmiersprachen miteinander. Dabei fällt C deutlich als älteste Programmiersprache auf und weist auch keine Objektorientierung auf, da dieses Paradigma damals noch nicht etabliert war. Bei den Datentypen sticht auch C mit seinen 12 Datentypen hervor, was es sicherlich nicht zur einfachsten Programmiersprache macht. Um „Hello World“ als Text auszugeben, sind in Python nur 20 Zeichen Code notwendig, während man bei C 62 und bei Java gar 100 benötigt. Würde man die Komplexität einer Programmiersprache nur an dieser Zahl festmachen, so wäre C mehr als dreimal so komplex wie Python und Java gar fünfmal so komplex. Dies spricht daher am eindeutigsten für Python – die Einfachheit des Codes.

8. Automatisierte Farbwertextraktion

Dieses Kapitel beschreibt die automatisierte Farbwertextraktion, welche auf Grundlage von texturierten LOD2-Gebäuden durchgeführt wird.

Zu Beginn wird nochmals kurz auf die texturierten Gebäude des 3D-Stadtmodells von Wien eingegangen, da diese benötigt werden, um eine Farbwertextraktion durchzuführen. Danach folgt eine Beschreibung der Daten, auf welche der Algorithmus später zugreifen wird, um Farbwerte aus Texturen zu extrahieren. Aufgrund der besonderen Bedeutung der Texturierung für die spätere Farbwertextraktion wird erstere noch einmal separat erklärt. Im nächsten Schritt wird auf die Funktionsweise des Python-Codes zur Farbwertextraktion eingegangen. Dabei werden nur Ablauf und Funktionsprinzip beschrieben, ohne auf den konkreten Code einzugehen, da dies den Rahmen dieser Masterarbeit sprengen würde. Der vollständige Code ist dann im Anhang enthalten. Zu guter Letzt erfolgt noch die Präsentation des Ergebnisses der Farbwertextraktion mit einigen Beispielen.

8.1. Ausgangslage

Wie bereits in 2.3. *Das 3D-Stadtmodell von Wien* beschrieben, existiert bei der Stadt Wien neben dem öffentlich sichtbaren 3D-Stadtmodell, in welchem sich alle Gebäude in gleicher Farbgebung präsentieren, noch eine interne Version, welche texturierte Gebäude beinhaltet.

Die texturierten Gebäude wurden von der Firma UVM Systems erstellt. Diese Firma bietet verschiedenste Dienstleistungen im Bereich 3D-Stadtmodelle an. Um die Texturierung durchzuführen, können entweder georeferenzierte Luftbilder oder Bilder von Mobile Mapping Systemen verwendet werden. (vgl. UVM Systems o.J.b)

Für die Stadt Wien wurden nur Texturierungen auf Basis von Georeferenzierten Luftbildern vorgenommen. Das bedeutet, dass auch für die Fassaden nur Ausschnitte aus den Luftbildern zur Verfügung stehen. Diese Ausschnitte ergeben sich aus dem sonst eher unerwünschten Bildsturz (siehe 4.3.6. *Bildsturz*), welcher nun als Textur auf die Fassadenflächen projiziert wird.

In *Abbildung 71* und *Abbildung 72* ist das LOD2-Gebäudemodell von Wien (LOD2.1 und LOD2.4) einmal ohne und einmal mit Texturen dargestellt. Das Ergebnis dieser Masterarbeit soll dabei vom Detaillierungsgrad der Gebäudeflächen her zwischen den beiden Abbildungen liegen.

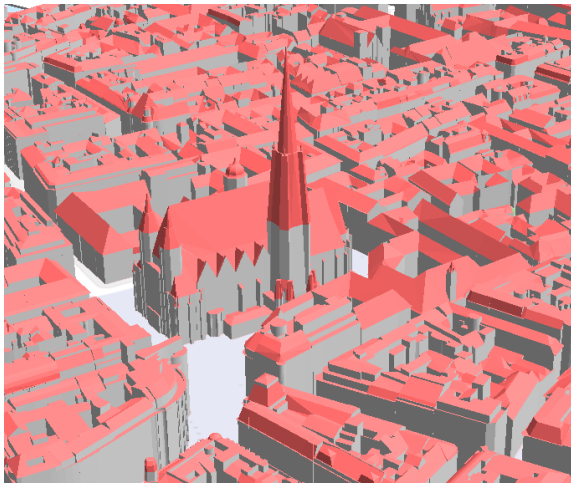


Abbildung 71: LOD2.4 Gebäudemodell von Wien ohne Texturen (Quelle: Stadt Wien o.J.a).



Abbildung 72: texturiertes LOD2.1 Stadtmodell von Wien (Quelle: Stadt Wien intern)

8.2. Ausgangsdaten

Der Texturierung selbst, welche von UVM Systems durchgeführt wurde (siehe 2.3.3.4. *City Grid*), liegt das 3D-Stadtmodell von Wien in LOD2 (siehe 18 *LOD2 Dachmodell*) zugrunde. Um die durch die vielen Gebäude der Stadt entstehenden Datenmengen leichter verarbeiten zu können, ist das Stadtgebiet in einzelne Kacheln unterteilt (vgl. Stadt Wien o.J.e).

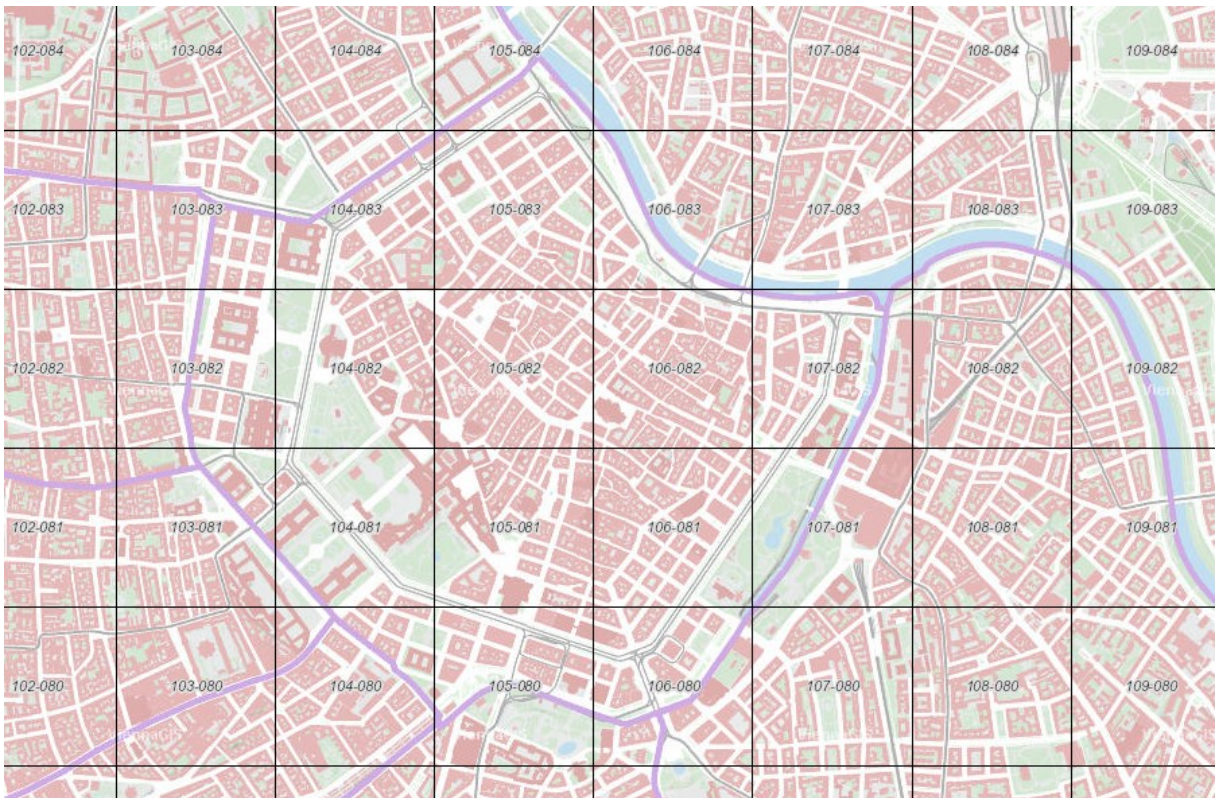


Abbildung 73: Kacheln im Blattschnitt 1:1000 (Quelle: Stadt Wien o.J.e).

Nach erfolgter Texturierung wird für jede Kachel ein Ordner ausgeliefert. Diese Ordner enthalten jeweils eine CityGML-Datei und einen weiteren Ordner „appearance“, welcher pro Gebäude eine JPEG-Datei mit den Texturen (in weiterer Folge Texturdatei genannt) enthält.



Abbildung 74: Texturdatei für ein Gebäude der Kachel 079-072 (Quelle: Stadt Wien intern).

8.3. Texturierung

Betrachtet man [Abbildung 74](#), so wird einem klar, dass eine solche Texturdatei nicht 1:1 auf eine Gebäudefassade „geklebt“ werden kann. Es muss also für jede Fläche am Gebäude ein Bereich von Pixeln ausgewählt werden, welcher auf diese Fläche projiziert werden soll. Die Lokalisierung dieser Pixel erfolgt innerhalb der Texturdatei mit sogenannten Pixelkoordinaten, wobei hier der Nullpunkt links oben liegt. Die Pixelkoordinate 85/50 würde beispielsweise 85 Pixel rechts vom und 50 Pixel unter dem Nullpunkt liegen.

Um nun die gewünschten Pixel (begrenzt durch ein imaginäres Polygon aus Pixelkoordinaten) auf eine Gebäudefläche projizieren zu können, muss ein Zusammenhang zwischen Eckpunkten der Gebäudefläche und Pixelkoordinaten in der Texturdatei hergestellt werden. Dazu werden die gewünschten Punkte einer Gebäudefläche in der CityGML-Datei als Koordinaten der Texturdatei angegeben. Diese Koordinaten entsprechen daher nicht der Lage der Punkte im 3D-Stadtmodell, sondern bereits der Lage innerhalb der Texturdatei, wobei sie noch nicht in Pixelkoordinaten angegeben werden, sondern als relative Werte im Wertebereich von 0 bis 1. Der Vorteil dieser relativen Koordinatenangabe liegt darin, dass auch bei veränderter Auflösung der Texturdatei noch immer ein Zusammenhang zwischen den Koordinaten in CityGML und den Pixelkoordinaten in der Texturdatei hergestellt werden kann.

```

1  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2  <!-- Written by 3D City Database Importer/Exporter, version "3.2-build1" -->
3  <!-- Chair of Geoinformatics, Technische Universität München, Germany, http://www.gis.bgu.tum.de/ -->
4  <!-- Updated by CityGML v10.50.027 (C) UTM Systems GmbH -->
5  <CityModel xmlns:xal="urn:oasis:names:gis:xml:xal:2.0" xmlns:gml="http://www.opengis.net/gml" xmlns:app="http://www.opengis.net/citygml/appearance/1.0" xmlns="http://www.opengis.net/citygml/1.0">
6    <cityObjectMember>
7      <building gml:id="UID_LOD2_000001-14e6c0f4-d14b-4dd8-bea7">
8        <gml:name>000001</gml:name>
9        <gml:boundedBy>
10         <gml:Envelope srsName="urn:ogc:def:crs:OGC:EPSG::31256" srsDimension="3">
11           <gml:lowerCorner>3135.08 341011.55 13.448</gml:lowerCorner>
12           <gml:upperCorner>3173.41 341049.13 29.84</gml:upperCorner>
13         </gml:Envelope>
14       </gml:boundedBy>
15       <creationDate>2016-09-29</creationDate>
16       <app:appearance>
17         <app:Appearance>
18           <app:theme>Bildflug_2013</app:theme>
19           <app:surfaceDataMember>
20             <app:ParameterizedTexture>
21               <app:imageURI>appearance/UID_LOD2_000001-14e6c0f4-d14b-4dd8-bea7.jpg</app:imageURI>
22               <app:mimeType>image/jpeg</app:mimeType>
23               <app:textureType>unknown</app:textureType>
24               <app:wrapMode>wrap</app:wrapMode>
25               <app:borderColor>1.0 1.0 1.0</app:borderColor>
26               <app:target uri="#UID_LOD2_000001-14e6c0f4-d14b-4dd8-bea7_d0e2bfe3-ebd1-4a28-8588-25a95e0cf855_2_poly">
27                 <app:TexCoordList>
28                   <app:textureCoordinates ring="#UID_LOD2_000001-14e6c0f4-d14b-4dd8-bea7_d0e2bfe3-ebd1-4a28-8588-25a95e0cf855_2_poly_0">0.419036 0.943908
29                 </app:TexCoordList>
30               </app:target>
31               <app:target uri="#UID_LOD2_000001-14e6c0f4-d14b-4dd8-bea7_e3c63e11-8b3b-4098-a5b4-d12aba33a73b_2_poly">
32                 <app:TexCoordList>
33                   <app:textureCoordinates ring="#UID_LOD2_000001-14e6c0f4-d14b-4dd8-bea7_e3c63e11-8b3b-4098-a5b4-d12aba33a73b_2_poly_0">0.432842 0.888050
34                 </app:TexCoordList>
35               </app:target>

```

Abbildung 75: Ausschnitt aus der CityGML-Datei der Kachel 106-082. Die relativen Koordinaten sind hier in den Zeilen 28 und 33 zu sehen (Quelle: Stadt Wien intern).

Zusätzlich muss für die korrekte Herstellung eines Zusammenhanges beachtet werden, dass x- und y-Koordinaten zwischen CityGML und Textur vertauscht sind und das Koordinatensystem um 90 Grad im Uhrzeigersinn gedreht werden muss.

Abbildung 76 veranschaulicht die Transformation vom Koordinatensystem der Textur in der CityGML-Datei in das Koordinatensystem der Texturdatei.

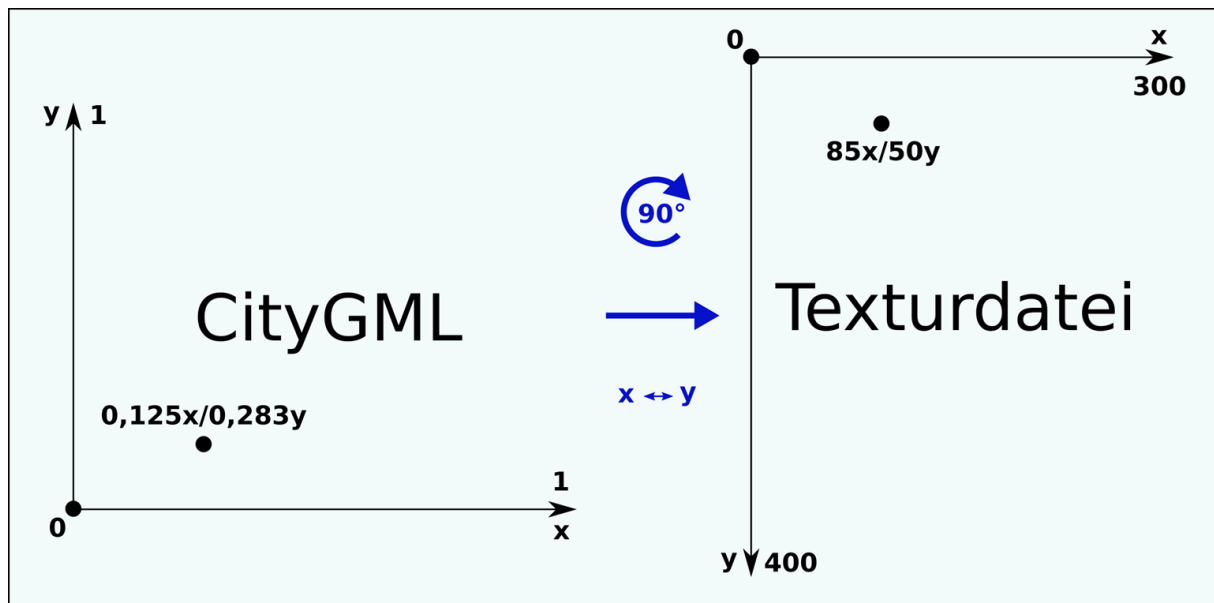


Abbildung 76: Zusammenhang zwischen der Koordinatenangabe in der CityGML-Datei (Werte zwischen 0 und 1) und in der Texturdatei (in Pixel) (Quelle: eigene Abbildung).

Weitere Informationen zur Texturierung befinden sich in 2.2.3.2. *Erscheinungsbild*.

8.4. Theoretische Methoden zur Ermittlung eines durchschnittlichen Farbwertes (Farbwertextraktion)

In der Theorie klingt es erstmal relativ einfach: Man leitet für jede Gebäudefläche einen durchschnittlichen Farbwert ab, doch betrachtet man das Texturierte 3D-Stadtmodell genauer, so ergeben sich dann doch ein paar Fragen:

Schon bei den Fassaden- oder Dachdetails stellt sich die Frage, ob der durchschnittliche Farbwert in Form des arithmetischen Mittels wirklich ein gutes Ergebnis liefern würde. Schließlich fließen ja alle Farbwerte in die Berechnung mit ein. Das Ergebnis liegt irgendwo in der Mitte und ist mitunter ein Farbwert, der der Realität doch nicht mehr so ähnlich sieht.

Auch fällt auf, dass die Gebäude je nach Betrachtungswinkel eine unterschiedliche Helligkeit aufweisen. Müssten die dunklen Gebäude nicht in der Realität anders aussehen?

Zusätzlich zu den Ungereimtheiten bei den Farben der Gebäude könnten noch Fragen zur Vegetation aufkommen. Denn bei genauerer Betrachtung der Gebäude sind auf den Texturen Bäume und Sträucher zu erkennen.

Dieser Abschnitt soll die eben genannten Probleme veranschaulichen und zugleich mögliche Lösungswege vorschlagen.

8.4.1. Repräsentativer Farbwert

Gebäudefassaden und Dächer bestehen meist nicht nur aus einer Farbe. Durch Fenster, Muster und Unterschiede im Anstrich kommt es bei fast jedem Gebäude zu einer Vielzahl an Farben, deren Anzahl im Endeffekt auf eins reduziert werden muss. Selbst einfarbige Fassaden weisen in Wirklichkeit unzählige verschiedene Farbwerte auf. Dieser eine Farbwert, der aus vielen Farbwerten entstehen soll, wird hier „Repräsentativer Farbwert“ genannt.



Abbildung 77: Fassade mit unterschiedlichen Farben und Fenstern (Quelle: Stadt Wien intern).

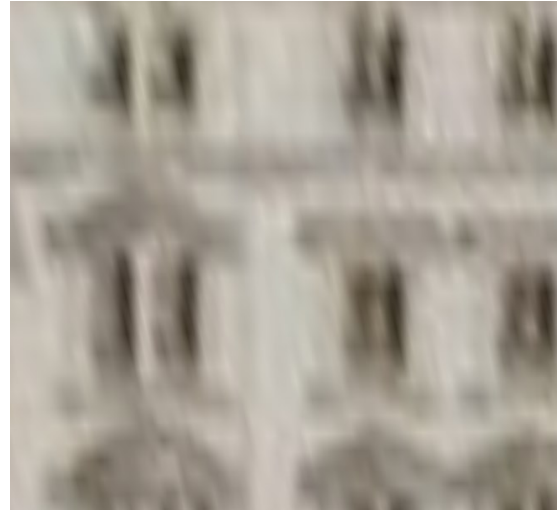


Abbildung 78: einfarbige Fassade mit Fenstern (Quelle: Stadt Wien intern).

Abbildung 78 zeigt eine Fassade mit Fenstern, bei der man sich noch relativ einfach für die Farbe zwischen den Fenstern entscheiden kann.

In *Abbildung 77* sieht es schon etwas anders aus, da die Fassade hier aus mehreren Farben besteht. Welche dieser Farben soll man nun auswählen, wenn man sich für eine Farbe entscheiden muss? Man könnte nun Befragungen und Testungen durchführen, welche Farbe mögliche Probanden bei einer kurzen Betrachtung des Gebäudes spontan nennen würden. Dies könnte aber zu ähnlich vielen Antworten wie die Anzahl der Befragten führen. Im Endeffekt kann diese Entscheidung ohne die geeigneten Hilfsmittel nur subjektiv ausfallen.

Daher wird es hier sinnvoll sein, sich einem Werkzeug wie der Statistik zu bedienen (*siehe 5. Grundlagen der Statistik*). Somit ist durch mathematische Regeln genau definiert, wie ein einzelner Farbwert aus der Masse ausgewählt wird. Da es natürlich auch in der Statistik verschiedene Wege gibt, um ein Ziel zu erreichen, kommt man nicht darum herum, die verschiedenen Methoden zu testen. Die geeignetste Methode kann dann dadurch ausgewählt werden, indem

man das Ergebnis mit dem texturierten Modell vergleicht. Jenes Ergebnis, das letzterem am ehesten gleicht, kann nun als repräsentativ angenommen werden. Man hat zwar hiermit die Subjektivität noch nicht vollkommen aus dem Programm genommen, jedoch hat man eine Entscheidung aufgrund einer fundierten mathematischen Methode getroffen, welche auch leicht reproduzierbar ist.

Zusammenfassend kann gesagt werden, dass es nur durch Vergleich verschiedener statistischer Methoden mit der Originaltextur möglich ist, einen repräsentativen Farbwert für eine Gebäudefläche zu bestimmen.

8.4.2. Licht – Schatten



Abbildung 79: Schattenseite von Gebäuden (Quelle: Stadt Wien intern).



Abbildung 80: Sonnenseite von Gebäuden (Quelle: Stadt Wien intern).

Die Aufnahmen, auf denen die Texturen basieren, wurden bei Sonnenhöchststand und mit Belaubung aufgenommen. Sonnenhöchststand bedeutet, dass sich der aktuelle Standpunkt auf der Erde der Sonne am nächsten befindet (vgl. Time and Date AS o.J.). Das minimiert zwar die Schattenbildung, hebt sie aber nicht gänzlich auf. Und gerade durch die Belaubung der Bäume kommen wieder zusätzliche Schattenquellen hinzu. Somit kommt es vor, dass die nordexponierten Gebäudeflächen auf den Texturen dunkler abgebildet werden als die südexponierten. Am besten wäre natürlich eine diffuse Lichtquelle, wie sie bei Bewölkung vorhanden ist. Aus rechtlichen und praktischen Gründen konnte für Wien bisher jedoch keine Aufnahme bei Bewölkung stattfinden.

Genau dieses „Licht-Schatten-Problem“ ist ein wichtiger Punkt, warum diese Farbwertextraktion überhaupt durchgeführt wird, denn ein ideales 3D-Stadtmodell soll nach Meinung des Autors zunächst frei von realen Umwelteinflüssen sein. So ist es neutral und kann möglichst

vielfältig eingesetzt werden. Will man später einen Schattenwurf simulieren, dann kann man diesen wieder digital hinzufügen.

Eine erste Möglichkeit, um dem Problem der unterschiedlichen Helligkeiten Herr zu werden ist, einzelne Pixel in Abhängigkeit von ihrem eigenen Farbwert mit mathematischen Funktionen anzupassen und dadurch aufzuhellen beziehungsweise abzudunkeln.

Des Weiteren besteht noch die Möglichkeit, dass man die Änderung der Helligkeit in Abhängigkeit von der Exposition vornimmt.

Generell muss beim Aufhellen beachtet werden, dass sich hinter dem zu dunklen Farbwert nicht immer der gewünschte originale Farbwert befindet. Dadurch kann es passieren, dass die aufgehellten Farben auffallend bunt erscheinen. Natürlich kann der Wahrheitsgehalt einer solchen auffälligen Buntheit nur durch einen Lokalaugenschein vor Ort zu hundert Prozent überprüft werden. Dies ist aber aus Effizienzgründen nicht für ganz Wien möglich. Ergibt ein Vergleich mit unbearbeiteten Flächen, dass die aufgehellten Flächen tatsächlich auffällig bunt sind, so muss man manche Farben ersetzen, um zu einem realistischeren Ergebnis zu kommen. Auch zu dunkle Farbwerte können - statt sie aufzuhellen - generell ersetzt werden. Man könnte nun aufschreien und meinen, man erfinde dadurch ja zufällig Farbwerte! Grundsätzlich sei jedoch gesagt, dass durch Unterbelichtung in der Aufnahme Informationen unwiderruflich verloren gehen. Diese lassen sich aus der betreffenden Aufnahme nicht wieder rekonstruieren. Es muss also entweder der existierende, zu dunkle Wert akzeptiert oder ein neuer Wert gefunden werden. Natürlich werden die zu ersetzenden Werte nicht einfach zufällig ausgewählt, sondern nach gewissen Gesichtspunkten möglichst realistisch rekonstruiert.

8.4.3. Vegetation

Stehen Bäume oder andere Vegetation zu knapp neben einer Fassade, so kommt es vor, dass diese auf der Textur sichtbar sind. In 8.4.2. *Licht – Schatten* wurde bereits darauf eingegangen, dass das 3D-Modell frei von Umwelteinflüssen sein soll. So kann man später entscheiden, welche Details man nun simulieren möchte und welche nicht. So wie die Schatten nicht am 3D-Modell sichtbar sein sollen, so sollen auch Bäume und andere Vegetation unsichtbar bleiben und bei Bedarf als eigenständige Modelle wieder hinzugefügt werden.



Abbildung 81: unerwünschte Vegetation an Gebäudefassade (Quelle: Stadt Wien intern).



Abbildung 82: vollständig mit Vegetation bedeckte Gebäudetexturen (Quelle: Stadt Wien intern).

In *Abbildung 81* ist ein Baum zu erkennen, welcher so nahe an dem betreffenden Gebäude steht, dass er auf der Textur zu sehen ist. *Abbildung 82* zeigt ein Gebäude, welches so stark von Vegetation umschlossen ist, dass kein Teil der wirklichen Fassade mehr sichtbar ist.

Für das Problem mit der Vegetation lässt sich nur schwer eine pauschale Lösung finden. Gerade im hier vorliegenden Fall stehen nur sehr eingeschränkte Informationen über die Vegetation zur Verfügung, da kein Infrarotkanal vorhanden ist. Somit ist man bei der Bearbeitung rein auf die vorhandenen Farbwerte angewiesen. Der große Nachteil daran ist, dass es auch Gebäudeflächen geben kann, die grüne Farbtöne aufweisen. Somit ist ein pauschaler Filter nach Farbtönen unbrauchbar.

Man könnte auch versuchen, aus Testflächen jeweils natürliche und künstliche Grüntöne zu extrahieren und zu analysieren. Sollten die künstlichen Grüntöne mehrheitlich von den natürlichen abweichen, so könnte man einfach Vegetation anhand ihrer charakteristischen Farbwerte identifizieren und herausfiltern.

Als letzte Möglichkeit könnte man das Herausfiltern von Grüntönen nach Exposition in Betracht ziehen. Bei dieser Möglichkeit würde der Hauptzweck der Filterung jedoch eher in der Reduzierung der „unnatürlichen Buntheit“ liegen, als in der „Herausfilterung“ der Vegetation.

Bei all den vorhin genannten möglichen Lösungswegen besteht die Gefahr, dass gewisse Methoden nicht pauschal anwendbar sind. Man kann demnach noch so viele Bedingungen in einen Algorithmus zur Filterung von Vegetation einbringen und wird dennoch nicht alle Möglichkei-

ten berücksichtigen können. Diesem Problem könnte (zumindest teilweise) mit Machine Learning Abhilfe geschaffen werden, was wiederum Gegenstand einer eigenständigen Arbeit sein könnte.

8.5. Funktionsweise des Python-Codes

In diesem Abschnitt wird die Funktionsweise des in dieser Masterarbeit entwickelten Python-Codes dargestellt.

Um den Code möglichst übersichtlich zu gestalten, ist er in vier einzelne Module aufgeteilt:

- **Umrechnung zwischen Farbsystemen:** *colorconverter.py*
- **Umrechnung der Koordinaten:** *coordinates.py*
- **Farbwertberechnung:** *color.py*
- **Farbwertextraktion:** *farbwertextraktion.py*

Die Basis für alle Module ist *farbwertextraktion.py* und stellt jenes Skript dar, welches direkt ausgeführt wird. Von diesem Skript ausgehend werden alle anderen Module aufgerufen. Manche Module sind auch verschachtelt: so wird beispielsweise *coordinates.py* innerhalb von *color.py* aufgerufen, welches wiederum vom Skript *farbwertextraktion.py* aufgerufen wird.



Abbildung 83: Grundlegender Ablauf der Farbwertextraktion (Quelle: eigene Abbildung).

In [Abbildung 83](#) ist der Grundlegende Ablauf der Farbwertextraktion mit den oben aufgelisteten Skripten beschrieben. Auf Basis dieses einfachen Ablaufdiagramms werden nun kurz die wichtigsten Methoden der einzelnen Skripte beschrieben, bevor deren konkrete Funktionen betrachtet werden.

Zu Beginn wird die bestehende CityGML-Datei, welche die texturierten LOD2-Gebäude enthält, ausgelesen. Es werden dabei alle relevanten Informationen in Variablen für die weitere Verwendung zwischengespeichert.

Bei der Umrechnung der Texturkoordinaten in Pixelkoordinaten wird jene Transformation durchgeführt, welche in 8.3. *Texturierung* bereits hinreichend beschrieben wird. Somit können jene Pixel gesondert abgespeichert werden, welche sich innerhalb einer Gebäudefläche befinden, von der später nur ein einzelner Farbwert übrigbleiben soll.

Auf diese Menge an Pixel werden nun verschiedene Methoden angewendet (*siehe 8.5.3. Farbwertberechnung (color.py)*), um daraus einen „durchschnittlichen“ Farbwert zu berechnen.

Wegen der in 8.4.2. *Licht – Schatten* beschriebenen Problematik müssen die zuvor berechneten Farbwerte noch aufbereitet werden. Das heißt, dass bestimmte Farbwerte durch andere ersetzt werden, um die Anzahl der verschiedenfarbigen Flächen pro Gebäude zu minimieren.

Zum Schluss wird eine neue CityGML-Datei erstellt, welche die neu berechneten Informationen enthält.

8.5.1. Umrechnung zwischen Farbsystemen (*colorconverter.py*)

Um die Umrechnung von Farbwerten zwischen den unterschiedlichen Farbsystemen einfacher und übersichtlicher zu gestalten und auch um eine Wiederverwendbarkeit zu gewährleisten, wurden auch die Farbkonvertierungs-Algorithmen in ein eigenes Modul ausgelagert.

Farbwerte können mit diesem Modul zwischen den Farbsystemen RGB, HSV und HEX umgerechnet werden, wobei HEX kein eigenes Farbsystem, sondern lediglich eine andere Schreibweise von RGB darstellt.

Wann immer eine Umrechnung zwischen Farbsystemen notwendig sein sollte, kann dieses Modul aufgerufen werden.

8.5.2. Umrechnung der Koordinaten (*coordinates.py*)

Mit dem Modul *coordinates.py* wird auch die Umwandlung der Koordinaten zu Gunsten der Übersicht aus dem Skript *farbwertextraktion.py* ausgelagert. Die umgerechneten Koordinaten werden später in Polygone umgewandelt. Diese Polygone können dann in einem weiteren Modul *color.py* für den „Point-in-Polygon“ Test verwendet werden, um zu überprüfen, ob Pixel innerhalb der Koordinaten des berechneten Polygons und somit innerhalb einer Gebäudefläche liegen. Hier wird in der gleichen Art und Weise vorgegangen, wie in der Theorie in 8.3. *Texturierung* beschrieben.

8.5.3. Farbwertberechnung (*color.py*)

Das Modul *color.py* dient der Berechnung der Farbwerte. Ihm werden die aus der CityGML-Datei ausgelesenen Daten übergeben, um daraus mit Hilfe der Module *coordinates.py* und *colorconverter.py* für jede Gebäudefläche einen Farbwert zu berechnen.

Bevor ein Farbwert berechnet wird, müssen noch einige Pixel herausgefiltert werden. Zunächst muss überprüft werden, ob ein Pixel innerhalb eines Polygons, welches die jeweilige Gebäudefläche (Dachfläche oder Fassadenfläche) repräsentiert, liegt. Dies geschieht mit Hilfe eines sogenannten „Point-In-Polygon“ Tests.

Da es auch Gebäude mit Innenhöfen gibt, kann es pro Gebäudefläche auch mehrere Polygone, sogenannte „innere und äußere Ringe“ geben. Es dürfen daher nur Pixel für die spätere Farbwertberechnung verwendet werden, welche zwar innerhalb des äußeren Rings, aber außerhalb des inneren Rings liegen.

Sind Gebäude direkt an andere Gebäude angebaut oder werden Gebäudeflächen von anderen Objekten verdeckt, so bezeichnet man diese Flächen als „warme Fassaden“. Warmen Fassaden können keine Texturen zugewiesen werden. Daher werden diese mit einem speziellen Grauton eingefärbt. Um warme Fassaden aus der Farbwertberechnung auszuschließen, wird dieser Grauton herausgefiltert.

Die Farbwerte der nun verbleibenden Pixel müssen mit statistischen Methoden zu einem einzelnen Wert zusammengefasst werden. Hierbei hat sich die K-Means Clusteranalyse als die geeignetste herausgestellt (*siehe 5.4. Clusteranalyse*). In Fällen, in denen aufgrund einer zu geringen Anzahl an Werten keine Clusteranalyse durchgeführt werden kann, wird auf den häufigsten Wert (Modus) einer Gebäudefläche zurückgegriffen (*siehe 5.3.2. Modus (Modalwert)*).

Wie bereits erwähnt, müssen die berechneten Farbwerte noch aufbereitet werden, um unterschiedliche Beleuchtungsverhältnisse auszugleichen. Dazu wird die Helligkeit in Abhängigkeit der Exposition angepasst. Sehr dunkle Flächen werden generell durch einen etwas helleren Wert (Defaultwert) ersetzt.

8.5.4. Farbwertextraktion (*farbwertextraktion.py*)

Dieses Python-Skript *farbwertextraktion.py* dient nicht als Modul wie alle bisher beschriebenen Skripte, denn es ist jenes Skript, welches initial gestartet wird und von welchem alle anderen Module aufgerufen werden. Hier starten alle Prozesse und laufen auch hier wieder zusammen.

Daher werden hier auch Parameter, welche dem Skript beim Start beigegeben werden, verarbeitet. Zusätzlich wird hier auch die Exposition berechnet (oder - wenn vorhanden - aus der CityGML-Datei eingelesen) und der Farbwertberechnung (*color.py*) als Parameter zur expositionsabhängigen Aufbereitung der Farbwerte übergeben.

Nachdem ein Teil der Aufbereitung der Farbwerte schon direkt nach der Farbwertberechnung durchgeführt wurde, wird hier nun jene Aufbereitung vorgenommen, welche die Kenntnis über alle Farbwerte eines Gebäudes benötigt (zuvor war immer nur ein Farbwert zur selben Zeit bekannt). Dazu werden alle Farbwerte der Dachflächen und alle Farbwerte der Fassadenflächen jeweils gemeinsam abgespeichert. So kann pro Gebäude der häufigste Farbwert aller Dachflächen und der häufigste Farbwert aller Fassadenflächen berechnet werden. Dadurch können durch die Beleuchtungssituation bedingte Farbunterschiede innerhalb eines Gebäudes ausgeglichen werden.

Am Ende des Skripts wird eine neue CityGML-Datei erstellt, welche die neuen Farbinformationen beinhaltet.

8.6. Ergebnisse der Farbwertextraktion

In diesem Unterkapitel werden die Ergebnisse der Farbwertextraktion anhand von mehreren Untersuchungsgebieten präsentiert. Es wird auch kurz auf Lösungsansätze eingegangen, welche in bestimmten Untersuchungsgebieten erprobt wurden, sich jedoch nach eingehender Betrachtung als ungeeignet erwiesen haben. Die Untersuchungsgebiete beziehen sich dabei auf einzelne Kacheln (siehe 8.2. Ausgangsdaten).

8.6.1. Hinterhainbach – Kachel 083-091

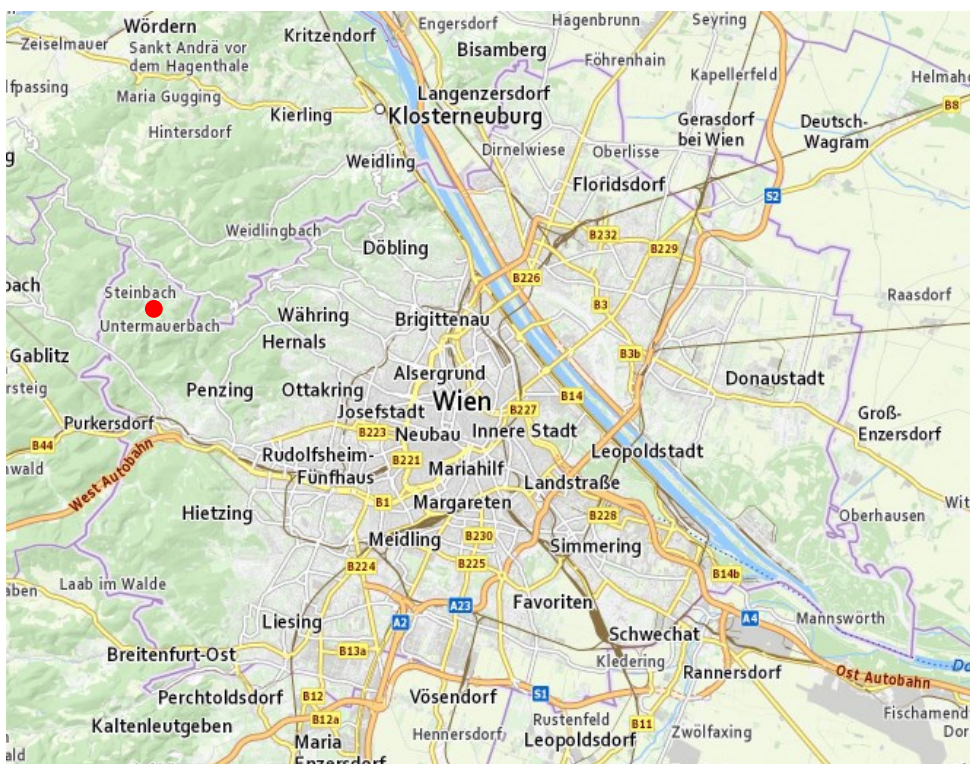


Abbildung 84: Lage von Hinterhainbach innerhalb von Wien (Quelle: Stadt Wien und Österreichische Länder 2016).

Hinterhainbach war das erste Untersuchungsgebiet für die ersten Testläufe des Skripts für die Farbwertextraktion. Hinterhainbach ist ein Dorf, welches im Wienerwald, im Westen von Wien, liegt. Es wurde zufällig ausgewählt, da diese Kachel nur wenige Gebäude beinhaltet, wodurch die Wartezeit beim Durchlauf des Skripts gering gehalten wird.

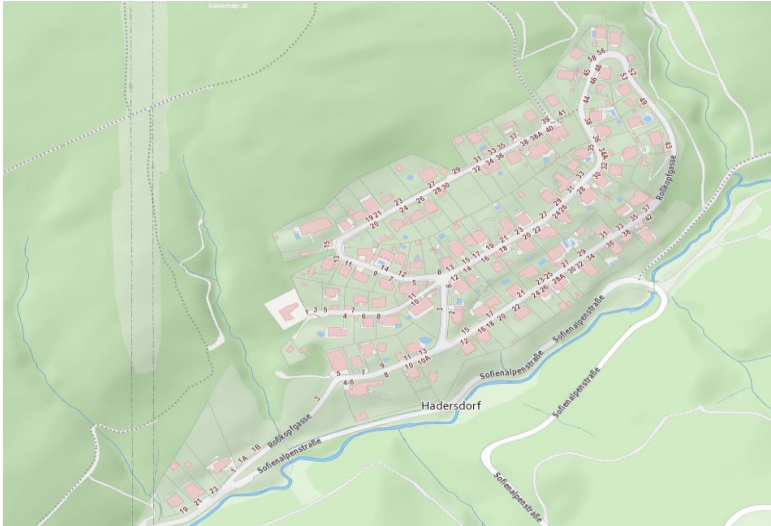


Abbildung 85: Lage der Kachel 083-091 innerhalb von Hinterhainbach (Quelle: Stadt Wien und Österreichische Länder 2016).

Bei den ersten Tests wurden die Farbwerte noch mit dem Mittelwert aller Pixel, welche innerhalb einer Gebäudefläche liegen, berechnet. Danach wurde auf den häufigsten Wert (Modalwert) zurückgegriffen, welcher im Endeffekt - bis auf wenige Ausnahmen (siehe 8.5.3. Farbwertberechnung (*color.py*)) – durch Werte aus der Clusteranalyse ersetzt wurde.

In *Abbildung 86* und *Abbildung 87* ist die Ausgangslage zu sehen, mit welcher der Autor vor der Entwicklung des Skripts zur automatisierten Farbwertextraktion konfrontiert war.

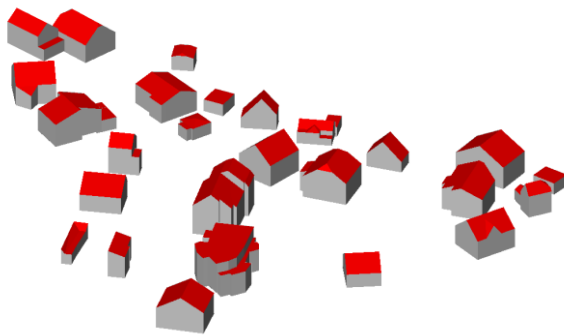


Abbildung 86: Hinterhainbach ohne Texturen (Quelle: Stadt Wien intern).



Abbildung 87: Hinterhainbach mit Texturen (Quelle: Stadt Wien intern).

Abbildung 88 und Abbildung 89 zeigen die Versuche, die Farbwerte mittels Mittelwert oder Modalwert zu berechnen:

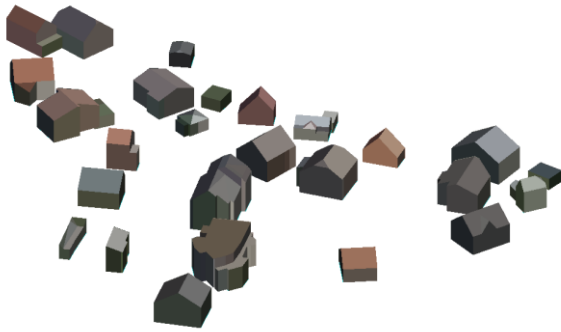


Abbildung 88: Hinterhainbach mit Farbwerten, berechnet aus dem Mittelwert (Quelle: Stadt Wien intern).

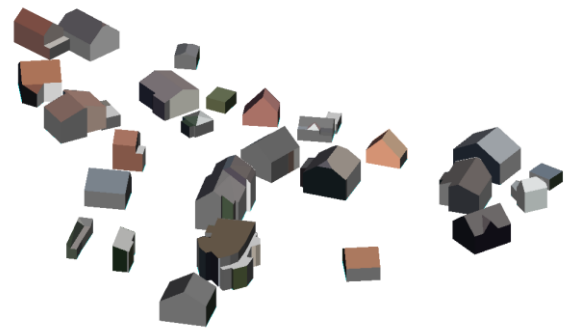


Abbildung 89: Hinterhainbach mit Farbwerten, berechnet aus dem Modalwert (Quelle: Stadt Wien intern).

Wie bereits erwähnt, wurde im Endeffekt doch auf die Clusteranalyse zurückgegriffen:

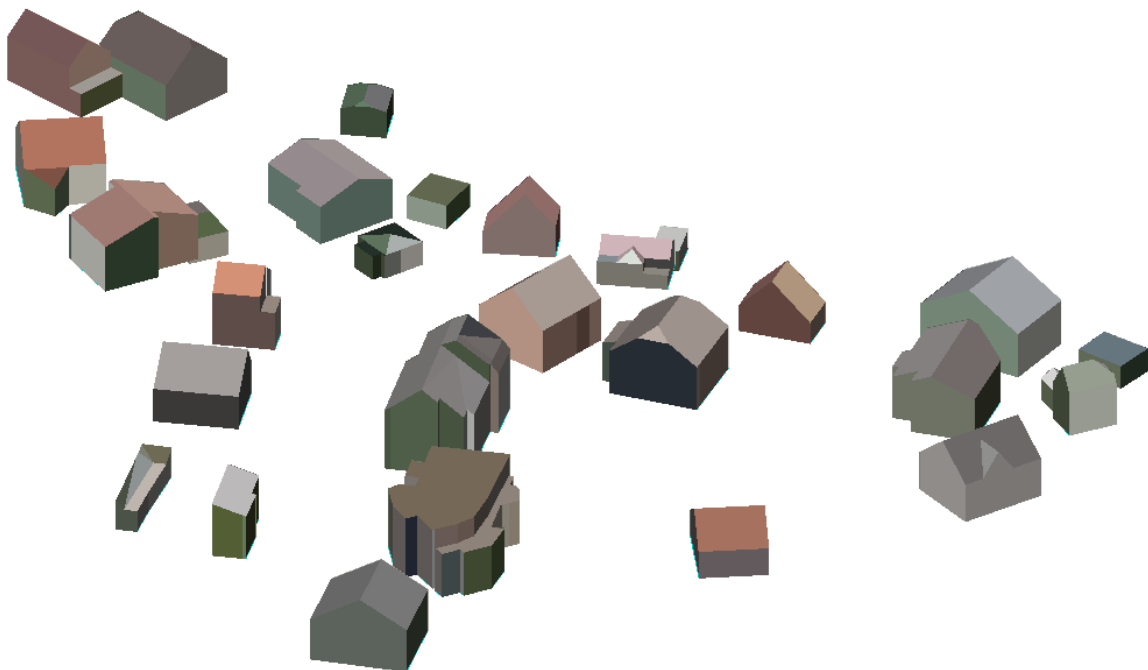


Abbildung 90: Hinterhainbach mit Farbwerten, berechnet mittels Clusteranalyse (Quelle: Stadt Wien intern).

8.6.2. Franz-Josefs-Bahnhof – Kachel 104-086

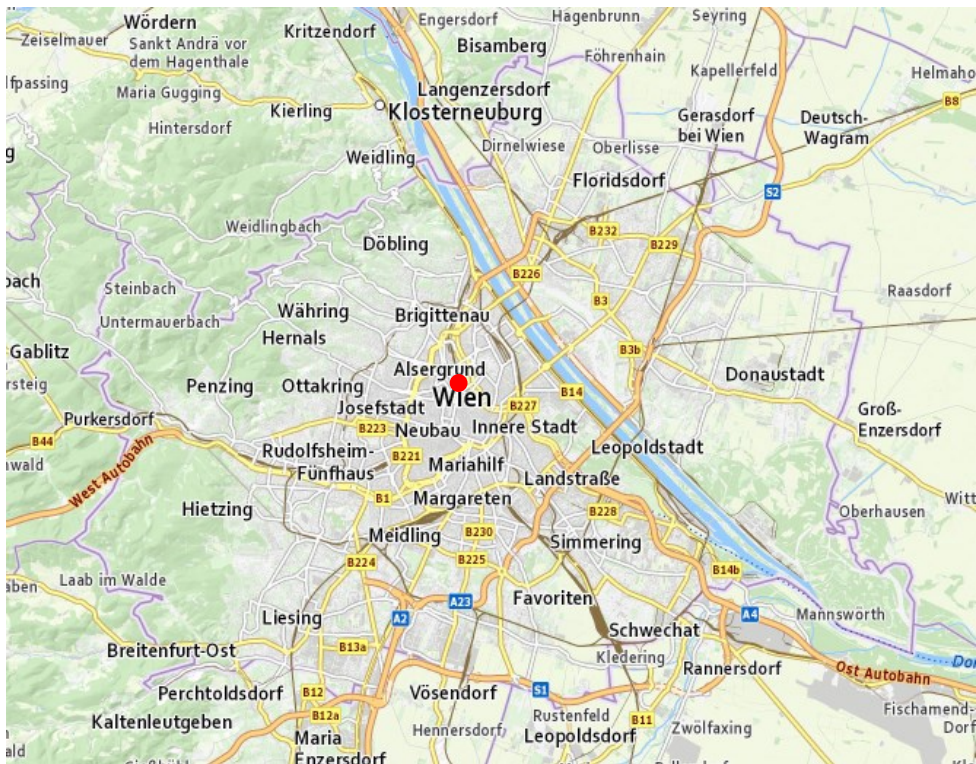


Abbildung 91: Lage des Franz-Josefs-Bahnhofs innerhalb von Wien (Quelle: Stadt Wien und Österreichische Länder 2016).

Als zweites Untersuchungsgebiet wurde die Gegend um den Franz-Josefs-Bahnhof gewählt, da der Autor dieser Arbeit dort wohnt und daher die Ergebnisse besser auf ihre Qualität beurteilen kann. Durch die größere Bebauungsdichte und höhere Komplexität der Gebäude im Vergleich zum Untersuchungsgebiet Hinterhainbach ist die Laufzeit vom Skript hier deutlich höher.



Abbildung 92: Karte des Untersuchungsgebietes um den Franz-Josefs-Bahnhof (Quelle: Stadt Wien und Österreichische Länder).

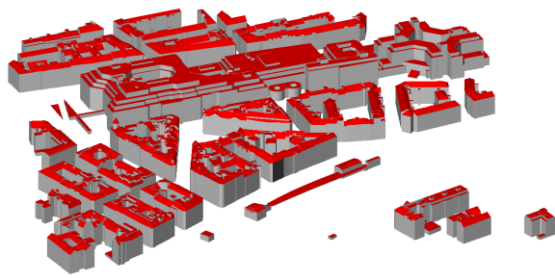


Abbildung 93: Franz-Josefs-Bahnhof ohne Texturen
(Quelle: Stadt Wien intern).

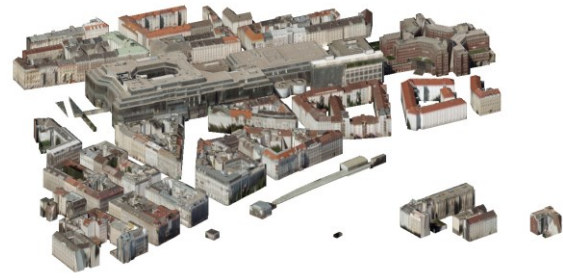


Abbildung 94: Franz-Josefs-Bahnhof mit Texturen
(Quelle: Stadt Wien intern).

Das Untersuchungsgebiet Franz-Josefs-Bahnhof diente unter anderem dazu, das Aufhellen zu dunkler Bereiche zu erproben. Dies wird in dem nun folgenden Exkurs kurz beschrieben:

Exkurs Aufhellen von Farbwerten

Da bei diversen Testläufen die Farbwerte von Gebäuden etwas dunkel erschienen, wurde teilweise ein Punktoperator für die jeweiligen Farbwerte implementiert, welcher mittels einer mathematischen Funktion die Farbwerte – ähnlich wie bei Gradationskurven in Bildbearbeitungsprogrammen – verändert. Ein Punktoperator verändert ein Pixel dabei nur in Abhängigkeit seines eigenen Zustandes. Benachbarte Pixel oder gar das ganze Bild spielen dabei keine Rolle. (vgl. Pietrzyk 2014)

Dazu wurde zunächst die erforderliche Kurve in GIMP ermittelt, indem ein Screenshot von mehreren 3D-Gebäuden, welche bereits mit Farbwerten versehen wurden, geladen und mit der Funktion „Kurven“ angepasst wurde.

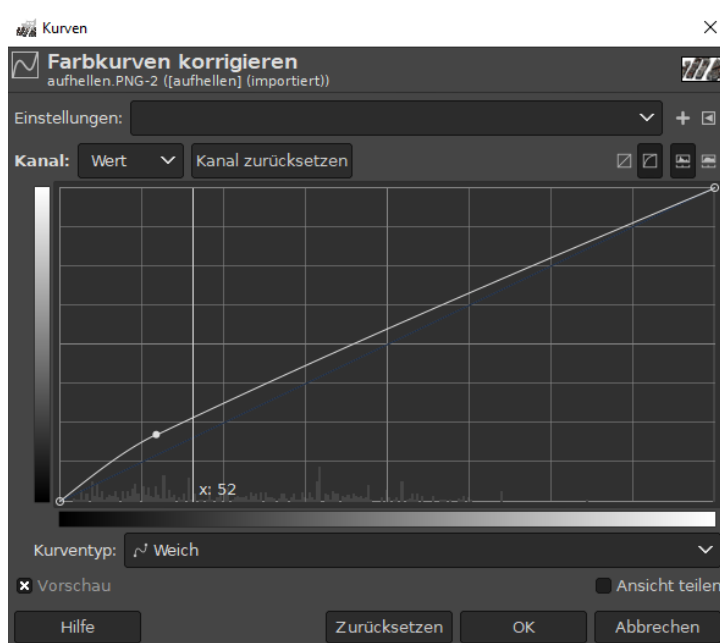


Abbildung 95: Gradationskurve in GIMP (Quelle: Screenshot).

Nun galt es, diese Kurve als Funktionsgleichung anzuschreiben, um sie so im Skript zur Farbwertberechnung verwenden zu können.

Dies geschah mit folgenden Online-Tools:

- Desmos (vgl. Desmos, Inc. 2020)
- MyCurveFit (vgl. MyCurveFit 2020)

In Desmos wurde der Screenshot der Kurve als Hintergrund geladen und in seiner Lage und Ausdehnung an die Koordinatenachsen angepasst. Im nächsten Schritt wurden 5 Punkte (beginnend mit 0/0) mit solchen Abständen, welche die Form der Kurve bestmöglich wiedergeben, über die Kurve gelegt. Dazu wurde eine neue Tabelle erstellt und zunächst x-Werte in den gewünschten Abständen eingetragen. Die y-Werte wurden danach so angepasst, dass die Punkte möglichst genau auf der Kurve zu liegen kamen.

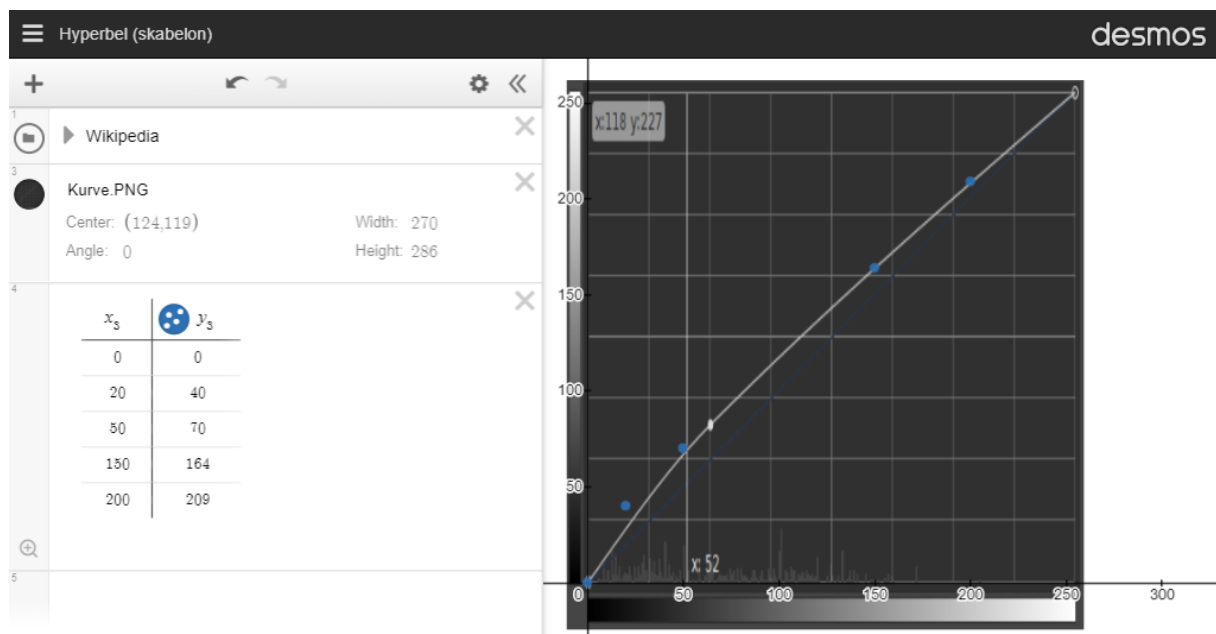


Abbildung 96: Angepasster Screenshot als Hintergrund in Desmos und Punkte aus Tabelle (über Hintergrund ermittelt) (Quelle: Screenshot).

Die in Desmos ermittelten Werte wurden folglich in die Applikation MyCurveFit übertragen, welche Standardkurven den gegebenen Punkten möglichst gut anpasst. Am besten passen (nach momentanem Kenntnisstand) die Sigmoidal Funktion und die Exponential Funktion (Power Curve).

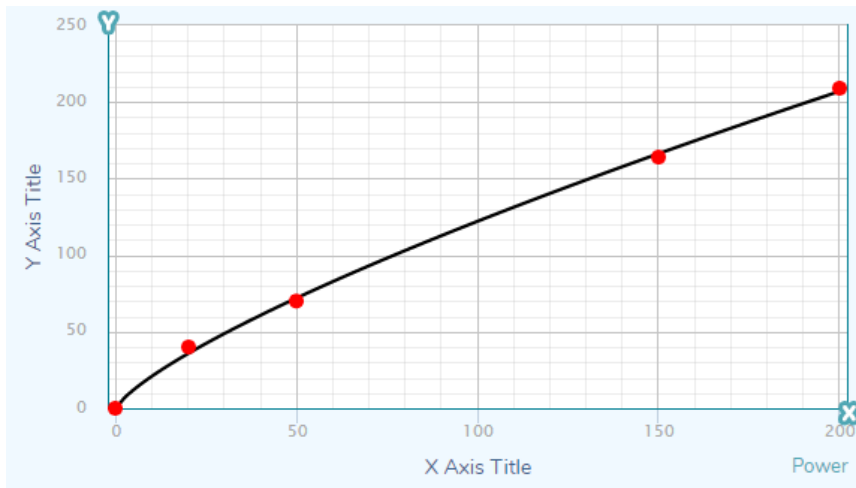


Abbildung 97: Angepasste Kurve (Exponentialfunktion) in MyCurveFit (Quelle: Screenshot).

Bei der Implementierung in Python wird dann jeweils der aktuelle R-, G- und B-Wert als x in die ermittelte Funktion eingesetzt.

Später wurden auch Tests mit einer Parabel durchgeführt, wodurch die Vorgehensweise weniger kompliziert ist. Die Parabel hat auch zur Folge, dass besonders dunkle Bereiche stärker aufgehellt werden. Um ab einem gewissen Punkt die Farbwerte unberührt zu lassen, wurde mit einer if-Verzweigung die Menge der Punkte bestimmt, welche durch den Punktoperator beeinflusst werden.

Auf jeden Fall ist zu beachten, dass durch einen Punktoperator auch das Erscheinungsbild jener Bereiche verändert wird, welche nicht im Schatten liegen und somit auch keiner Bearbeitung bedürften. So kann es mitunter auch zu einer Verschlechterung der Qualität kommen, auch wenn das Ziel, dunkle Bereiche aufzuhellen, grundsätzlich erfüllt wurde. Daher wurde später von der Verwendung eines Punktoperators wieder abgesehen.

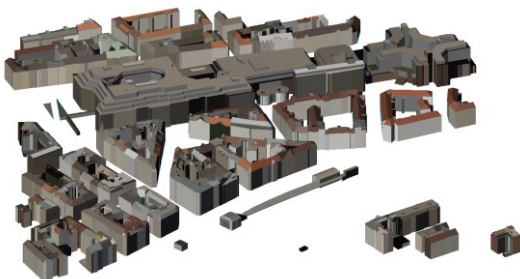


Abbildung 98: Franz-Josefs-Bahnhof mit Farbwerten aus dem Modalwert (Quelle: Stadt Wien intern).

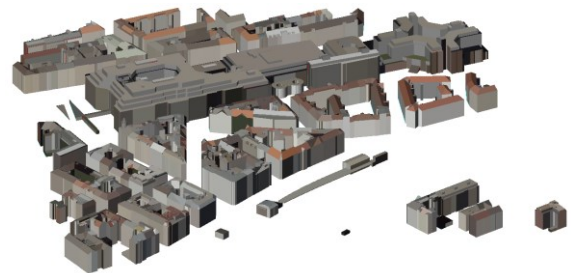


Abbildung 99: mit Farbwerten aus dem Modalwert und geringfügiger Erhöhung der Helligkeit in dunklen Bereichen (Quelle: Stadt Wien intern).

In *Abbildung 99*, in der die Helligkeit mittels Punktoperator erhöht wurde, ist kaum ein Unterschied zu *Abbildung 98* zu erkennen. In diesem Fall kam es zwar zu keiner Verschlechterung wie oben beschrieben, jedoch hält sich die Verbesserung auch in Grenzen.

Auch der Franz-Josefs-Bahnhof sei abschließend noch nach der Farbwertberechnung mittels Clusteranalyse gezeigt:

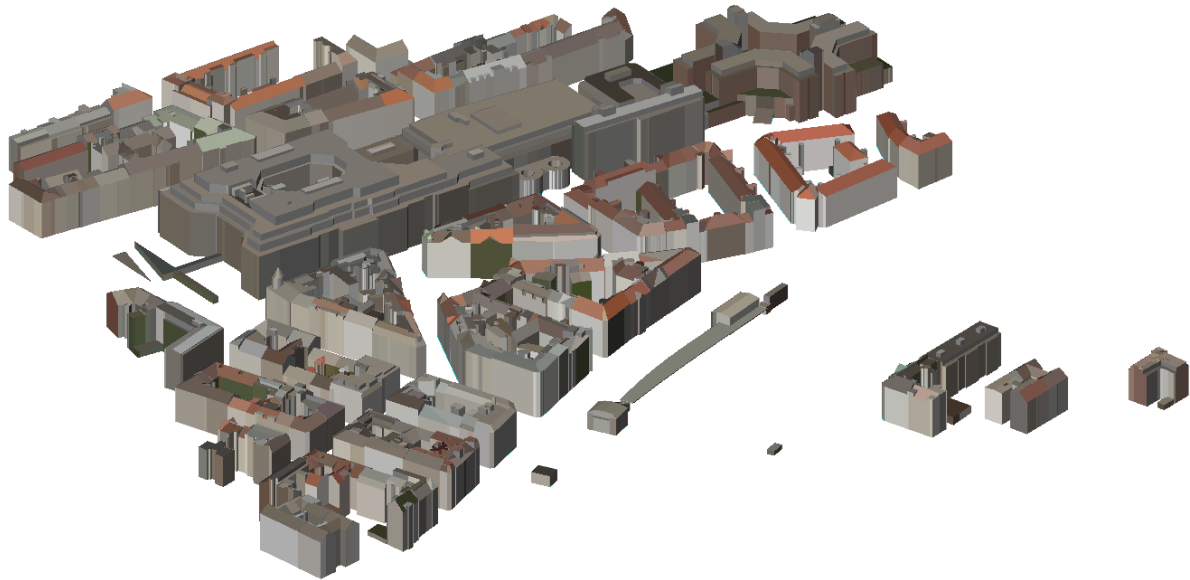


Abbildung 100: Franz-Josefs-Bahnhof mit Farbwerten, berechnet mittels Clusteranalyse (Quelle: Stadt Wien intern).

8.6.3. Richard-Wagner-Platz – Kachel 099-082

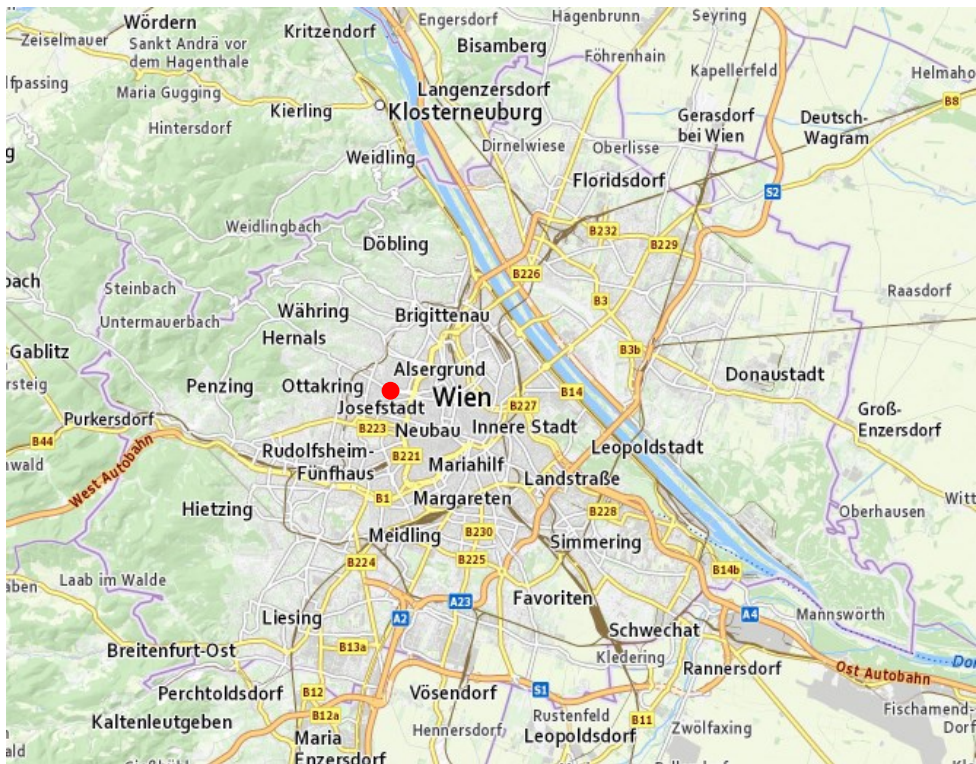


Abbildung 101: Lage des Richard-Wagner-Platzes innerhalb von Wien (Quelle: Stadt Wien und Österreichische Länder 2016).

Das dritte Untersuchungsgebiet liegt rund um den Richard-Wagner-Platz, westlich des Gürtels (Kachel 099-082) und wurde gewählt, da es die durchschnittliche Bebauungsstruktur der Wiener Innenstadt aufweist und zudem auch keine besonders komplexen Objekte enthält, was sich nachteilig auf die Performanz auswirken würde.

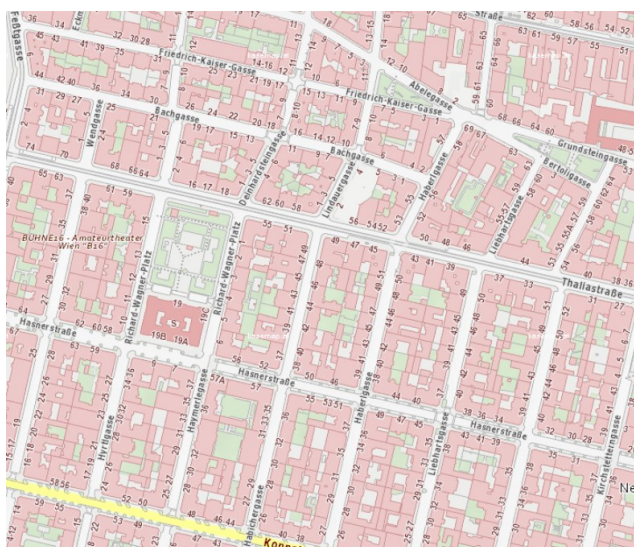


Abbildung 102: Karte des Untersuchungsgebietes um den Richard-Wagner-Platz (Quelle: Stadt Wien und Österreichische Länder 2016).

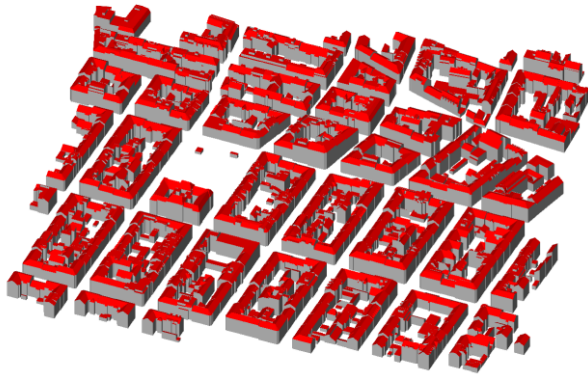


Abbildung 103: Richard-Wagner-Platz ohne Texturen (Quelle: Stadt Wien intern).



Abbildung 104: Richard-Wagner-Platz mit Texturen (Quelle: Stadt Wien intern).

Beim Untersuchungsgebiet Richard-Wagner-Platz ging es vor allem um den Test eines Algorithmus, welcher zu dunkle Pixel aussortieren sollte, falls diese nicht den überwiegenden Teil einer Fläche ausmachten.

In der Praxis hat sich jedoch gezeigt, dass dieser Algorithmus nicht immer ganz schlüssig arbeitet:

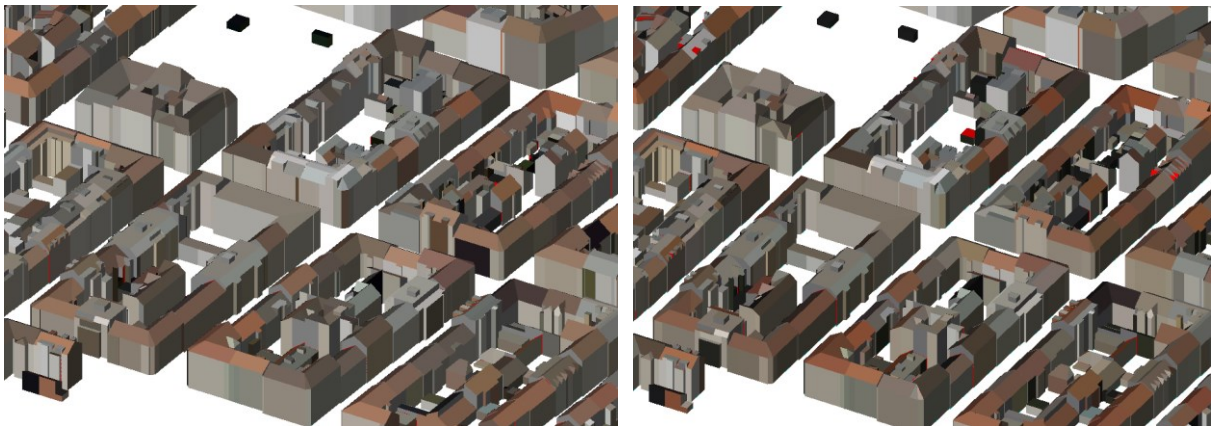


Abbildung 105: 3D-Stadtmodell ohne Algorithmus (links) und mit Algorithmus (rechts) (Quelle: Stadt Wien intern).

Man erkennt schnell, dass (zumindest in diesem Ausschnitt des Untersuchungsgebietes) die Verschlechterungen überwiegen. Es ist auch nur schwer nachvollziehbar, warum im rechten Bild überhaupt Flächen dunkler werden können als sie vorher waren, da zu keinem Zeitpunkt die Zahl der dunklen Pixel steigt. Sie bleibt entweder gleich oder sinkt sogar. Somit sollte der häufigste Wert nicht plötzlich dunkler sein. Auch wurden andere Farbveränderungen beobachtet, welche das Gesamtbild nicht unbedingt positiv beeinflussen.

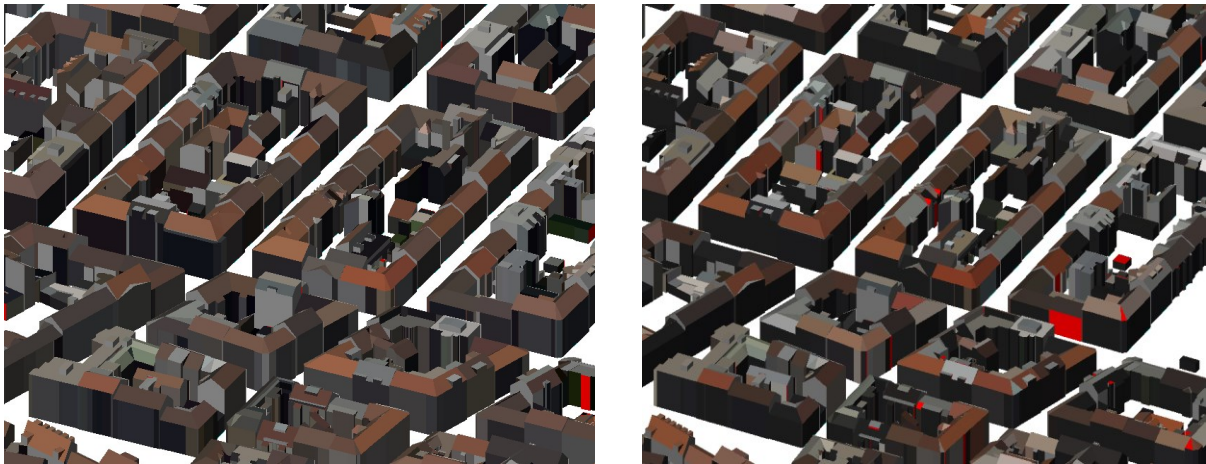


Abbildung 106: 3D-Stadtmodell ohne Algorithmus (links) und mit Algorithmus (rechts) (Quelle: Stadt Wien intern).

Auf der Schattenseite des Modells fällt zudem auf, dass Gebäude noch dunkler geworden sind, obwohl der Algorithmus eigentlich dunkle Flächen minimieren sollte und höchstens dunkle Pixel aus der Menge herausnehmen sollte, wodurch nur dunkle Flächen heller, aber keine hellen Flächen dunkler werden sollten.

Die roten Flächen stellen Flächen dar, für die kein Farbwert berechnet werden konnte. Für diese wurde ein roter Defaultwert vergeben. Es ist ebenfalls zu erkennen, dass auch die Zahl der Defaultwerte durch die Anwendung des Algorithmus gestiegen ist.

Abbildung 107 zeigt nun auch den Richard-Wagner-Platz in seiner endgültigen Version:

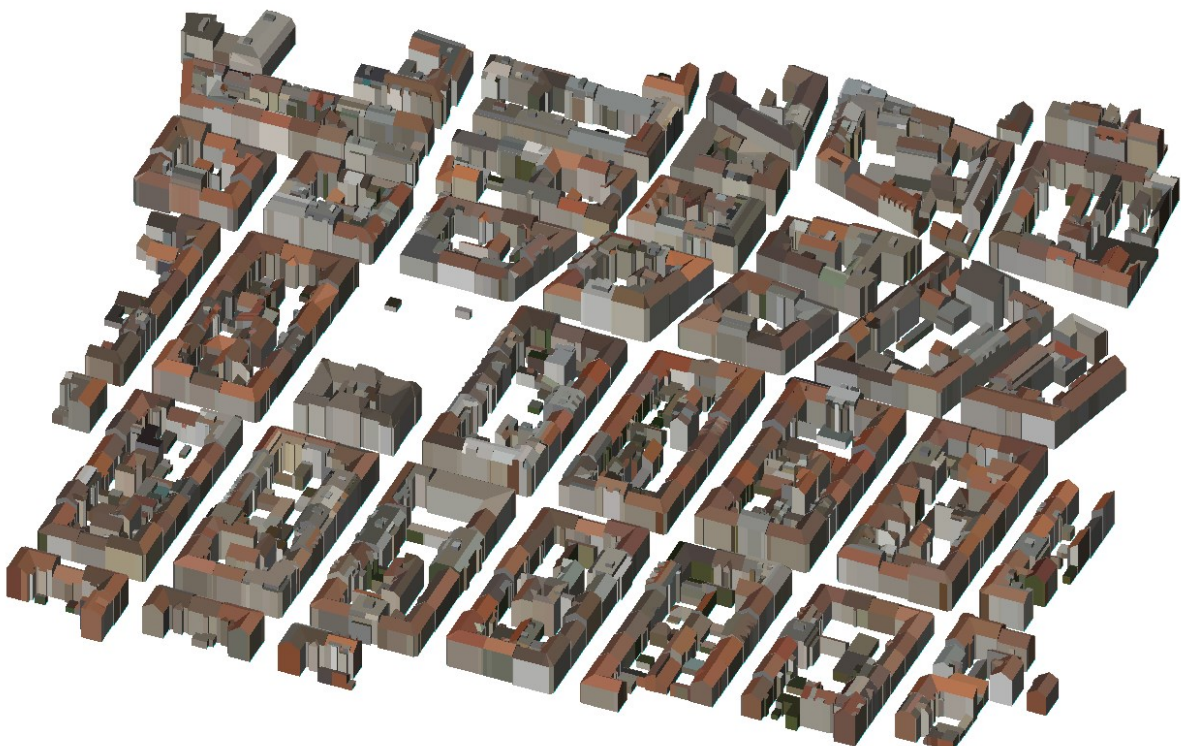


Abbildung 107: Richard-Wagner-Platz mit Farbwerten, berechnet mittels Clusteranalyse (Quelle: Stadt Wien intern).

8.6.4. Stephansplatz – Kachel 106-082

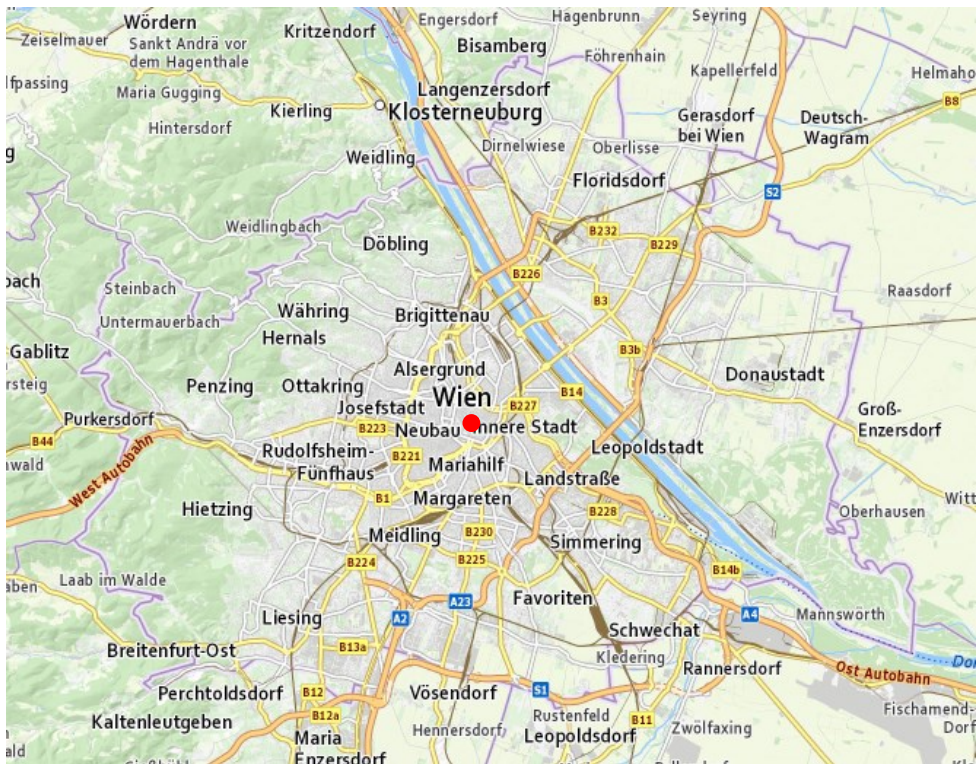


Abbildung 108: Lage des Richard-Wagner-Platz innerhalb von Wien (Quelle: Stadt Wien und Österreichische Länder 2016).

Als viertes und letztes Untersuchungsgebiet diente der Stephansplatz, da der Stephansdom selbst eines der Wahrzeichen Wiens ist und die Umgebung die Innenstadt gut repräsentiert. Somit kann auch hier gut die Qualität der Farbwertextraktion beurteilt werden.



Abbildung 109: Karte des Untersuchungsgebietes um den Stephansplatz (Quelle: Stadt Wien und Österreichische Länder 2016).

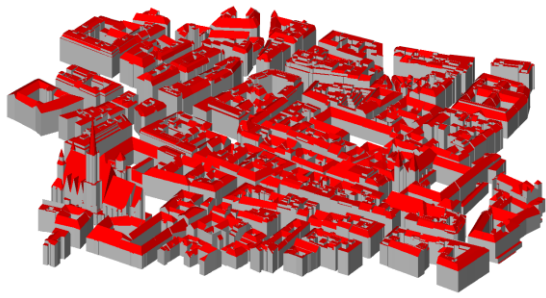


Abbildung 110: Stephansplatz ohne Texturen (Quelle: Stadt Wien intern).



Abbildung 111: Stephansplatz mit Texturen (Quelle: Stadt Wien intern).

Beim Stephansplatz wurden keine großen Neuerungen mehr erprobt, sondern nur kleine Korrekturen an den Farbwerten getestet.

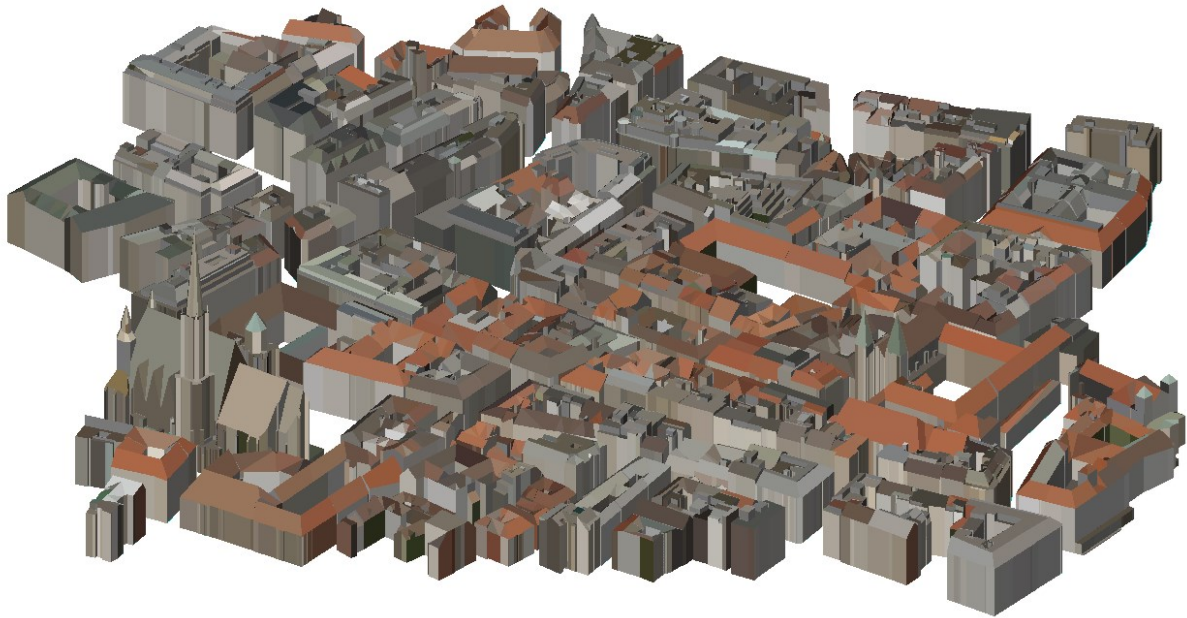


Abbildung 112: Stephansplatz mit Farbwerten, berechnet mittels Clusteranalyse (Quelle: Stadt Wien intern).

9. Darstellung der Bodenbedeckung

Im Zuge dieser Masterarbeit wird zusätzlich zu den mit Farbwerten versehenen Gebäuden auch eine dazu passende Darstellung für die Bodenbedeckung entworfen.

In diesem Kapitel wird zunächst auf allgemeine Aspekte der Bodenbedeckung eingegangen. Es wird erläutert, warum die Bodenbedeckung so wichtig für ein dreidimensionales Stadtmodell ist und warum es für diese Arbeit von Bedeutung ist, eine eigene Darstellung für die Bodenbedeckung zu entwickeln. Danach wird die Entwicklung einer eigenen Bodenbedeckung – passend zu den mit Farbwerten versehenen 3D-Gebäuden – beschrieben und dokumentiert. Es werden dabei die einzelnen Klassen der Bodenbedeckung aufgeführt und das Design mit dem Bezug zur Realität begründet.

Das Ergebnis soll ein Bodenbedeckungsdesign sein, welches ohne Texturen – nur auf Basis von Farben – das reale Erscheinungsbild möglichst realistisch wiedergibt.

9.1. Definition

Als Bodenbedeckung wird die Gesamtheit aller Elemente bezeichnet, die die Erdoberfläche bedecken. Sie kann dabei in verschiedene Kategorien wie zum Beispiel Vegetation, Fels, Gewässer, Verkehrsflächen oder Gebäude eingeteilt werden. Diese Kategorien können je nach Bedarf weiter unterteilt werden. (vgl. Europa Zollportal 2005)

9.2. Notwendigkeit einer Darstellung der Bodenbedeckung

Auch in 3D-Stadtmodellen wird der Bodenbedeckung eine wichtige Rolle zuteil. Ohne diese würden Gebäude (bei Nicht-Vorhandensein eines Geländemodells) in der Luft schweben oder zumindest irgendwo in einem nicht näher definierten Gelände stehen.

Im Folgenden wird ein Vergleich zwischen 3D-Stadtmodellen mit 3D-Gelände und ohne 3D-Gelände durchgeführt. Beide Modelle stammen von der Firma F4 (vgl. F4 o.J.) und enthalten 3D-Gebäude und Bodenbedeckung. Die Daten stammen vom OpenStreetMap-Projekt (vgl. OpenStreetMap o.J.) und sind daher nicht vollständig modelliert. Das Geländemodell wurde vermutlich (es finden sich diesbezüglich keine Angaben auf der Homepage von F4) aus SRTM-Daten erzeugt.

Sowohl in *Abbildung 113*, als auch in *Abbildung 114* ist der gleiche Ausschnitt der Wiener Innenstadt dargestellt. In *Abbildung 114* fehlt die dreidimensionale Geländedarstellung. Trotzdem erkennt man auf den ersten Blick keinen Unterschied zu *Abbildung 113* (ausgenommen Schatten und Beschriftungen).

Da es bei sehr geringen Detaillierungsgraden auch vorkommen kann, dass das Gelände das einzige dreidimensionale Element eines 3D-Stadtmodells darstellt, kann man behaupten, dass die Bodenbedeckung der elementarste Bestandteil eines 3D-Stadtmodells ist. Denn ein Geländemodell ohne Bodenbedeckung könnte man nicht mehr als Stadtmodell bezeichnen.

113

9.3. Art der Darstellung

Die Art der Darstellung der Bodenbedeckung kann je nach Anwendungszweck und vorhandenen Ressourcen unterschiedlich ausfallen.

9.3.1. Bestehende Darstellungsformen der Bodenbedeckung

Die gängigste und auch einfachste Methode, die Bodenbedeckung darzustellen, ist die Visualisierung mittels Grundkarte oder Orthofoto, da diese in den meisten Fällen bereits unabhängig vom 3D-Stadtmodell existieren und somit kein eigener Stil für die Bodenbedeckung entworfen werden muss.

Nachteilig wirkt sich hingegen aus, dass die Gebäudemodelle optisch nicht zwingend zu der Bodenbedeckung passen. Man könnte nun über die genaue Reihenfolge streiten, ob nun die Gebäude zur Bodenbedeckung oder die Bodenbedeckung zu den Gebäuden passen müssen. Fakt ist jedoch, dass - wie bereits erwähnt - die Bodenbedeckung in vielen Fällen bereits als Grundkarte oder Orthofoto existiert. Somit ist klar, dass die Gebäude an die Bodenbedeckung angepasst werden müssen, wenn man ein von der Darstellung her harmonisches 3D-Stadtmodell erzeugen will.

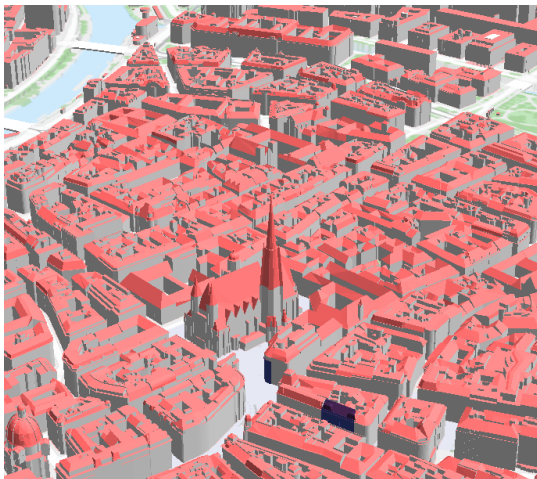


Abbildung 115: 3D-Stadtmodell von Wien mit Gebäuden passend zur Grundkarte (Quelle: Stadt Wien o.J.a).



Abbildung 116: 3D-Stadtmodell von Wien mit Gebäuden passend zum Orthofoto (Quelle: Stadt Wien intern).

Auch beim 3D-Stadtmodell von Wien wurden die Gebäude an die Bodenbedeckung angepasst.

Das Orthofoto als Bodenbedeckung und die 3D-Gebäude ohne Textur würden hingegen nicht so gut zusammenpassen. Genau gleich verhält es sich mit der Kombination aus texturierten Gebäuden und Grundkarte als Bodenbedeckung.

In *Abbildung 117* und *Abbildung 118* wurden die vorhin genannten, designtechnisch ungünstigen Kombinationen umgesetzt.

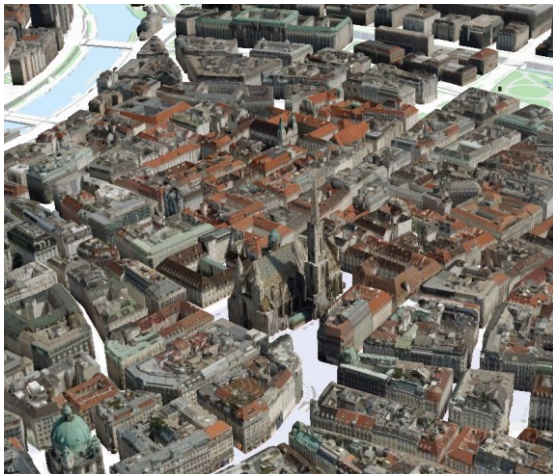


Abbildung 117: 3D-Stadtmodell von Wien mit texturierten Gebäuden und Grundkarte als Bodenbedeckung (Quelle: Stadt Wien intern).

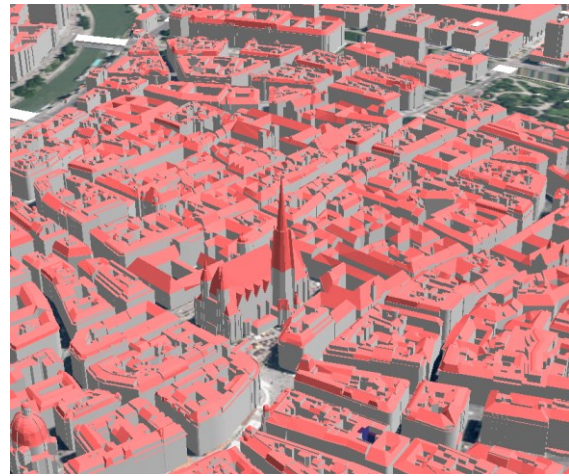


Abbildung 118: 3D-Stadtmodell von Wien mit untexturierten Gebäudemodellen und Orthofoto als Bodenbedeckung (Quelle: Stadt Wien intern).

9.3.2. Neue Darstellungsform der Bodenbedeckung

Will man eine neue Darstellungsform für Gebäude – unabhängig von bestehenden Orthofotos oder Grundkarten – entwerfen, so sollte man auch die Bodenbedeckung anpassen (und somit meist neu entwerfen), wenn Gebäude und Bodenbedeckung zusammen ein harmonisches Bild ergeben sollen.

Dies ist auch bei dieser Masterarbeit der Fall, da zuvor bereits eine neue Darstellungsform für Gebäude umgesetzt wurde, welche nun eine passende Darstellung für die Bodenbedeckung erfordert.

Genau wie bei den Gebäuden beläuft sich das Abstraktionsniveau auch bei der Bodenbedeckung genau in der Mitte zwischen Orthofoto und Landkarte (Stadtplan). Dabei werden die einzelnen Flächen nicht nach Nutzung, sondern nach Bodenbedeckung eingefärbt. Man könnte nun meinen, das Orthofoto mache eben genau das – die Bodenbedeckung wiedergeben. Wie vorhin bereits erwähnt, liegt jedoch das Abstraktionsniveau der zu entwerfenden Darstellung etwas höher als das eines Orthofotos. Es wird zwar versucht, die Darstellungskriterien nicht nach kartographischen Gesichtspunkten, sondern nach realen Gegebenheiten festzulegen, dennoch bleibt es eine Abstraktion (wenn auch nicht auf so hohem Niveau wie bei einer Landkarte oder bei einem Stadtplan), da sich Flächen gleicher Klasse (zum Beispiel Waldflächen) in der Realität trotzdem immer individuell von Objekt zu Objekt unterscheiden.

Der Zweck dieser Darstellungsart ist (wie auch bei den Gebäuden), die Realität zwar möglichst originalgetreu wiederzugeben, aber dennoch von störenden Einflüssen wie Schatten oder Autos zu befreien. Auch sonstige geringfügige Unterschiede innerhalb einzelner Flächen hinsichtlich Struktur und Farbe werden dadurch eliminiert – die Darstellung wirkt somit „ruhiger“ und erfordert bei der Betrachtung weniger Anstrengung. Es werden nun Objekte auf den ersten Blick sichtbar, welche bei einem Orthofoto erst bei genauerer Betrachtung identifiziert werden können.

Zusätzlich sollen bei der Bodenbedeckung nur Dinge dargestellt werden, welche einen weitgehend (unter Vernachlässigung von Gehsteigkanten, Gras, usw.) zweidimensionalen Charakter haben und direkt am Boden liegen. Brücken, Kioske, Stadtmöbel usw. könnten - und werden auch teilweise - als 3D-Modelle dargestellt und wären daher in der Darstellung doppelt vorhanden (vor allem wichtig bei Brücken).

Die neu zu entwerfende Darstellung für die Bodenbedeckung entspricht gewissermaßen einer Landkarte, welche von der Farbgebung her einem Orthofoto gleicht. Das heißt, bei der Farbwahl ist der Bezug zur Realität wichtiger als die kartographischen Gestaltungsprinzipien, auf die sonst geachtet werden sollte. So können zum Beispiel auch größere Flächen in einem eher dunklen oder kräftigeren Farbton erscheinen (falls dies der realen Gegebenheit entspricht), was man in der klassischen Kartographie eher vermeiden würde. Konkret würde man in der Kartographie zum Beispiel Wasserflächen zwar blau einfärben, aber dieses Blau wäre mit Sicherheit heller als in der Realität. Bei Waldflächen oder Wiesen verhält es sich ähnlich: Auf einer Karte werden oft Waldflächen genauso hell, wie Wiesen in der Realität erscheinen, dargestellt.

9.4. Datenquelle(n)

Als Datenquelle für die Darstellung der Bodenbedeckung dient die Wiener Flächen-Mehrzweckkarte, welche auf der Mehrzweckkarte basiert.

9.4.1. Mehrzweckkarte (MZK)

„Die Mehrzweckkarte ist die digitale Stadtkarte von Wien. Sie bildet für das gesamte Stadtgebiet alle markanten, mit der Geländeoberfläche verbundenen Objekte - wie zum Beispiel Gebäude, Fahrbahnen, Straßenbahnen, Gehsteige oder Kanaleinstiege - detailliert ab und wird laufend aktualisiert. Die Mehrzweckkarte ist eine der wichtigsten Planungsgrundlagen der Stadt.

Sie ist beispielsweise die Basis für den Flächenwidmungs- und Bebauungsplan und das dreidimensionale Stadtmodell von Wien.“ (Stadt Wien o.J.d)



Abbildung 119: Mehrzweckkarte (Quelle: Stadt Wien o.J.d).

9.4.2. Flächen-Mehrzweckkarte (FMZK)

„Die Flächen-Mehrzweckkarte ist die digitale Stadtkarte von Wien in einer flächigen Darstellungsform. Aufbauend auf der Mehrzweckkarte zeigt die Flächen-Mehrzweckkarte die kleinräumige Bodennutzung für das gesamte Wiener Stadtgebiet. Aufgrund ihres klar strukturierten Erscheinungsbildes ist die Flächen-Mehrzweckkarte beispielsweise besonders gut für Bezirks- beziehungsweise Stadtpläne geeignet.“ (Stadt Wien o.J.c)



Abbildung 120: Flächenmehrzweckkarte (Quelle: Stadt Wien o.J.c).

9.5. Bodenbedeckungsklassen

Die Bodenbedeckungsklassen entstehen aus der Kombination verschiedener Bodennutzungsklassen und „Gewässer-Namens-Klassen“ der Flächen-Mehrzweckkarte.

9.5.1. Bodennutzungsklassen der Flächen-Mehrzweckkarte

Die Bodennutzungsklassen werden in der Flächen-Mehrzweckkarte (von nun an mit FMZK abgekürzt) in den Attributen F_KLASSE und KLASSE_SUB angegeben. Das Attribut KLASSE_SUB gibt bei übereinanderliegenden Flächen die Bodennutzung der untersten (direkt am Boden liegenden) Fläche an. (vgl. Stadt Wien 2020: 1)

Liste der Bodennutzungsklassen:

- 11 Gebäude
- 12 Überbauung, Verbindungsgang zwischen Gebäuden
- 13 Flugdach
- 14 Glashaus
- 16 Brückenpfeiler
- 19 Sonstige Gebäudefläche (Nebengebäude, etc.)
- 21 Fahrbahn (Haupt-, Nebenfahrbahn, Autobahn, Radweg)
- 22 Verkehrsinsel
- 23 Gehsteig, Geh-, Radweg und Stationsbereich im öffentlichen Gut
- 24 Fußgängerzone
- 25 Fläche für Fußgänger und Radverkehr auf Privatgrund
- 26 Verkehrsfläche auf Privatgrund (wenn vorrangig für Autoverkehr, z.B. Einfahrten, Parkplätze, etc.)
- 27 Schienenbereich
- 28 Selbstständiger Gleiskörper (Straßenbahn, U-Bahn)
- 29 Bahnhofsbereich, Gleiskörper (ÖBB) 30 Zebrastreifen
- 31 Straßenmöbel
- 32 Fahrbahnaufwölbung (Schwelle), Einfahrts-, Auffahrtsrampe
- 33 Parkplatz im öffentlichen Gut
- 39 Sonstige Verkehrsfläche
- 41 Natürliches Gewässer, Badensee
- 42 Schwimmbecken, Biotop (auf Privatgrund)

- 43 Brunnen (im Park), künstliche Wasserfläche (auf öffentlichen Flächen)
 - 44 Gerinne
 - 49 Sonstige Gewässerfläche
 - 51 Hof, Innenhof (bei Gebäuden)
 - 52 Wald, Fläche mit Baumbestand
 - 53 Wiese, naturnahe Grünfläche
 - 54 Feld, Acker, Beet, Baumschule, Obstgarten, landwirtschaftlich genutzte Fläche
 - 55 Weingarten
 - 57 Gräberfeld
 - 58 Grünfläche (kultivierte Wiese, Rasen), sonstige unversiegelte Fläche
 - 59 Befestigte (versiegelte) Fläche auf Privatgrund (wenn nicht 25 oder 26)
 - 60 Baustelle, Baugrube
 - 61 Sportfeld (Rasenfläche Sportplatz, Tennisplatz, Hartplatz, Eislaufplatz)
 - 62 Deponie
 - 63 Lagerplatz
 - 64 Schottergrube
 - 69 Sonstige Grünfläche
 - 71 Mauer, Mauer im Sinne einer Einfriedung, Stützmauer
 - 73 Sockelzaun
 - 74 Stiege, Stufe, Rollstuhl-, Kinderwagenrampe
 - 81 Denkmal
 - 82 Telefonzelle
 - 83 Stationseinrichtung (öffentlicher Verkehr)
 - 84 Kiosk, Würstelstand
 - 86 Portal
 - 91 Rohrleitung
 - 92 Energieversorgung
 - 93 Klärbecken
 - 99 Sonstige nicht zuordenbare Fläche
- (Stadt Wien 2020: 1f)

9.5.2. „Gewässer-Namens-Klassen“ der Flächen-Mehrzweckkarte

Zusätzlich zur Klassifizierung als Gewässer in den Bodennutzungsklassen existiert eine Klassifizierung in namhafte Gewässer Wiens. Diese Klassen wurden (aufgrund fehlender Benennung auf offizieller Seite) vom Autor „Gewässer-Namens-Klassen“ genannt. In der FMZK sind sie als Attribut GEWAESSER angegeben.

Liste der „Gewässer-Namens-Klassen“:

- 0 Keine Kategorisierung
- 1 Donau
- 2 Donaukanal
- 3 Alte Donau
- 4 Neue Donau
- 5 Hafengewässer
- 6 Wienfluss
- 7 Liesing
- 8 Marchfeldkanal
- 9 Sonstiges Fließgewässer
- 10 Stehendes Gewässer

(Stadt Wien 2020: 3)

9.5.3. Einteilung der Bodenbedeckungsklassen

Da es sich um keine konventionelle Karte handelt, sondern um die Bodenbedeckung für eine 3D-Karte, dürfen – wie bereits erwähnt - nur direkt am Boden liegende Objekte und Flächen dargestellt werden. Dies gilt besonders für Brücken, da diese später als dreidimensionale Objekte modelliert werden und sonst doppelt vorhanden wären. Für diese Anforderung ist das Attribut KLASSE_SUB von entscheidender Bedeutung, denn dank dessen Existenz können Brücken einfach ausgeblendet werden.

Somit hat das Attribut KLASSE_SUB eine höhere Priorität als das Attribut F_KLASSE. Letzteres kommt nur zur Anwendung, wenn es kein Attribut KLASSE_SUB gibt.



Abbildung 121: fehlende Eisenbahnbrücke (Quelle: Stadt Wien intern).



Abbildung 122: fehlende Brücken bei Autobahnknoten (Quelle: Stadt Wien intern).

Des Weiteren fließt noch das Attribut GEWAESSER in die Bodenbedeckungsklassen mit ein, welches die vom Autor sogenannten „Gewässer-Namens-Klassen“ repräsentiert. Mit diesen Klassen können verschiedene Gewässer unterschiedlich eingefärbt werden. Dies ist für die realistische Einfärbung der Gewässer besonders wichtig, da sich auch in Wien die Gewässer hinsichtlich ihrer Farbe teilweise deutlich unterscheiden.

Die Farbe von Gewässern wird unter anderem in der Gewässeroptik untersucht. Dabei geht es hauptsächlich um das Verhalten von Licht im und am Wasser. (vgl. Spektrum der Wissenschaft Verlagsgesellschaft mbH 1999)



Abbildung 123: Unterschiedliche Farben von Donau und Neuer Donau (Quelle: Stadt Wien und Österreichische Länder 2016).

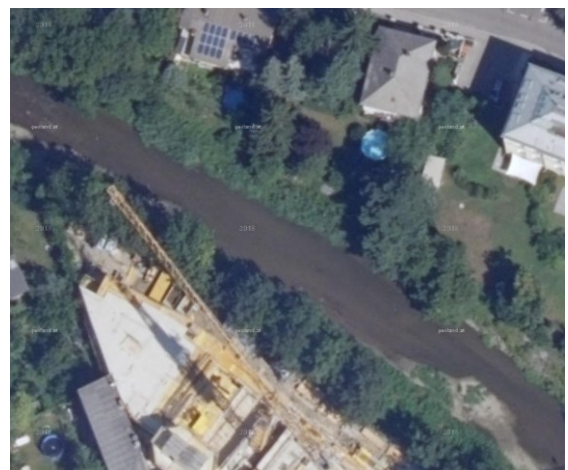


Abbildung 124: Farbe von Wienfluss (Quelle: Stadt Wien und Österreichische Länder 2016).

In weiterer Folge werden nun verschiedene Bodennutzungsklassen und „Gewässer-Namens-Klassen“ aus der FMZK zu neuen Bodenbedeckungsklassen kombiniert. *Tabelle 5* gibt eine Übersicht über die Zuordnung von Bodennutzungsklassen und „Gewässer-Namens-Klassen“ zu den Bodenbedeckungsklassen. Zusätzlich sind auch die dazugehörigen Farben angeführt.

Man kann die Bodenbedeckungsklassen in **Versiegelte Flächen**, **Gewässerflächen** und **Vegetationsflächen** zusammenfassen. Versiegelte Flächen sind **grau**, Gewässerflächen **blau** und Vegetationsflächen **grün** beschriftet.

Tabelle 5: Liste der Bodenbedeckungsklassen (Reihenfolge wie Layerstruktur).

Bodennutzungsklasse oder Gewässer-Namens-Klasse	Bodenbedeckungsklasse	Farbe
92	Energieversorgung	
11, 12, 19	Gebäude	
91	Rohrleitung	
71, 73	Mauer, Sockelzaun	
27	Schiene	
28, 29	Gleisbereich	
30	Zebrastreifen	
74	Stiege	
22	Verkehrinsel	
23, 24, 25, 31, 81, 82, 83, 84, 86	Fußgängerfläche	
13, 21, 26, 32, 33, 39, 63	Verkehrsfläche	
16	Brückenpfeiler	
51, 59	Versiegelte Fläche	
99	Sonstige Fläche	
60	Baustelle	
62	Deponie	
64	Schottergrube	
1	Donau	
2	Donaukanal	
3	Alte Donau	
4	Neue Donau	

5	Hafengewässer	
6	Wienfluss	
7	Liesing	
8	Marchfeldkanal	
9	Natürliches Fließgewässer	
10, 41, 49	Natürliches stehendes Gewässer	
43, 9	Brunnen, künstliches Fließgewässer	
10, 42	Pool, Biotop	
93	Klärbecken	
61	Sportplatz	
52	Wald	
53	Wiese	
58	Grünfläche	
69	Sonstige Grünfläche	
55	Weingarten	
14, 54	Acker	
57	Gräberfeld	

9.5.4. Farbgebung der Bodenbedeckungsklassen

Wie bereits in 115 erwähnt, orientiert sich die Farbgebung der Bodenbedeckung an den Farben der Realität. Als Vorbild dient dabei das Orthofoto aus der österreichischen Verwaltungsgrundkarte basemap.at (vgl. Stadt Wien und Österreichische Länder 2016).

Da die einzelnen Klassen in der Realität in sich relativ inhomogen sind, ist jede Art von realitätsnaher Farbgebung ein Kompromiss. Es muss für jede Klasse eine Farbe gefunden werden, welche der Realität im Durchschnitt am ehesten entspricht. Diese Methode liefert daher in manchen Bereichen relativ realistische Ergebnisse, in anderen wiederum sieht die Bodenbedeckung sehr künstlich aus.

Erschwerend kommt in manchen Fällen noch hinzu, dass die Farben auch von Klasse zu Klasse zueinander passen sollten.

Dieser Abschnitt soll einen Überblick über die Gedankengänge des Autors während der Erstellung des Designs für die Darstellung der Bodenbedeckung geben. Es wird auf die oben genannten Schwierigkeiten eingegangen und erklärt, wann und warum welche Farben zum Einsatz kommen. Dabei werden nicht alle Bodenbedeckungsklassen einzeln beschrieben, sondern nur jene exemplarisch aufgegriffen, welche nach Meinung des Autors von besonderer Bedeutung für die Erläuterung des Farbschemas für die Bodenbedeckung sind.

Versiegelte Flächen

Gerade bei den versiegelten Flächen ist das Verhältnis der Farben zueinander relativ wichtig, da die Farben nicht nur zueinander passen, sondern auch deutlich unterscheidbar sein sollen. Gehsteig und Fahrbahn beispielsweise können nur durch verschiedene Farben unterschieden werden, da die Gehsteigkanten nicht dreidimensional ausgebildet sind. So ist es daher notwendig -

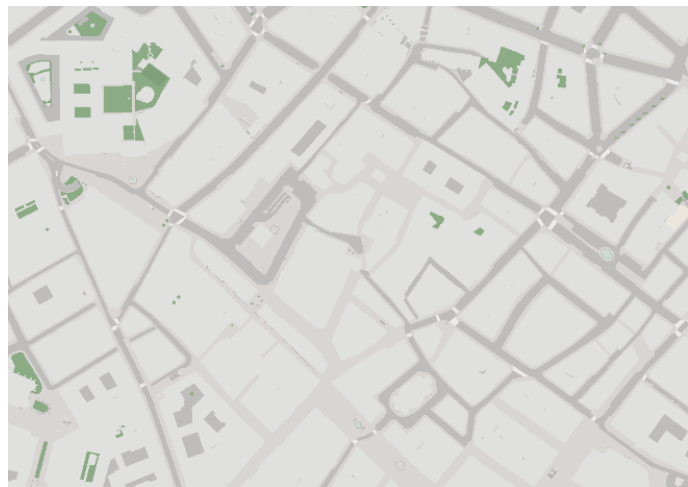


Abbildung 125: Versiegelte Flächen (Quelle: Stadt Wien intern).

auch wenn an manchen Stellen Gehsteig und Fahrbahn in der Realität die gleiche Farbe aufweisen - dass sie sich aufgrund der Notwendigkeit der Unterscheidbarkeit im Modell trotzdem farblich unterscheiden. Das Gleiche gilt natürlich auch für die Unterscheidung zu anderen versiegelten Flächen, wie zum Beispiel Stiegen oder Lagerplätzen (um nur einige zu nennen). Flächen mit ähnlicher Nutzungsart werden in weiterer Folge zusammengefasst. So fallen alle Flächen, die von Autos befahren werden, in die Klasse *Verkehrsfläche*.

Verkehrsflächen werden in einem dunkleren Grauton eingefärbt als andere versiegelte Flächen, da auf dem Orthofoto von basemap.at (vgl. Stadt Wien und Österreichische Länder 2016) beobachtet werden kann, dass Verkehrsflächen für den Autoverkehr häufig dunkler erscheinen, als Fußgängerflächen. Da dies besonders bei Hauptstraßen der Fall ist, könnte dies an dem durch den starken Verkehr vermehrt auftretenden Gummiabrieb liegen. Es könnte sich aber auch einfach nur um frischen Asphalt handeln.

Fußgängerflächen erscheinen auf dem Orthofoto hingegen meist in einem eher helleren Grauton, was - wie bei den *Verkehrsflächen* - entweder auf die Nutzungsart (kein Gummiabrieb) oder auf das Material zurückzuführen sein könnte.



Abbildung 126: Tendenz zu dunkleren Verkehrsflächen und helleren Fußgängerflächen (Quelle: Stadt Wien und Österreichische Länder 2016).



Abbildung 127: Nachahmung der in Abbildung 122 beschriebenen Tendenz (Quelle: Stadt Wien intern).

Gewässerflächen

In 9.5.3. *Einteilung der Bodenbedeckungsklassen* wurde bereits darauf eingegangen, dass Gewässer wie in der Realität in verschiedenen Farben dargestellt werden. Diese Unterscheidung wird durch die „Gewässer-Namens-Klassen“ der FMZK ermöglicht.

Besonders wichtig ist die farbliche Unterscheidung der Gewässerflächen bei Donau, Alter Donau und neuer Donau, da diese mit Abstand die markantesten Gewässer Wiens sind und daher auch zur Orientierung sehr wichtig sind. Für die Orientierung ist nicht nur die Lage von Bedeutung, sondern auch die Farbe, da diese erst den Wiedererkennungswert schafft.

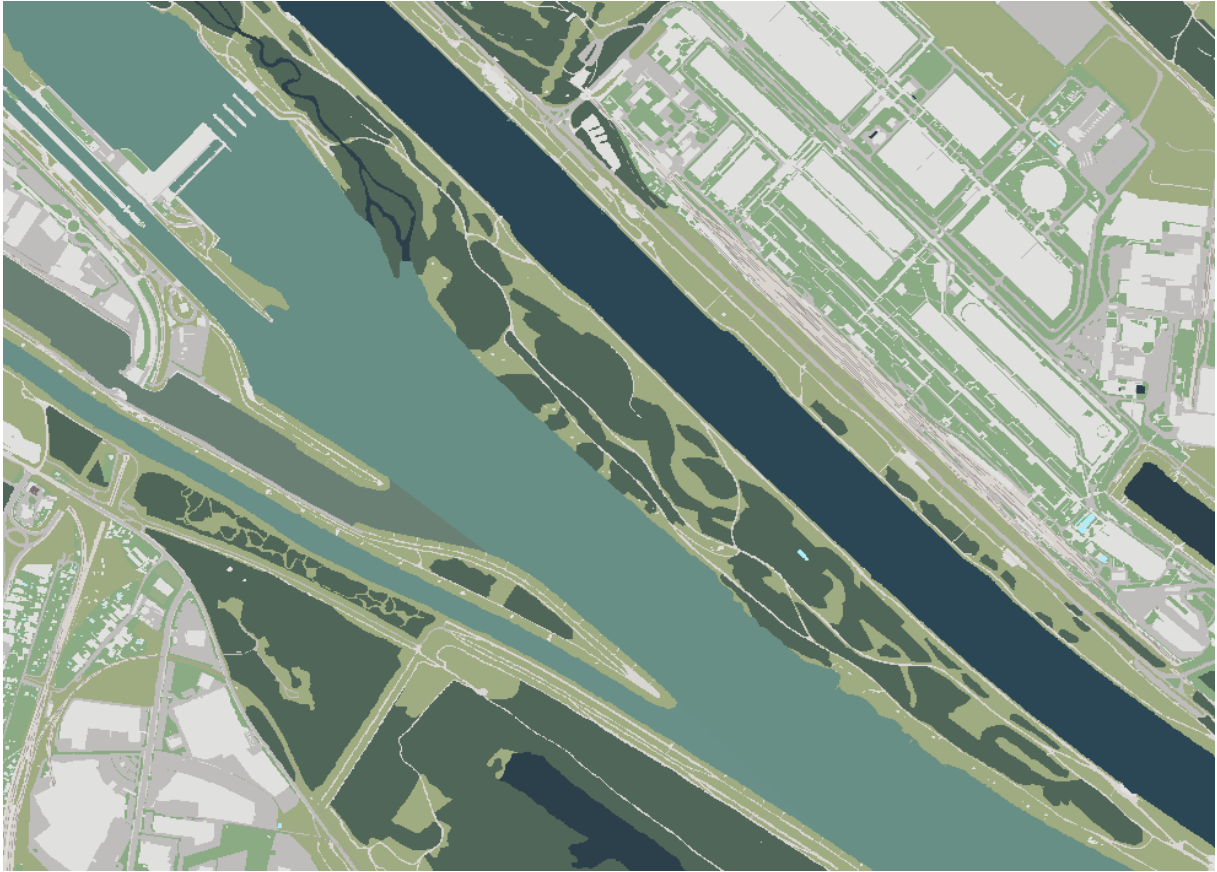


Abbildung 128: Unterschiedliche Farbgebung der Gewässerflächen (hier: Donau und Neue Donau) (Quelle: Stadt Wien intern).

Die in dieser Arbeit neu entworfene Bodenbedeckung unterscheidet sich besonders bei den Gewässerflächen dahingehend von konventionellen Landkarten, da diese hier nicht immer in den sonst üblichen Blautönen eingefärbt sind (siehe [Abbildung 129](#)).



Abbildung 129: Gewässerfläche in nicht-blauer Farbgebung (Quelle: Stadt Wien intern).

Vegetationsflächen

Gerade bei den Vegetationsflächen tritt die Problematik besonders zu Tage, dass sich Flächen mit gleicher Klasse in der Realität deutlich vom Erscheinungsbild her unterscheiden. Schon der Umstand, ob zum Beispiel eine Wiese gemäht wurde, kann einen deutlichen Unterschied in der farblichen Erscheinung bewirken. Besonders die Jahreszeit, in der man Vegetationsflächen betrachtet, spielt bei den Farben eine sehr große Rolle. Man muss sich daher zwangsläufig für eine Jahreszeit entscheiden. Durch diesen Umstand bedingt kann man natürlich auch mehrere Darstellungsarten je nach Jahreszeiten entwerfen.



Abbildung 130: Darstellung der Vegetationsflächen in der Bodenbedeckung (Quelle: Stadt Wien intern).

In *Abbildung 131* treffen zwei verschiedene, mit Gras bewachsene Flächen aufeinander: Wiese und Grünfläche. Man könnte sich nun fragen, warum zwei Flächen, welche beide mit Gras bewachsen sind, so unterschiedlich eingefärbt sind. Diese Farbgebung ist vermutlich der Jahreszeit geschuldet. Im Sommer erscheinen vor allem gemähte Wiesen aufgrund von warmen Temperaturen und teilweise auch aufgrund von Trockenheit in bräunlichen oder olivgrünen Farbtönen. Städtische Grünflächen oder private Rasen-



Abbildung 131: Unterschiedliche Grasflächen in der Bodenbedeckung (Wiese und Grünfläche) (Quelle: Stadt Wien intern).

flächen erscheinen in einem frischeren Grünton, da sie vermutlich mehr gepflegt und daher auch bewässert werden. Hier ist natürlich auch zu beachten, dass sich diese Umstände auch während des Sommers jederzeit ändern können. Wird eine naturnahe Grünfläche zum Beispiel nicht gemäht und regnet es einmal etwas häufiger, so kann diese Fläche auch in einem saftigeren Grün erscheinen. Demgegenüber können auch private Rasenflächen manchmal etwas trockener erscheinen.

9.5.5. Technische Aspekte der Bodenbedeckungsklassen

Um den Bodenbedeckungslayer im 3D-Viewer anzeigen zu können, muss dieser als WMS-Service zur Verfügung stehen. Es bestehen zahlreiche Möglichkeiten, diese Anforderung zu erfüllen. Da die bestehenden Bodenbedeckungslayer des 3D-Stadtmodells bereits in ArcMap erstellt wurden, existiert bereits ein sogenanntes Layer-File, welches das existierende Design für die Bodenbedeckung beinhaltet. Das neu zu entwerfende Design kann nun auf Basis dieses Existierenden Layers aufgebaut werden. Die bestehenden Klassen werden neu zugeordnet und die Farben entsprechend *Tabelle 5* neu vergeben. Das überarbeitete Layer-File kann danach als neuer WMS-Service auf ArcGIS-Server veröffentlicht werden. (vgl. ESRI 2020b)

Abbildung 132 und *Abbildung 133* bieten einen Vergleich zwischen dem existierenden Design für die Bodenbedeckung und dem neu entworfenen Design. In *Abbildung 132* erkennt man deutlich, dass das Design eher dem einer Landkarte gleicht und daher von der Farbgebung nicht viel

Ähnlichkeit mit der Realität aufweist. *Abbildung 133* zeigt das über die Anpassung des Ursprungslayers entstandene neue Bodenbedeckungsdesign.



Abbildung 132: bestehendes Design für die Bodenbedeckung (Quelle: Stadt Wien intern).



Abbildung 133: neu entworfenes Design für die Bodenbedeckung (Quelle: Stadt Wien intern).

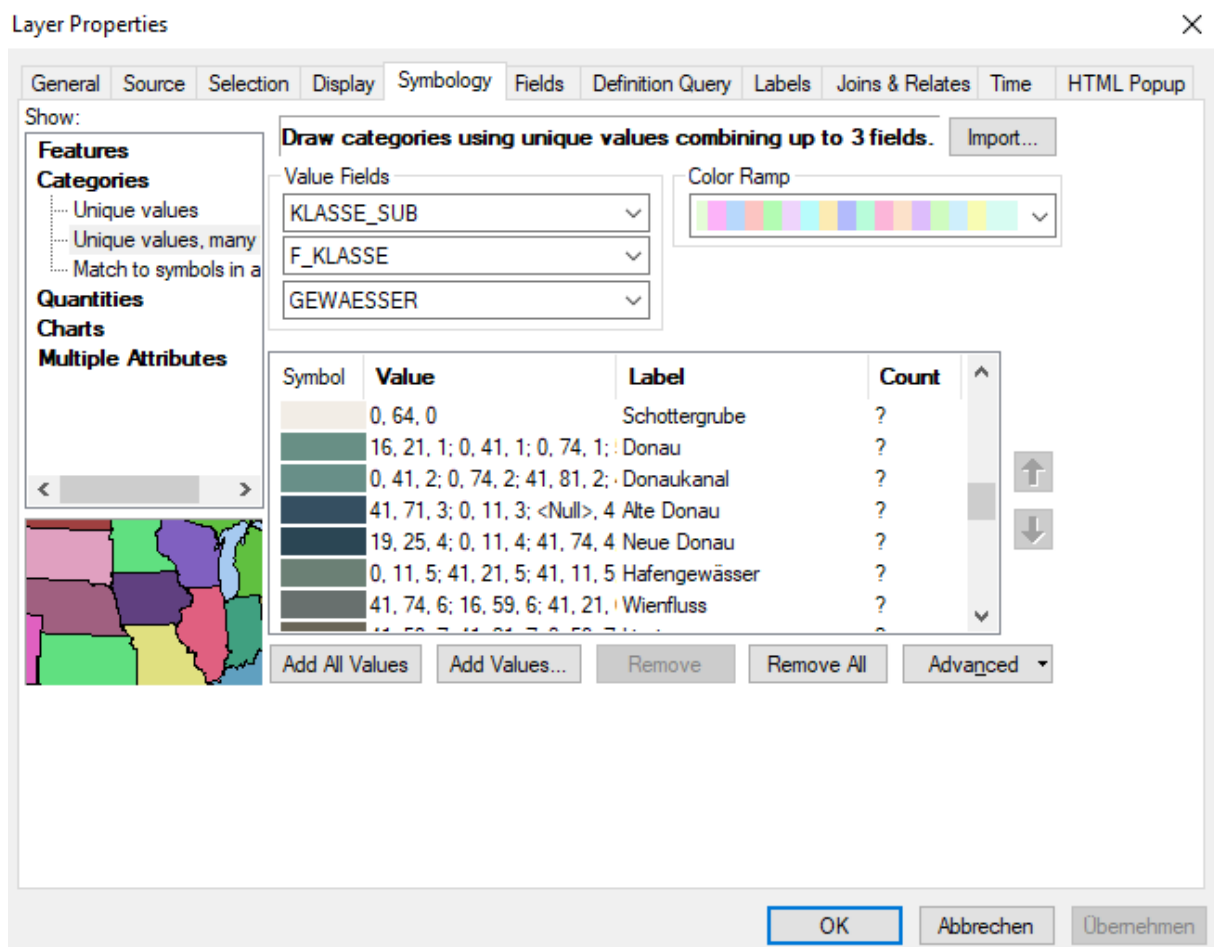


Abbildung 134: Layer Eigenschaften des neuen Bodenbedeckungslayers (Quelle: Stadt Wien intern).

Abbildung 134 zeigt, wie in ArcMap die Kombination der verschiedenen Attribute realisiert wird. Jede Bodenbedeckungsklasse wird über die drei Attribute KLASSE_SUB, F_KLASSE und GEWAESSER bestimmt. Unter *Value* sind jene Werte der Attribute angeführt, welche die jeweilige Klasse kennzeichnen. Diese sind durch ein Komma voneinander getrennt. Da es hier pro Klasse zu mehreren Kombinationen der drei Attribute kommen kann, wurden mehrere Klassen mit verschiedenen Kombinationen zu jeweils einer Klasse gruppiert. Diese gruppierten Unterklassen sind durch eine Trennung der Werte durch ein Semikolon im Bereich *Values* ersichtlich.

10. Fazit und Ausblick

In diesem Kapitel soll noch einmal diese Masterarbeit kurz zusammengefasst werden und die Ergebnisse beziehungsweise mögliche zukünftige Entwicklungsmöglichkeiten diskutiert werden. Zusätzlich sollen hier auch die Forschungs- und Arbeitsfragen mit den während der Erstellung dieser Arbeit erlangten Erkenntnissen beantwortet und aufgetretene Herausforderungen und Probleme diskutiert werden.

10.1. Zusammenfassung

Das Ziel dieser Masterarbeit war es, ein neues Erscheinungsbild für 3D-Stadtmodelle zu entwerfen. Dabei lag der Fokus nicht nur auf einer theoretischen Erarbeitung dieser Thematik, sondern vor allem auch auf dem praktischen Teil, in dem ein solches Erscheinungsbild für die Stadt Wien umgesetzt wurde.

Zu diesem Erscheinungsbild zählt die Visualisierung von LOD2-Gebäuden und die Darstellung der Bodenbedeckung. Weitere Elemente von 3D-Stadtmodellen wurden vernachlässigt, um den Rahmen dieser Arbeit nicht zu sprengen.

Konkret wurde versucht, bestehende Texturen von Gebäuden so weit zu vereinfachen, dass für jede Gebäudefläche nur mehr ein einzelner Farbwert übrigbleibt. Die Farbgebung für die Bodenbedeckung wurde auf Basis von Vektordaten mittels Vergleich mit einem Orthofoto entwickelt.

10.2. Beantwortung der Forschungsfragen

Wie kann man Farbwerte von Dach- und Fassadenflächen aus Texturen ableiten?

Farbwerte können aus Texturen abgeleitet werden, indem man die Menge aller Farbwerte pro Fläche mit statistischen Methoden weiterverarbeitet und dadurch einen einzelnen Farbwert generiert. Das heißt man berechnet zum Beispiel den Mittelwert oder den Modalwert aller Farbwerte, welche sich auf einer Dachfläche oder einer Fassadenfläche befinden. Noch bessere Ergebnisse können mit der Clusteranalyse erzielt werden.

Wie soll die Bodenbedeckung aussehen, damit sie optimal zu den mit Farbwerten versehenen Gebäuden passt?

Auch bei der Bodenbedeckung soll für jede Fläche nur ein einziger Farbwert existieren. Dieser sollte möglichst nahe der mittleren Farbe der Realität entsprechen. Man kann also ähnliche

Methoden wie bei den Gebäudeflächen für die Farbwertextraktion anwenden, oder aber einzelne Farbwerte für Kategorien vergeben.

10.3. Beantwortung der Arbeitsfragen

Welche Software beziehungsweise Programmiersprache eignet sich am besten, um eine automatisierte Extraktion von Farbwerten aus Texturen durchzuführen?

Nach der Evaluierung in Kapitel 7. *Softwarewahl zur Farbwertextraktion aus Gebäudetexturen* erscheint Python als geeignetste Programmiersprache, um Farbwerte aus Texturen zu extrahieren.

Wie lässt sich die Farbwertberechnung optimieren, um die bestmögliche Performanz zu erreichen?

Die Anzahl der Pixel einer Gebäudefläche bestimmt die Performanz maßgeblich. Daher sollte diese Zahl möglichst gering gehalten werden.

Welche Datenquelle eignet sich am besten für die Visualisierung der Bodenbedeckung?

Die Flächen-Mehrzweckkarte (*siehe 9.4.2. Flächen-Mehrzweckkarte (FMZK)*) eignet sich am besten für die Visualisierung der Bodenbedeckung.

Welche Technologie ist am besten für die Integration der Bodenbedeckung in das 3D-Stadtmodell geeignet?

Die Bodenbedeckung wird mittels WMS-Service in das 3D-Stadtmodell integriert.

10.4. Herausforderungen und Probleme

Wie bei jeder herausfordernden Aufgabe traten auch bei der praktischen Durchführung dieser Arbeit einige Herausforderungen und Probleme auf.

Zunächst bestand die Schwierigkeit darin, die vorhandene Dokumentation von CityGML zu „entschlüsseln“, da außer dieser keine praktischen Anwendungsbeispiele im Internet zu finden waren. Dies betraf zunächst die Koordinaten der Texturen und dann auch die genaue Form, in welcher die neuen Farbwerte in einer CityGML-Datei abzuspeichern waren.

Danach bereitete die korrekte Berechnung der Exposition Probleme. Deshalb wurde die Exposition später den CityGML-Dateien entnommen, wo sie verfügbar war.

Besonders herausfordernd war das Ausgleichen der dunklen Flächen in den Schattenseiten gegenüber den Sonnenseiten.

10.5. Ausblick

Wie bereits mehrmals in dieser Arbeit erwähnt, ist das hier entworfene Erscheinungsbild nur ein Teil des Großen und Ganzen. Zunächst könnte man die Farbwertextraktion noch weiter perfektionieren, bis auch die kleinsten Unstimmigkeiten beseitigt sind. Zusätzlich würde zu einem vollständigen Erscheinungsbild noch die Darstellung von Bäumen oder Tunnelportalen gehören. Bestehende Elemente wie Brücken könnten auch noch in ihrem Erscheinungsbild verbessert werden.

Quellenverzeichnis

1 & 1 IONOS SE (2021): Was ist ein Graphical User Interface (GUI)?; <https://www.ionos.at/digitalguide/websites/web-entwicklung/was-ist-ein-gui/> (06.01.2021).

3D geoinformation group (o.J.): CityJSON; <https://www.cityjson.org/> (18.01.2021).

ACADEMIC (o.J.): Kommandozeilenprogramm; <https://deacademic.com/dic.nsf/dewiki/785551> (06.01.2021).

ANDERS T. (2005): Die CIE-Normfarbtafel; <https://de.wikipedia.org/wiki/CIE-NORMVALENZSYSTEM#/Media/Datei:CIE-NORMFARBTAfel.PNG> (20.11.2020).

Aspose Pty Ltd (2021): What is an OBJ File?; <https://docs.fileformat.com/3d/obj/> (18.01.2021).

BAUMANN U. (2011): Farbsysteme in Kunst und Wissenschaft. – Zürich; <http://www.colorsystem.com/> (26.09.2020).

BILLEN R., CUTTING-DECELLE A., MARINA O., ALMEIDA J., CAGLIONI M., FALQUET G., LEDUC T., MÉTRAL C., MOREAU G., PERRET J., RABINO G., SAN JOSE R., YATSKIV I. und ZLATANOVA S. (2014): 3D City Models and urban information: Current issues and perspectives;
DOI: 10.1051/TU0801/201400001.

BILJECKI F. (2013): The concept of level of detail in 3D city models. - PhD Research Proposal; <http://www.gdmc.nl/publications/reports/GIS62.pdf> (10.02.2021).

BILJECKI F. (2017): Level of Detail in 3D City Models. – Dissertation, Delft University of Technology, Delft;
DOI: 10.4233/uuid:f12931b7-5113-47ef-bfd4-688aae3be248.

Blom Norway AS (2021): BLOM NORWAY; <https://blom.no/> (10.02.2021).

Brain4ce Education Solutions Pvt. Ltd. (2021): How to Parse and Modify XML in Python?; <https://www.edureka.co/blog/python-xml-parser-tutorial/#:~:text=the%20upcoming%20examples.,Python%20XML%20Parsing%20Modules,of%20that%20particular%20XML%20file.> (19.01.2021).

Bundesamt für Landestopographie swisstopo (o.J.): 3D-Viewer – SwissTopo; <https://www.swisstopo.admin.ch/de/karten-daten-online/karten-geodaten-online/3d-viewer.html> (09.02.2021).

City of New York (2017): Open Data for All New Yorkers; <https://opendata.cityofnewyork.us/> (09.02.2021).

COORS V., ANDRAE C. und BÖHM K. (2016): 3D-Stadtmodelle. Konzepte und Anwendungen mit CityGML. - Berlin.

Creative Dragon Works (2020): Kleine Fotoschule Teil 1: Kameras & Objektive; <https://creative-dragon-works.de/kleine-fotoschule-1> (26.12.2020).

CT Aeropics (o.J.): Danish Light Aircraft; <https://www.ctaeropics.com/Europe/Denmark/Danish-Light-Aircraft/i-sJsMwLg/> (12.01.2021).

Desmos, Inc. (2020): Desmos; <https://www.desmos.com/?lang=de> (30.12.2020).

DigitalOcean, LLC (2021): How To Write Your First Python 3 Program; <https://www.digitalocean.com/community/tutorials/how-to-write-your-first-python-3-program> (07.01.2021).

ESRI (2020a): Multipatches; <https://desktop.arcgis.com/de/arcmap/latest/extensions/3d-analyst/multipatches.htm> (18.01.2021).

ESRI (2020b): WMS-Services; <https://enterprise.arcgis.com/de/server/latest/publish-services/linux/wms-services.htm> (19.10.2020).

E-teaching.org (o.J.): Dateiformat; <https://www.e-teaching.org/materialien/glossar/dateiformat> (18.01.2021).

Europa Zollportal (2005): Definition: Bodenbedeckung; <https://www.zolltarifnummern.de/info/abkuerzungen/2069> (07.10.2020).

Evermotion (o.J.): Tree 33 AM171 Archmodels; https://evermotion.org/shop/show_product/tree-33-am171-archmodels/13156 (01.02.2021).

F4 (o.J.): F4map; <https://demo.f4map.com/#lat=48.6677832&lon=14.9345006&zoom=19&camera.theta=59.61&camera.phi=63.885> (08.02.2021).

FERMUM L. (o.J.): Farbfilter: Standard-Zubehör für Objektive; <https://www.vision-doctor.com/farbfilter.html> (27.10.2020).

Fuchs Media Solutions (o.J.): Auflösung Begriffserklärung und Definition; <https://www.seo-analyse.com/seo-lexikon/a/aufloesung/> (13.01.2021).

GADE T. (2019): Tutorial: 8-Bit / 16 Bit; <https://www.photoinfos.com/Fotolabor/8-Bit-16-Bit/8-Bit-16-Bit.htm> (02.01.2021).

GeoFly GmbH (o.J.): True Orthofotos; <https://web.geofly.eu/de/produkte/true-orthophoto/true-orthophoto-oberflaechenmodell.html?acceptCookie=1> (12.01.2021).

GLAVENU E. (2020): Paris 3D; <https://www.arcgis.com/home/item.html?id=0614eaf9dd043e9ba157b9c20d3c538> (09.02.2021).

GONZALEZ R. UND WOODS R. (o.J.): Farbmodelle; <https://kik.informatik.fh-dortmund.de/abschluss-arbeiten/chameleon/chapters/basics/farbmodelle.html> (21.11.2020).

Hackr.io (2021): Best Programming Languages to Learn in 2021 (for Job & Future); <https://hackr.io/blog/best-programming-languages-to-learn-2021-jobs-future> (06.01.2021).

HAKE G. und GRÜNREICH D. (1994): Kartographie. – Berlin, New York;
DOI: <https://doi-org.uaccess.univie.ac.at/10.1515/9783110870572>.

HAUPTMANN T. (o.J.): Einführung in die Statistik; <https://www.mathe-online.at/lernpfade/Lernpfad796/?kapitel=2> (05.12.2021).

HÄSSLER U. (2018): Farbe auf Wisotop. – HSB/HSV und HSL-Farbmodell. – Köln;
<https://wisotop.de/hsv-und-hsl-farbmodell.php> (22.10.2020).

Heinze GmbH (o.J.): Farbenlehre: Farbe – Entstehung und Wirkung; <https://www.bauemotion.de/magazin/farbenlehre-farbe-entstehung-und-wirkung/15002724/> (26.10.2020).

HELDENS W., BURMEISTER C., KANANI-SÜHRING F., MARONGA B., PAVLIK D, SÜHRING M, ZEIDLER J und ESCH T. (2020): Geospatial input data for the PALM model system 6.0: model requirements, data sources and processing; <https://gmd.copernicus.org/articles/13/5833/2020/#&gid=1&pid=1> (01.02.2021).

Here (2021): NAVTEQ is HERE; <https://www.here.com/navteq> (10.02.2021).

HESSELING K. (o.J.): Ich sehe was, was du nicht siehst: Metadaten in Fotos; <https://irighits.info/artikel/metadaten-fotos-anbringen-loeschen-bearbeiten/26353> (02.01.2021).

Hochschule Ravensburg-Weingarten University (2011a): Einführung Digitale Spiegelreflexkameras (DSLR) Teil 1 von 3; <https://www.youtube.com/watch?v=jHFXVA5od2g> (12.12.2020).

Hochschule Ravensburg-Weingarten University (2011b): Einführung Digitale Spiegelreflexkameras (DSLR) Teil 2 von 3; <https://www.youtube.com/watch?v=O0JoWnSFPhk> (12.12.2020).

HODAM H. (o.J.a): Räumliche Auflösung; <https://fis.uni-bonn.de/recherchetools/infobox/profis/auf%C3%B6sung/r%C3%A4umliche-auf%C3%B6sung> (13.01.2021).

HODAM H. (o.J.b): Spektrale Auflösung; <https://fis.uni-bonn.de/recherchetools/infobox/profis/auf%C3%B6sung/spektrale-auf%C3%B6sung> (13.01.2021).

HODAM H. (o.J.c): Radiometrische Auflösung; <https://fis.uni-bonn.de/recherchetools/infobox/profis/auf%C3%B6sung/radiometrische-auf%C3%B6sung#:~:text=Die%20radiometrische%20Aufl%C3%B6sung%20ist%20neben,eines%20Bildes%20Grauwerte%20zu%20unterscheiden.> (13.01.2021).

HODAM H. (o.J.d): Zeitliche Auflösung; <https://fis.uni-bonn.de/recherchertools/infobox/pro-fis/aufloesung/zeitliche-aufloesung#:~:text=Den%20zeitlichen%20Abstand%20zwischen%20zwei,Gebietes%20nennt%20man%20zeitliche%20Aufloesung.> (13.01.2021).

IGG Uni Bonn (o.J.): The five levels-of-detail (LOD) defined by CityGML; https://www.researchgate.net/figure/The-five-levels-of-detail-LOD-defined-by-CityGML-IGG-Uni-Bonn_fig31_342315126 (21.01.2021).

Imaging Arts Printing (2015): Why Do My Print Colors Not Match My Monitor?; <https://www.imagingartsprinting.com/2015/01/19/why-do-my-print-colors-not-match-my-monitor/> (26.10.2020).

JESUSSEK M. (2020): Hypothesen und Hypothesentests; https://www.youtube.com/watch?v=BVyAomYugcA&feature=emb_logo (05.12.2021).

KOLBE (2016): 3D City Model of New York City; <https://www.lrg.tum.de/gis/projekte/new-york-city-3d/> (09.02.2021).

KÖNIG D. (2010): Vollerhebung oder Teilerhebung (Stichprobe)?; <http://www.marketingkoenig.de/vollerhebung-oder-teilerhebung-stichprobe/> (05.01.2021).

KÖSTNER J. (o.J.): Farben; https://www.die-judith.at/sites/design_farben.html (07.11.2020).

LAKE R. (o.J.): Introduction to GML; <https://www.w3.org/Mobile/posdep/GMLIntroduction.html> (19.01.2021).

Land Tirol (o.J.): Orthofotos; <https://www.tirol.gv.at/sicherheit/geoinformation/geodaten/orthofotos/> (13.01.2021).

LEHNER H. und DORFFNER L. (2020): Digital geoTwin Vienna: Towards a Digital Twin City as Geodata Hub; DOI: <https://doi.org/10.1007/s41064-020-00101-4>.

LIEBMANN C. und WELSCH N. (2018): Farben. Natur, Technik, Kunst. – Berlin, Heidelberg.

LÖWNER M.-O., CASPER E., BECKER T., BRENNER J., GRÖGER G., GRUBER U., HÄFELE K.-H., KADEN R. und SCHLÜTER S. (2012): CityGML 2.0. Ein internationaler Standard für 3D-Stadtmodelle Teil 2: CityGML in der Praxis. – In: Zeitschrift für Geodäsie, Geoinformation und Landmanagement 138 (2), 131-143. – Braunschweig u.a.; auch online unter: https://geodaesie.info/sites/default/files/privat/zfv_2013_2_Loewner_et-al_Teil-2.pdf (10.02.2020).

LÜKEN J. und SCHIMMELPFENNIG H. (2017): Partitionierende Clusteranalyse;

<https://www.ifad.de/so-funktioniert-die-partitionierende-clusteranalyse/> (06.01.2021).

MAIER M. (2015): 8-Bit, 16-Bit oder 32-Bit Bilder; <https://urban-base.net/2015/03/8-bit-16-bit-oder-32-bit-bilder/> (02.01.2021).

Max-Planck-Gesellschaft (2020): Genauere Betrachtung der einzelnen Wellenlängenbereiche; <https://www.mpifr-bonn.mpg.de/563197/einteilung> (18.10.2020).

Medium (2020): Top 10 In-Demand programming languages to learn in 2020; <https://towards-datascience.com/top-10-in-demand-programming-languages-to-learn-in-2020-4462eb7d8d3e> (06.01.2021).

Mentorium GmbH (2021): Deskriptive Statistik; <https://www.statistik-nachhilfe.de/ratgeber/statistik/deskriptive-statistik#:~:text=Die%20beschreibende%20oder%20deskriptive%20Statistik,wenn%20es%20um%20Datenanalyse%20geht.&text=Es%20geht%20Dir%20bei%20der,diesen%20konkreten%20Datensatz%20zu%20machen.> (05.01.2021).

Mind Kommunikation GmbH (o.J.): Dateiformate für Bilder und ihre Eigenschaften; <https://www.mind.ch/dateiformate-fuer-bilder-und-ihre-eigenschaften/> (12.12.2020).

MyCurveFit (2020): MyCurveFit; <https://mycurvefit.com/> (30.12.2020).

Northeastern University (2019): The 10 Most Popular Programming Languages To Learn In 2020; <https://www.northeastern.edu/graduate/blog/most-popular-programming-languages/> (06.01.2021).

Novustat (2021): Deskriptive Statistik; <https://novustat.com/statistik-glossar/deskriptive-statistik.html#:~:text=Die%20deskriptive%20Statistik%20verwendet%20Kennzahlen,oder%20Grafiken%2C%20oder%20mehrere%20davon.> (05.01.2021).

OERTHEL F. und TUSCHL S. (1995): Statistische Datenanalyse mit dem Programmpaket SAS. - München.

Open Geospatial Consortium (2012): OGC City Geography Markup Language (CityGML) Encoding Standard; https://portal.opengeospatial.org/files/?artifact_id=47842 (19.01.2021).

Open Geospatial Consortium (o.J.): Model of IndoorGML schema; https://www.researchgate.net/figure/Model-of-IndoorGML-schema-Figure-courtesy-of-OGC_fig10_273203404 (23.01.2021).

OpenStreetMap (o.J.): OpenStreetMap; <https://www.openstreetmap.org> (07.10.2020).

ÖTK Österreichischer Touristenklub (2018): Digitale Landkarten; <https://www.oetk.at/wege/digitale-karten/> (13.01.2021).

Parewa Labs Pvt. Ltd. (o.J.a): C „Hello, World!“ Program; <https://www.programiz.com/c-programming/examples/print-sentence> (07.01.2021).

Parewa Labs Pvt. Ltd. (o.J.b): C Programming; <https://www.programiz.com/c-programming/guide> (07.01.2021).

PCC Gruppe (2019): Wie entstehen Farben und wieso sehen wir sie?. Farbmodelle und ihre Beschreibung; <https://www.products.pcc.eu/de/blog/wie-entstehen-farben-und-wieso-sehen-wir-sie-farbmodelle-und-ihre-beschreibung/> (26.10.2020).

PETRI B. und PETRI B. (2021): Hello World!; <https://www.java-tutorial.org/hello-world.html> (07.01.2021).

PFUND A. (o.J.a): Metadaten; <http://andreas-pfund.de/definition/metadaten/metadaten.php> (02.01.2021).

PIETRZYK J. (2014): Histogramme und Punktoperatoren; http://www.pub.zih.tu-dresden.de/~photo/teachlets/Histogramme_Punktoperatoren/index.html (14.01.2021).

PRATZNER A. (o.J.a): ISO-Empfindlichkeit: Lichtempfindlichkeit des Bildsensors; <https://www.foto-kurs.com/iso-empfindlichkeit.htm> (01.01.2021).

PRATZNER A. (o.J.b): Weißabgleich – White Balance (WB); <https://www.foto-kurs.com/weissabgleich-white-balance-wb.php> (01.01.2021).

Prefa (2021): Tipps für die Wahl der richtigen Dacheindeckung für ihr neues Dach; <https://www.prefa.at/bautipps/hochwertige-dacheindeckung-fuer-jede-dachform/> (22.01.2021).

Pxhere (o.J.a): Sonnenlicht; <https://pxhere.com/de/photo/833927> (27.10.2020).

Pxhere (o.J.b): Kerzenlicht; <https://pxhere.com/de/photo/561043> (27.10.2020).

QGIS (o.J.): Räumliche Analyse (Interpolation); https://docs.qgis.org/2.18/de/docs/gentle_gis_introduction/spatial_analysis_interpolation.html (20.01.2021).

QUARK67 (2006): Additive Farbmischung; https://de.wikipedia.org/wiki/Additive_Farbmi-schung#/media/Datei:Synthese+.svg (21.11.2020).

Reese Online e.K. (2021): Visualisierung; <https://www.fremdwort.de/suchen/bedeutung/visualisierung#> (15.01.2021).

Refsnes Data (2021a): Introduction to XML; https://www.w3schools.com/xml/xml_wha-tis.asp (19.01.2021).

Refsnes Data (2021b): How Can XML be Used?; https://www.w3schools.com/xml/xml_usedfor.asp (19.01.2021).

Refsnes Data (2021c): XML Tree; https://www.w3schools.com/xml/xml_tree.asp (19.01.2021).

Refsnes Data (2021d): XML Syntax Rules; https://www.w3schools.com/xml/xml_syntax.asp (19.01.2021).

Refsnes Data (2021e): XML Attrivutes; https://www.w3schools.com/xml/xml_attributes.asp (19.01.2021).

Refsnes Data (2021f): XML Parser; https://www.w3schools.com/xml/xml_parser.asp (19.01.2021).

Refsnes Data (2021g): XML Schema Tutorial; https://www.w3schools.com/xml/schema_intro.asp (19.01.2021).

Refsnes Data (2021h): XSD How To?; https://www.w3schools.com/xml/schema_howto.asp (19.01.2021).

Refsnes Data (2021i): XSD – The <schema> Element; https://www.w3schools.com/xml/schema_howto.asp (19.01.2021).

Refsnes Data (2021j): Python Intro; https://www.w3schools.com/python/python_intro.asp (07.01.2021).

Refsnes Data (2021k): Python Data Types; https://www.w3schools.com/python/python_da-tatypes.asp (07.01.2021).

Refsnes Data (2021l): Java Intro; https://www.w3schools.com/java/java_intro.asp (07.01.2021).

Refsnes Data (2021m): Java Data Types; [https://www.w3schools.com/java/java_data ty-pes.asp](https://www.w3schools.com/java/java_data_ty-pes.asp) (07.01.2021).

Refsnes Data (2021n): Python Classes and Objects; https://www.w3schools.com/python/py-thon_classes.asp (07.01.2021).

Refsnes Data (2021o): Java OOP; https://www.w3schools.com/java/java_oop.asp (07.01.2021).

SACH W. (2018): GEORES; <https://docplayer.org/59122411-Produktdokumentation.html> (01.02.2021).

Safe Software Inc (2021): FME; <https://www.safe.com/> (04.02.2021).

SCHÄFER T. (2016): Methodenlehre und Statistik. - Einführung in Datenerhebung, deskriptive Statistik und Inferenzstatistik. - Wiesbaden.

SCHÜNGEL F. (1999): Licht und Farbe; <http://www.foto-net.de/net/licht/licht.html> (26.10.2020).

Sorembâ GmbH (2020): CMYK – eine kurze Erklärung; <https://www.soremba.eu/glossar/cmyk> (22.11.2020).

Spektrum der Wissenschaft Verlagsgesellschaft mbH (1999): Gewässeroptik; <https://www.spektrum.de/lexikon/optik/gewaesseroptik/1159> (14.10.2020).

Spektrum der Wissenschaft Verlagsgesellschaft mbH (2001a): Bildflug; <https://www.spektrum.de/lexikon/kartographie-geomatik/bildflug/529> (12.01.2021).

Spektrum der Wissenschaft Verlagsgesellschaft mbH (2001b): Bildflugzeug; <https://www.spektrum.de/lexikon/kartographie-geomatik/bildflugzeug/531> (12.01.2021).

Spektrum der Wissenschaft Verlagsgesellschaft mbH (2001c): Bildflugplanung; <https://www.spektrum.de/lexikon/kartographie-geomatik/bildflugplanung/530> (12.01.2021).

Spektrum der Wissenschaft Verlagsgesellschaft mbH (2001d): Bildüberdeckung; <https://www.spektrum.de/lexikon/kartographie-geomatik/bildueberdeckung/560> (12.01.2021).

Spektrum der Wissenschaft Verlagsgesellschaft mbH (2001e): innere Orientierung; <https://www.spektrum.de/lexikon/kartographie-geomatik/innere-orientierung/2377> (13.01.2021).

Spektrum der Wissenschaft Verlagsgesellschaft mbH (2001f): äußere Orientierung; <https://www.spektrum.de/lexikon/kartographie-geomatik/aeussere-orientierung/339> (13.01.2021).

Spektrum der Wissenschaft Verlagsgesellschaft mbH (2001g): Orthofoto; <https://www.spektrum.de/lexikon/kartographie-geomatik/orthophoto/3776> (13.01.2021).

Spektrum der Wissenschaft Verlagsgesellschaft mbH (2001h): Kartographische Generalisierung; <https://www.spektrum.de/lexikon/kartographie-geomatik/kartographische-generalisierung/2714> (16.03.2021).

Stadt Leipzig (2021a): 3D Stadtmodell – Stadt Leipzig; https://static.leipzig.de/fileadmin/mediendatenbank/leipzig-de/Stadt/02.6_Dez6_Stadtentwicklung_Bau/62_Amt_fuer_Geoinformationen_und_Bodenordnung/Kartographie/LOD1.PNG (23.01.2021).

Stadt Leipzig (2021b): 3D Stadtmodell – Stadt Leipzig; https://static.leipzig.de/fileadmin/mediendatenbank/leipzig-de/Stadt/02.6_Dez6_Stadtentwicklung_Bau/62_Amt_fuer_Geoinformationen_und_Bodenordnung/Kartographie/LOD2.PNG (23.01.2021).

Stadt Leipzig (2021c): 3D Stadtmodell – Stadt Leipzig; https://static.leipzig.de/fileadmin/mediendatenbank/leipzig-de/Stadt/02.6_Dez6_Stadtentwicklung_Bau/62_Amt_fuer_Geoinformationen_und_Bodenordnung/Kartographie/Alte_Hauptpost_3D.jpg (23.01.2021).

Stadt Posen (o.J.): Poznan – Model 3D;
<http://sip.poznan.pl/model3d/#/> (27.02.2020).

Stadt Wien (2020): Flächen-Mehrzweckkarte (FMZK); <https://www.wien.gv.at/stadtentwicklung/stadtvermessung/pdf/fmzk-attribute-wertebereiche.pdf> (08.10.2020).

Stadt Wien (o.J.a): Stadtplan 3D;
<https://www.wien.gv.at/stadtplan3d/#/> (19.02.2020).

Stadt Wien (o.J.b): Baukörpermodell; <https://www.wien.gv.at/stadtentwicklung/stadtvermessung/geodaten/bkm/> (04.02.2021).

Stadt Wien (o.J.c): Flächen-Mehrzweckkarte; <https://www.wien.gv.at/stadtentwicklung/stadtvermessung/geodaten/fmzk/> (08.10.2020).

Stadt Wien (o.J.d): Mehrzweckkarte; <https://www.wien.gv.at/stadtentwicklung/stadtvermessung/geodaten/mzk/> (08.10.2020).

Stadt Wien (o.J.e): Geodatenviewer der Stadtvermessung Wien;
<https://www.wien.gv.at/ma41datenviewer/public/start.aspx> (02.11.2020).

Stadt Wien (o.J.f): Dachmodell; <https://www.wien.gv.at/stadtentwicklung/stadtvermessung/geodaten/dachmodell/> (04.02.2021).

Stadt Wien (o.J.g): Dreidimensionales Stadtmodell; <https://www.wien.gv.at/stadtentwicklung/stadtvermessung/geodaten/stadtmodell/> (13.04.2021).

Stadt Wien (o.J.h): Geländemodell; <https://www.wien.gv.at/stadtentwicklung/stadtvermessung/geodaten/dgm/> (04.02.2021).

Stadt Wien und Österreichische Länder (2016): basemap.at; <https://basemap.at/> (04.02.2021).

Statista GmbH (o.J.): Definition Statistik für Anfänger – Einführung in die Statistik;
https://de.statista.com/statistik/lexikon/definition/11/statistik_fuer_anfaenger_einfuehrung_in_die_statistik/ (05.01.2021).

STOCKER T. UND STEINKE I. (2017): Statistik. - Grundlagen und Methodik. - Berlin, Boston.

Sun.org GmbH (o.J.): Elektromagnetisches Spektrum; <http://www.sun.org/de/encyclopedia/electromagnetic-spectrum> (11.10.2020).

Südwestrundfunk (2020): Wie funktioniert ein Spiegel?; <https://www.swr.de/wissen/1000-antworten/wissenschaft-und-forschung/1000-antworten-1444.html> (26.10.2020).

Technisches Zeichnen (2008): perspektivische Darstellungstechnik – Zentralprojektion; <https://www.technisches-zeichnen.net/technisches-zeichnen/grundkurs-01/zentralprojektion.php> (13.01.2021).

Tech Target (2021): texture mapping; <https://whatis.techtarget.com/definition/texture-mapping> (20.01.2021).

The Khronos Group Inc. (2021): COLLADA; <https://www.khronos.org/collada/> (18.01.2021).

Time and Date AS (o.J.): Sonnenzeit: Was und wann ist Mittag?; <https://www.timeand-date.de/astronomie/mittag> (03.11.2020).

TIOBE Software BV (2020): TIOBE Index for January 2021. - January Headline: Python is TIOBE's Programming Language of 2020!; <https://www.tiobe.com/tiobe-index/> (06.01.2021).

Trimble Inc. (2021): DTMaster; http://trl.trimble.com/docushare/dsweb/Get/Document-696436/022516-019D_Inpho_DTMaster_TS_USL_0516_LR.pdf (07.01.2021).

Tutorialspoint (2020a): C Language - Overview; https://www.tutorialspoint.com/cprogramming/c_overview.htm (07.01.2021).

Tutorialspoint (2020b): C – Data Types; https://www.tutorialspoint.com/cprogramming/c_data_types.htm (07.01.2021).

USGS (o.J.): Earth Explorer; <https://earthexplorer.usgs.gov/> (20.01.2021).

UVM Systems (o.J.a): Homepage von UVM Systems; <https://www.uvmsystems.com/index.php/de-de/> (27.02.2020).

UVM Systems (o.J.b): 3D Stadtmodelle; <https://www.uvmsystems.com/index.php/de-de/service/serv-city#a4> (02.11.2020).

UVM Systems (o.J.c): City Grid; <https://www.uvmsystems.com/index.php/de-de/> (04.02.2021).

UVM Systems (o.J.d): Wiener Linien; <https://www.uvmsystems.com/index.php/de-de/34-de/content-de/content-de-proj/content-de-proj-city/386-art11> (04.02.2021).

ViewApp GmbH (o.J.): FotoFassade.- Hochqualitative 3D-Gebäudemodelle mit Wiedererkennungswert; <https://viewapp.com/aktuelle-entwicklungen> (11.02.2021).

Virtual City Systems (o.J.a): Homepage von Virtual City Systems; <https://vc.systems/> (04.02.2021).

Virtual City Systems (o.J.b): Building Reconstruction; <https://vc.systems/produkte/building-reconstruction/> (04.02.2021).

Virtual City Systems (o.J.c): Berlin 3D; <https://berlin.virtualcitymap.de/#/legend> (08.02.2021).

Virtual City Systems (o.J.d): Bremen 3D; <https://bremen.virtualcitymap.de/#/> (09.02.2021).

Vogel IT-Medien GmbH (2021a): Was ist ein Standard?; <https://www.it-business.de/was-ist-ein-standard-a-919780/> (18.01.2021).

Vogel IT-Medien GmbH (2021b): Was ist eine Programmiersprache?; <https://www.dev-insider.de/was-ist-eine-programmiersprache-a-577066/> (06.01.2021).

WAGNER P. (o.J.): Tonwerte, Histogramm und Tonwertkurve; <https://www.scandig.eu/TonwertHistogramm.html> (02.01.2021).

WÄGER M. (2017): Das ABC der Farbe. – Bonn.

WECKNER A. (o.J.): Tutorial Farbenlehre; http://face2image.de/Tutorial_Farbenlehre.html (26.10.2020).

WEKA MEDIA GmbH & Co. KG (o.J.): Was ist Anwendungssoftware; <https://www.foerderland.de/technik/anwendungssoftware/> (06.01.2021).

X-Rite (2020): Please Explain the Term „Black body curve“; <https://www.xrite.com/service-support/please-explain-the-term-black-body-curve> (20.11.2020).

ZLATANOVA S., STOTER J. und ISIKDAG U. (2012): Standards for Exchange and Storage of 3D Information: Challenges and Opportunities for Emergency Response. - Delft; http://zlatanova.xyz/pdfs/Standards_Exchange_Storage_3D_Information.pdf (18.01.2021).

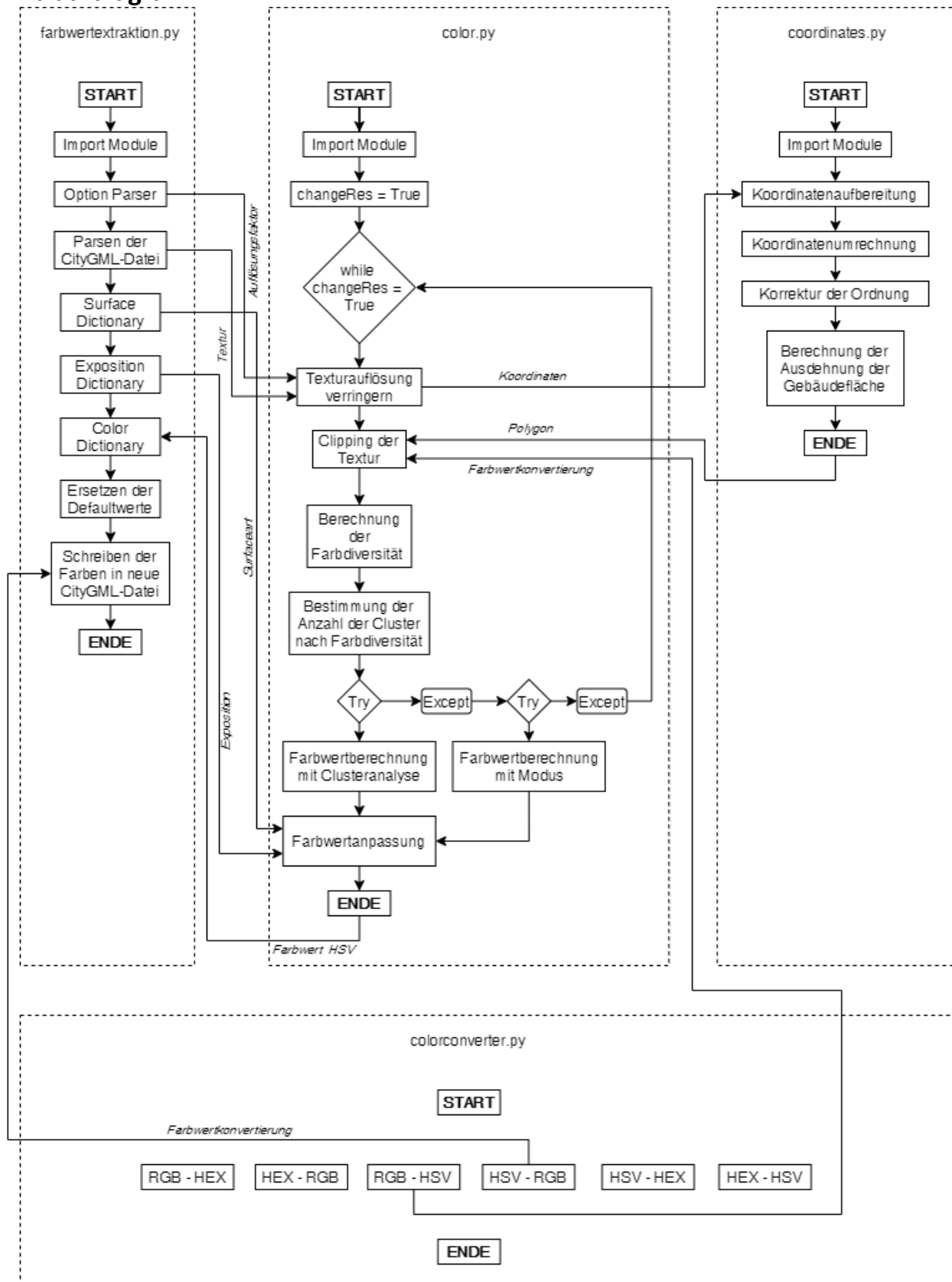
ZOETHESWAGRID (2015): Histogramme; <https://zoecelinebell.wordpress.com/2015/04/19/histogramme/> (02.01.2021).

Ich habe mich bemüht, sämtliche Inhaber der Bildrechte ausfindig zu machen und ihre Zustimmung zur Verwendung der Bilder in dieser Arbeit eingeholt. Sollte dennoch eine Urheberrechtsverletzung bekannt werden, ersuche ich um Meldung bei mir.

Hinweis: Inhalte, welche nur im internen System der Stadt Wien verfügbar sind und somit nicht öffentlich mit einem Link aufrufbar sind, wurden in dieser Arbeit mit „Stadt Wien intern“ zitiert.

Anhang

Ablaufdiagramm



Umrechnung zwischen Farbsystemen (*colorconverter.py*)

```
#####

####---Quellen:---####
# PIL ImageColor Library: https://pillow.readthedocs.io/en/stable/refer-
# Colorsys - Conversion between color systems:https://docs.python.org/3/library/co-
# Konvertierung von RGB in HEX: https://stackoverflow.com/questions/3380726/conver-
# Konvertierung von RGB in HEX: https://stackoverflow.com/questions/3380726/conver-
# Konvertierung von RGB in HEX: https://stackoverflow.com/questions/3380726/conver-
#####

####---Erklärung der verwendeten Variablen:---####
# r: Rot.
# g: Grün.
# b: Blau.
# hexcolor: RGB-Farbe im Hexadezimalformat.
# color: Hilfsvariable.
# color2: Hilfsvariable.
# h: hue (Farbton).
# s: saturation (Sättigung).
# v: value (Helligkeit).
#####

# Import der benötigten Libraries:
import colorsys
from PIL import ImageColor
# -----

# Konvertierung von RGB nach HEX:
def rgb2hex(r, g, b):
    return '#{0:02x}{1:02x}{2:02x}'.format(r, g, b)
# -----

# Konvertierung von HEX nach RGB:
def hex2rgb(hexcolor):
    return ImageColor.getcolor(hexcolor, "RGB")
# -----

# Konvertierung von RGB nach HSV:
def rgb2hsv(r, g, b):
    color = colorsys.rgb_to_hsv(r/255, g/255, b/255)
    color2 = round(color[0] * 360), round(color[1] * 100), round(color[2] * 100)
    return color2
# -----

# Konvertierung von HSV nach RGB:
def hsv2rgb(h, s, v):
    return tuple(round(i * 255) for i in colorsys.hsv_to_rgb(h/360, s/100,
v/100))
# -----

# Konvertierung von HSV nach HEX:
def hsv2hex (h, s, v):
    return rgb2hex(hsv2rgb (h, s, v)[0], hsv2rgb (h, s, v)[1], hsv2rgb (h, s,
v)[2])
# -----

# Konvertierung von HEX nach HSV:
def hex2hsv(hexcolor):
    return rgb2hsv(hex2rgb(hexcolor)[0], hex2rgb(hexcolor)[1], hex2rgb(hexco-
lor)[2])
# -----
#####

#####
```

Umrechnung der Koordinaten (*coordinates.py*)

```
#####

####---Quellen:---####
# Matplotlib.path: https://matplotlib.org/3.3.3/api/path_api.html
# Numpy: https://numpy.org/
# W3schools Numpy: https://www.w3schools.com/python/numpy_intro.asp
#####

####---Erklaerung der verwendeten Variablen:---####
# xmax: maximale Ausdehnung der Texturdatei in x-Richtung.
# ymax: maximale Ausdehnung der Texturdatei in y-Richtung.
# koordinatelist: Liste mit "Koordinatenstrings" (jeder Koordinatenstring beinhaltet die Koordinaten eines Polygons als String).
# polygons: fertige Polygone fuer Point-in-Polygon Test; werden aus den Koordinatenstrings erzeugt.
# coord_split_string: in Einzelkoordinaten aufgesplittete Strings.
# coords: Einzelkoordinaten fertig fuer die Umrechnung.
# coordlist_reversed: Koordinatenliste, in der x- und y-Koordinaten vertauscht sind.
# coordlist_indexlist: Liste zur Speicherung der Indizes, um die Koordinaten spaeter richtig ordnen zu koennen.
# coordlist: fertig umgerechnete Koordinaten als Liste.
# count: Zaehlvariable, um herauszufinden, ob eine Koordinate an gerader oder ungerader Stelle liegt.
# coordlist_index_odd: ungerader Index.
# coordlist_index_even: gerader Index.
# coordarray: Array mit Koordinaten, welches aus der Liste "coordlist" erzeugt wird.
# xMaxPoly: maximale Ausdehnung in x-Richtung.
# yMaxPoly: maximale Ausdehnung in y-Richtung.
# xMinPoly: minimale Ausdehnung in x-Richtung.
# yMinPoly: minimale Ausdehnung in y-Richtung.
#####

# Import der benoetigten Librarys:
import numpy as np
import matplotlib.path as mplPath
#-----

#BEGINN# Funktion zur Koordinatenumwandlung #BEGINN#
def convert(xmax, ymax, koordinatelist):

    polygons = []

    for coordinates in koordinatelist:

        # Aufbereitung:
        coord_split_string = coordinates.split(" ")

        coords = []

        for css in coord_split_string:
            coords.append(float(css))
        #-----

        coordlist_reversed = []
        coordlist_indexlist = []
        coordlist = []

        count = 1

        coordlist_index_odd = 1
        coordlist_index_even = 0
```

```

# Umrechnung:
for c in coords:
    if count % 2 == 0:
        x = c * xmax
        coordlist_reversed.append(x)
        coordlist_indexlist.append(coordlist_index_even)
        coordlist_index_even += 2
    else:
        y = c * ymax
        coordlist_reversed.append(y)
        coordlist_indexlist.append(coordlist_index_odd)
        coordlist_index_odd += 2

    count += 1

#-----

# Ordnung:
for cr, ci in zip(coordlist_reversed, coordlist_indexlist):
    coordlist.insert(int(ci), int(cr))
#-----

coordarray = np.array(coordlist)
coordarray = coordarray.reshape(-1, 2)
xMaxPoly = round(max(x[0] for x in coordarray))
yMaxPoly = round(max(y[1] for y in coordarray))
xMinPoly = round(min(x[0] for x in coordarray))
yMinPoly = round(min(y[1] for y in coordarray))
polygons.append(mplPath.Path(coordarray))

return polygons, xMaxPoly, yMaxPoly, xMinPoly, yMinPoly

#ENDE# Funktion zur Koordinatenumwandlung #ENDE#
#-----
#####
#####

```

Farbwertberechnung (*color.py*)

```

#####

####---Quellen:---####
# Collections Library: https://docs.python.org/3/library/collections.html#collections.Counter
# Mathplotlib.path: https://matplotlib.org/3.3.3/api/path\_api.html
# Numpy: https://numpy.org/
# W3schools Numpy: https://www.w3schools.com/python/numpy\_intro.asp
# Python Imaging Library Pillow: https://pillow.readthedocs.io/en/stable/
# Statistics Library: https://docs.python.org/3/library/statistics.html
# Clusterberechnung mit sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
# Ermitteln der Texturgroesse: https://pillow.readthedocs.io/en/stable/handbook/concepts.html?highlight=size#size
# Ermitteln der Texturgroesse: https://stackoverflow.com/questions/6444548/how-do-i-get-the-picture-size-with-pil
# Veraendern der Aufloesung (Texturgroesse): https://www.geeksforgeeks.org/python-pil-image-resize-method/
#####

####---Erklaerung der verwendeten Variablen:---####
# inputResolution: Wert, durch den die originale Aufloesung der Textur dividiert wird, um die Aufloesung zu verringern.
# texture: originale Texturdatei.
# textureCoordinatesList: Liste mit "Koordinatenstrings" (jeder Koordinatenstring beinhaltet die Koordinaten eines Polygons als String).
# surface: 0 = ground, 1 = wall, 2 = roof

```

```

# exposition: Exposition in Grad.
# changeRes: Variable fuer while-Schleife als Bedingung, ob Aufloesung geaendert
werden soll.
# pixelList: Liste mit Pixeln pro Flaeche.
# diverseDict: beinhaltet die Information, welche Farbklassen in einer Gebaeude-
flaeche vorkommen.
# workingTexture: Kopie der originalen Textur, um diese nicht zu veraendern.
# textureSize: Groesse der Textur in Pixel.
# xmax: maximale Ausdehnung der Texturdatei in x-Richtung.
# ymax: maximale Ausdehnung der Texturdatei in y-Richtung.
# polygons: fertige Polygone fuer Point-in-Polygon Test; werden aus den Koordina-
tenstrings erzeugt.
# xMaxPoly: maximale Ausdehnung in x-Richtung.
# yMaxPoly: maximale Ausdehnung in y-Richtung.
# xMinPoly: minimale Ausdehnung in x-Richtung.
# yMinPoly: minimale Ausdehnung in y-Richtung.
# pointInPolygonList: Liste mit Wahrheitswerten, ob ein Pixel innerhalb eines Poly-
gons liegt (Liste deshalb, weil es durch innere Ringe auch mehrere Polygone pro Ge-
baeudeflaeche geben kann).
# pixelRGB: Pixel im RGB-Format.
# pixelHSV: Pixel im HSV-Format.
# colors: pixelList als Array.
# diversity: Anzahl an Farbklassen pro Gebaeude.
# k: Anzahl der Cluster.
# defaultVar: ist diese 1, so handelt es sich bei einem Farbwert um einen Default-
wert.
# kmeans: Ergebnis der Clusterberechnung.
# centers: Liste der Clustermittelpunkte (Farbwerte).
# n_kmeans: Liste, welche angibt, zu welchem Cluster welches Element gehoert. Jede
Zahl ist dabei ein Index fuer die Position des Elements in der Liste "centers".
# clusterColor: Wert des Mittelpunktes des groessten Clusters.
#####

# Import der benoetigten Librarys:
from collections import Counter
import colorconverter
import coordinates
import matplotlib.path as mplPath
import numpy as np
from PIL import Image, ImageColor
from statistics import mode, mean
from sklearn.cluster import KMeans
#-----

#BEGINN# Funktion zur Farbwertberechnung mittels Clusteranalyse #BEGINN#
def extract(inputResolution, texture, textureCoordinatesList, surface, exposition):

    changeRes = True #####

    #BEGINN# while-Schleife, damit Aufloesung so lange erhoeht wird, bis ein Farb-
wert berechnet werden kann #BEGINN#
    while changeRes == True:

        # Erstellen einer Liste, welche spaeter mit jenen Pixeln befuellt wird,
welche innerhalb des aeusseren Polygons, welches jeweils eine Gebaeudeflaeche be-
grenzt, liegen:
        pixelList = [] ####

#.....

        diverseDict = {1: False, 2: False, 3: False, 4: False, 5: False, 6: False,
7: False, 8: False, 9: False, 10: False, 11: False, 12: False,
13: False, 14: False, 15: False, 16: False, 17: False, 18:
False, 19: False, 20: False, 21: False, 22: False, 23: False, 24: False}

#.....

        # Aufbereitung der Texturdatei; Berechnung der Ausdehnungen; Berechnung des
Polygons, welches eine Gebaeudeflaeche abgrenzt:

```

```

workingTexture = texture
textureSize = workingTexture.size

# Mehrere Versuche fuer resize, falls Ausgangsdatei aufgrund der geringen
Groesse nicht durch die geforderte Zahl dividiert werden kann:
try:
    workingTexture = workingTexture.resize((int(textureSize[0]/inputResolu-
tion), int(textureSize[1]/inputResolution)), Image.ANTIALIAS)
except:
    try:
        workingTexture = workingTexture.resize((int(textureSize[0]/(in-
putResolution / 2)), int(textureSize[1]/(inputResolution / 2))), Image.ANTIALIAS)
    except:
        workingTexture = workingTexture.resize((int(textureSize[0]/(in-
putResolution / 4)), int(textureSize[1]/(inputResolution / 4))), Image.ANTIALIAS)

#.....

# Abfrage der Ausdehnung der Texturdatei:
textureSize = workingTexture.size
xmax = textureSize[0]
ymax = textureSize[1]

#.....

# Berechnung der Polygone und dessen Ausdehnungen:
polygons = coordinates.convert(xmax, ymax, textureCoordinatesList)[0]
xMaxPoly = coordinates.convert(xmax, ymax, textureCoordinatesList)[1]
yMaxPoly = coordinates.convert(xmax, ymax, textureCoordinatesList)[2]
xMinPoly = coordinates.convert(xmax, ymax, textureCoordinatesList)[3]
yMinPoly = coordinates.convert(xmax, ymax, textureCoordinatesList)[4]

#.....
#-----

#BEGINN# Befuellen von "pixelList", falls Pixel in Polygon liegt (ent-
spricht einem Clipping der Textur durch ein Polygon) #BEGINN#
for x in range(xMinPoly, xMaxPoly):
    for y in range(yMinPoly, yMaxPoly):

        pointInPolygonList = []

        for polygon in polygons:
            # Point-in-Polygon Test und Verschiebung der Pixelkoordinaten
            in den Pixelmittelpunkt (Koordinate - 0.5):
                pointInPolygonList.append(polygon.contains_point([(x - 0.5), (y
- 0.5)]))

#.....

# Pixel muss innerhalb des aeusseren Ringes und ausserhalb der in-
neren Ringe liegen:
if pointInPolygonList[0] is True and pointInPolygon-
List[1:].count(False) == len(pointInPolygonList[1:]):

    # Abfrage der Farbwerte aus der Texturdatei und Umwandlung in
HSV:
    pixelRGB = workingTexture.getpixel((x,y))
    pixelHSV = colorconverter.rgb2hsv(pixelRGB[0], pixelRGB[1], pi-
xelRGB[2])

#.....

# Ausschiessen von warmen Fassaden (Farbwert wurde in Gimp er-
mittelt) und :
if pixelHSV != (0, 0, 50):
    pixelList.append(pixelHSV)####

```

```

#.....

# Bestimmung der Diversitaet der Farben innerhalb einer Gebaeu-
deflaeche: Unterscheidung zwischen Rot, Orange, Gelb, Gruen, Blau, Violett, Saettig-
ung, Helligkeit --> ergibt mit allen moeglichen Kombinationen 24 Klassen.
# Faellt ein Pixel in eine dieser Klassen, so wird der dazuge-
hoerige Wert im Dictionary "diverseDict" auf "True" geaendert. Je mehr Klassen von
einer Gebaeudeflaeche gefuellert werden, desto hoeher ist die Diversitaet:
    if (pixelHSV[0] >= 335 or pixelHSV[0] < 15) and (pixelHSV[1] <
50) and (pixelHSV[2] < 50):
        diverseDict[1] = True
    elif (pixelHSV[0] >= 335 or pixelHSV[0] < 15) and (pixelHSV[1]
< 50) and (pixelHSV[2] > 50):
        diverseDict[2] = True
    elif (pixelHSV[0] >= 335 or pixelHSV[0] < 15) and (pixel-
HSV[1] > 50) and (pixelHSV[2] < 50):
        diverseDict[3] = True
    elif (pixelHSV[0] >= 335 or pixelHSV[0] < 15) and (pixel-
HSV[1] > 50) and (pixelHSV[2] > 50):
        diverseDict[4] = True
    elif (pixelHSV[0] >= 15 and pixelHSV[0] < 45) and (pixelHSV[1]
< 50) and (pixelHSV[2] < 50):
        diverseDict[5] = True
    elif (pixelHSV[0] >= 15 and pixelHSV[0] < 45) and (pixelHSV[1]
< 50) and (pixelHSV[2] > 50):
        diverseDict[6] = True
    elif (pixelHSV[0] >= 15 and pixelHSV[0] < 45) and (pixel-
HSV[1] > 50) and (pixelHSV[2] < 50):
        diverseDict[7] = True
    elif (pixelHSV[0] >= 15 and pixelHSV[0] < 45) and (pixel-
HSV[1] > 50) and (pixelHSV[2] > 50):
        diverseDict[8] = True
    elif (pixelHSV[0] >= 45 and pixelHSV[0] < 65) and (pixelHSV[1]
< 50) and (pixelHSV[2] < 50):
        diverseDict[9] = True
    elif (pixelHSV[0] >= 45 and pixelHSV[0] < 65) and (pixelHSV[1]
< 50) and (pixelHSV[2] > 50):
        diverseDict[10] = True
    elif (pixelHSV[0] >= 45 and pixelHSV[0] < 65) and (pixel-
HSV[1] > 50) and (pixelHSV[2] < 50):
        diverseDict[11] = True
    elif (pixelHSV[0] >= 45 and pixelHSV[0] < 65) and (pixel-
HSV[1] > 50) and (pixelHSV[2] > 50):
        diverseDict[12] = True
    elif (pixelHSV[0] >= 65 and pixelHSV[0] < 165) and (pixelHSV[1]
< 50) and (pixelHSV[2] < 50):
        diverseDict[13] = True
    elif (pixelHSV[0] >= 65 and pixelHSV[0] < 165) and (pixelHSV[1]
< 50) and (pixelHSV[2] > 50):
        diverseDict[14] = True
    elif (pixelHSV[0] >= 65 and pixelHSV[0] < 165) and (pixel-
HSV[1] > 50) and (pixelHSV[2] < 50):
        diverseDict[15] = True
    elif (pixelHSV[0] >= 65 and pixelHSV[0] < 165) and (pixel-
HSV[1] > 50) and (pixelHSV[2] > 50):
        diverseDict[16] = True
    elif (pixelHSV[0] >= 165 and pixelHSV[0] < 270) and (pixel-
HSV[1] < 50) and (pixelHSV[2] < 50):
        diverseDict[17] = True
    elif (pixelHSV[0] >= 165 and pixelHSV[0] < 270) and (pixel-
HSV[1] < 50) and (pixelHSV[2] > 50):
        diverseDict[18] = True
    elif (pixelHSV[0] >= 165 and pixelHSV[0] < 270) and (pixel-
HSV[1] > 50) and (pixelHSV[2] < 50):
        diverseDict[19] = True
    elif (pixelHSV[0] >= 165 and pixelHSV[0] < 270) and (pixel-
HSV[1] > 50) and (pixelHSV[2] > 50):
        diverseDict[20] = True

```

```

        elif (pixelHSV[0] >= 270 and pixelHSV[0] < 335) and (pixel-
HSV[1] < 50) and (pixelHSV[2] < 50):
            diverseDict[21] = True
        elif (pixelHSV[0] >= 270 and pixelHSV[0] < 335) and (pixel-
HSV[1] < 50) and (pixelHSV[2] > 50):
            diverseDict[22] = True
        elif (pixelHSV[0] >= 270 and pixelHSV[0] < 335) and (pixel-
HSV[1] > 50) and (pixelHSV[2] < 50):
            diverseDict[23] = True
        elif (pixelHSV[0] >= 270 and pixelHSV[0] < 335) and (pixel-
HSV[1] > 50) and (pixelHSV[2] > 50):
            diverseDict[24] = True

#.....

    colors = np.array(pixelList)
    #ENDE# Befuellen von "pixelList", falls Pixel in Polygon liegt (entspricht
einem Clipping der Textur durch ein Polygon) #ENDE#
    #-----

    # Berechnung der Diversitaet durch Auswertung des "diverseDict":
    diversity = 0
    for i in range(1, 25):
        if diverseDict[i] == True:
            diversity += 1
    #-----

    # Berechnung der Clusteranzahl: Je hoeher die Diversitaet, desto hoeher die
Anzahl der Cluster "k":
    k = 12
    if diversity <= 4:
        k = 11
    elif diversity > 4 and diversity <= 8:
        k = 12
    elif diversity > 8 and diversity <= 12:
        k = 13
    elif diversity > 12 and diversity <= 16:
        k = 14
    elif diversity > 16 and diversity <= 20:
        k = 15
    elif diversity > 20 and diversity <= 24:
        k = 16
    #-----

    defaultVar = 0 #***

    # Farbwertberechnung mittels Clusteranalyse oder haeufigstem Wert (Modus):
    while k > 1:
        try:
            # Clusteranalyse:
            kmeans = KMeans(n_clusters = k).fit(colors)
            n_kmeans = kmeans.predict(colors)
            centers = kmeans.cluster_centers_
            clusterColor = tuple(centers[mode(n_kmeans)])
            """
            # Ausgabe des Farbarrays und der Clusteranzahl fuer die spaetere
Verwendung zum Plotten:
            print(list(colors.reshape(-1)))
            print(k)
            """
            break
        except:
            k -= 1
    if k == 1:
        try:
            # Modus:
            clusterColor = Counter(pixelList).most_common(1)[0][0]
        except:
            if inputResolution >= 2:

```

```

        inputResolution /= 2
        continue
    else:
        clusterColor = tuple((333, 100, 100))
        defaultVar = 1 #***
        break
    clusterColor = tuple((abs(int(clusterColor[0])), abs(int(clusterColor[1])),
abs(int(clusterColor[2]))))
#-----

#BEGINN# Veraenderung der Farbwerte je nach Exposition, Helligkeit und
Surface-Art #BEGINN#
#BEGINN# Surface Roof #BEGINN#
if surface == 2:

    #BEGINN# Helligkeit ok #BEGINN#
    if clusterColor[2] > 18:

        # Nordexposition:
        if exposition >= 315 or exposition < 45:

            # Ersetzen von Blau und Violett durch eine default-Farbe und
Kennzeichnung der Vergabe eines default-Wertes:
            if clusterColor[0] >= 155 and clusterColor[0] < 335:
                clusterColor = tuple((239, 100, 100))
                defaultVar = 1 #***

#.....

        # Veraenderung der Farbwerte:
        else:
            if clusterColor[1] < 10:
                if (clusterColor[2] + 30) >= 100:
                    clusterColor = tuple((clusterColor[0], clusterCo-
lor[1], 95))
                else:
                    clusterColor = tuple((clusterColor[0], clusterCo-
lor[1], (clusterColor[2] + 30)))
            else:
                if (clusterColor[2] + 30) >= 100:
                    clusterColor = tuple((clusterColor[0], (clusterCo-
lor[1] - 5), 95))
                else:
                    clusterColor = tuple((clusterColor[0], (clusterCo-
lor[1] - 5), (clusterColor[2] + 30)))

#.....

#.....

        # Ost- oder Westexposition:
        elif (exposition >= 45 and exposition < 135) or (exposition >= 225
and exposition < 315):

            # Ersetzen von Blau und Violett durch eine default-Farbe und
Kennzeichnung der Vergabe eines default-Wertes:
            if clusterColor[0] >= 155 and clusterColor[0] < 335:
                clusterColor = tuple((239, 100, 100))
                defaultVar = 1 #***

#.....

        # Veraenderung der Farbwerte:
        else:
            if clusterColor[1] < 5:
                if (clusterColor[2] + 15) >= 100:
                    clusterColor = tuple((clusterColor[0], clusterCo-
lor[1], 95))

```

```

        else:
            clusterColor = tuple((clusterColor[0], clusterCo-
lor[1], (clusterColor[2] + 15)))
        else:
            if (clusterColor[2] + 15) >= 100:
                clusterColor = tuple((clusterColor[0], clusterCo-
lor[1], 95))
            else:
                clusterColor = tuple((clusterColor[0], clusterCo-
lor[1], (clusterColor[2] + 15)))
#.....

#.....

        # Suedexposition:
        elif exposition >= 135 and exposition < 225:
            clusterColor = tuple((clusterColor[0], clusterColor[1], clus-
terColor[2]))
#.....

        #ENDE# Helligkeit ok #ENDE#
#.....

        # Zu dunkel:
        else:
            clusterColor = tuple((239, 100, 100))
            defaultVar = 1 #***
#.....

        #ENDE# Surface Roof #ENDE#
#.....

        #BEGINN# Surface Wall #BEGINN#
        elif surface == 1:

            #BEGINN# Helligkeit ok #BEGINN#
            if clusterColor[2] > 18:

                # Nordexposition:
                if exposition >= 315 or exposition < 45:

                    # Ersetzen von Blau und Violett durch eine default-Farbe und
Kennzeichnung der Vergabe eines default-Wertes:
                    if clusterColor[0] >= 155 and clusterColor[0] < 335:
                        clusterColor = tuple((56, 100, 100))
                        defaultVar = 1 #***
#.....

                    # Veraenderung der Farbwerte:
                    else:
                        if clusterColor[1] < 10:
                            if (clusterColor[2] + 16) >= 100:
                                clusterColor = tuple((clusterColor[0], clusterCo-
lor[1], 95))
                            else:
                                clusterColor = tuple((clusterColor[0], clusterCo-
lor[1], (clusterColor[2] + 16)))
                        else:
                            if (clusterColor[2] + 16) >= 100:
                                clusterColor = tuple((clusterColor[0], (clusterCo-
lor[1] - 10), 95))
                            else:

```

```

clusterColor = tuple((clusterColor[0], (clusterColor[1] - 10), (clusterColor[2] + 16)))
#.....

#.....

# Ost- oder Westexposition:
elif (exposition >= 45 and exposition < 135) or (exposition >= 225
and exposition < 315):

# Ersetzen von Blau und Violett durch eine default-Farbe und
Kennzeichnung der Vergabe eines default-Wertes:
if clusterColor[0] >= 155 and clusterColor[0] < 335:
    clusterColor = tuple((56, 100, 100))
    defaultVar = 1 #***
#.....

# Veraenderung der Farbwerte:
else:
    if clusterColor[1] < 5:
        if (clusterColor[2] + 8) >= 100:
            clusterColor = tuple((clusterColor[0], clusterColor[1], 95))
        else:
            clusterColor = tuple((clusterColor[0], clusterColor[1], (clusterColor[2] + 8)))
        else:
            if (clusterColor[2] + 8) >= 100:
                clusterColor = tuple((clusterColor[0], (clusterColor[1] - 5), 95))
            else:
                clusterColor = tuple((clusterColor[0], (clusterColor[1] - 5), (clusterColor[2] + 8)))
#.....

#.....

# Suedexposition:
elif exposition >= 135 and exposition < 225:
    if (clusterColor[2] + 5) >= 100:
        clusterColor = tuple((clusterColor[0], clusterColor[1], 95))
    else:
        clusterColor = tuple((clusterColor[0], clusterColor[1], (clusterColor[2] + 5)))
#.....

#ENDE# Helligkeit ok #ENDE#
#.....

# Zu dunkel:
else:
    clusterColor = tuple((56, 100, 100))
    defaultVar = 1 #***
#.....

#ENDE# Surface Wall #ENDE#
#.....

```

```

#ENDE# Veraenderung der Farbwerte je nach Exposition, Helligkeit und
Surface-Art #ENDE#
#-----

#BEGINN# Generelles Aufhellen der Farbwerte in unterschiedlicher Inten-
sitaet (Dach oder Fassade) #BEGINN#
if surface == 2:
    if clusterColor[2] >= 75 and clusterColor[2] < 85:
        clusterColor = tuple((clusterColor[0], clusterColor[1], 75))
    elif clusterColor[2] >= 85:
        clusterColor = tuple((clusterColor[0], clusterColor[1], 85))
    else:
        pass
elif surface == 1:
    if clusterColor[2] > 25 and clusterColor[2] <= 50:
        clusterColor = tuple((clusterColor[0], (clusterColor[1]), (cluster-
Color[2] + 10)))
    elif clusterColor[2] <= 25:
        clusterColor = tuple((clusterColor[0], (clusterColor[1]), (cluster-
Color[2] + 15)))
    else:
        pass
#ENDE# Generelles Aufhellen der Farbwerte in unterschiedlicher Intensitaet
(Dach oder Fassade) #ENDE#
#-----

changeRes = False #####+++
#-----

return clusterColor, defaultVar
#ENDE# while-Schleife, damit Aufloesung so lange erhoeht wird, bis ein Farbwert
berechnet werden kann #ENDE#
#-----

#ENDE# Funktion zur Farbwertberechnung mittels Clusteranalyse #ENDE#
#-----
#####
#####

```

Farbwertextraktion (*farbwertextraktion.py*)

```

#####

####---Quellen:---####
# Parser fuer Command Line Optionen: https://docs.python.org/2/library/opt-
parse.html
# Python exit commands: https://www.geeksforgeeks.org/python-exit-commands-quit-
exit-sys-exit-and-os-\_exit/
# Python os Modul: https://docs.python.org/3/library/os.html
# Python XML-Parser xml.etree.ElementTree: https://docs.py-
thon.org/3/library/xml.etree.elementtree.html
# Python XML Parser Tutorial: https://www.youtube.com/watch?v=r6dyk68gymk
# Parsing XML with namespace in Python via ElementTree: https://stackoverflow-
flow.com/questions/14853243/parsing-xml-with-namespace-in-python-via-elementtree
# Python numpy: https://www.w3schools.com/python/numpy\_intro.asp
# Konvexe Huelle: https://docs.scipy.org/doc/scipy/reference/generated/scipy.spa-
tial.ConvexHull.html
# Equation of a plane through three points: http://kit-
chingroup.cheme.cmu.edu/blog/2015/01/18/Equation-of-a-plane-through-three-points/
# Normalvektor bei Ebenen: https://www.youtube.com/watch?v=CHn\_yyU8yPY
# n-Vektor bestimmen mit dem Kreuzprodukt: https://www.y-
outube.com/watch?v=UzWnp97GN9g
# Python Imaging Library Pillow: https://pillow.readthedocs.io/en/stable/
# How to: Insert a subelement into an XML element: https://kite.com/python/examp-
les/3596/xml-insert-a-subelement-into-an-xml-element

```

```

# Element Library Functions (XML wird lesbar dargestellt): https://eff-
bot.org/zone/element-lib.htm
# Pretty printing XML in Python: https://stackoverflow.com/questions/749796/pretty-
printing-xml-in-python
#####

####---Erklaerung der verwendeten Variablen:---####
# start: Startzeit des Skripts.
# parser: Eingaben des Option-Parser.
# directory: wird durch Option-Parser mit Pfad zu den Kacheln befuellt.
# filepath: wird durch Option-Parser mit Pfad zu einer Kachel befuellt.
# inputResolution: Divisor fuer die Reduktion der Aufloesung (default = 16).
# fileList: Liste aller GML-Dateien, welche sich im "directory" befinden.
# directoryName: Hilfsvariable fuer die Befuellung der "fileList".
# filename: Hilfsvariable fuer die Befuellung der "fileList".
# textureGML: texturiertes LOD2-Stadtmodells (GML-Datei).
# colorGML: Kopie des texturierten LOD2-Stadtmodells (GML-Datei), in welche spaeter
die Farbwerte geschrieben werden.
# treeTX: geparster XML-Baum der originalen GML-Datei.
# CityModelTX: Wurzelelement der GML-Datei.
# treeCL: geparster XML-Baum der mit Farbwerten zu befuellenden GML-Datei.
# CityModelCL: Wurzelelement der mit Farbwerten zu befuellenden GML-Datei.
# namespaces: Dictionary mit Namespaces der GML-Dateien.
# surfaceDict: Dictionary zum Abspeichern der der Surface-Art (Grundflaeche, Mauer
oder Dach) als Paar ID:Surface-Art.
# expositionDict: Dictionary zum Abspeichern der Exposition je Flaeche.
# koordinaten: Koordinaten des Polygons, von dem die Exposition berechnet werden
soll als String (Koordinaten durch Leerzeichen getrennt).
# coord_split_string: Liste mit Koordinaten als String, welche an den Leerzeichen
aufgesplittet wurden.
# coords: Liste mit Koordinaten als float.
# coordsArray: Koordinaten als Array.
# coordsArrayXY: x- und y-Koordinaten als Array (z-Koordinate geloescht).
# coordsArrayXZ: x- und z-Koordinaten als Array (y-Koordinate geloescht).
# coordsArrayYZ: y- und z-Koordinaten als Array (x-Koordinate geloescht).
# fortsetzung: Hilfsvariable zur Steuerung der while-Schleife fuer die Berechnung
der konvexen Huellen.
# convexXY: konvexe Huelle der Punkte in x-y Richtung.
# convexXZ: konvexe Huelle der Punkte in x-z Richtung.
# convexYZ: konvexe Huelle der Punkte in y-z Richtung.
# commonIndices: jedes Element eines Arrays kann ueber Indizes angesprochen werden.
"convex.vertices" liefert die Indizes, welche eine konvexe Huelle ergeben. Diese
Variable speichert alle Indizes, welche in jeder Betrachtungsrichtung vorkommen
(xy, xz, yz).
# index: Zaehlvariable fuer Indizes des "coordsArray".
# coordsIndicesConvex: wird mit True oder False befuellt - je nach dem, ob der ak-
tuelle Wert von "index" in "commonIndices" vorhanden ist oder nicht.
# coordsConvex: konvexe Koordinaten als Array.
# r: Vektor, welcher zusammen mit dem Vektor "s" eine Flaeche aufspannt.
# s: Vektor, welcher zusammen mit dem Vektor "r" eine Flaeche aufspannt.
# n: Normalvektor.
# b: Winkel beta.
# a: Winkel alpha = Exposition.
# colorDict: Dictionary zum Abspeichern der Farwerte je Flaeche.
# textureUri: relativer Dateipfad (von der GML-Datei aus gesehen) zur Texturdatei.
# texturefileTX: Dateipfad zur Texturdatei.
# textureTX: mit PIL-Library geoeffnete Texturdatei.
# surfaceUri: ID der aktuellen Flaeche.
# textureCoordinatesListTX: Liste, welche die Strings der einzelnen Polygone (in-
nere und aeussere Ringe) beinhaltet.
# surface: Ergebnis der Abfrage der Surface-Art aus dem "surfaceDict".
# exposition: Ergebnis der Abfrage der Exposition aus dem "expositionDict".
# roofColorList: Liste aller Dachfarben pro Gebaeude, welche keine Defaultwerte
sind.
# wallColorList: Liste aller Fassadenfarben pro Gebaeude, welche keine Defaultwerte
sind.
# ParameterizedTextureTX: Hilfsvariable fuer die Navigation in der GML-Datei.
# modeColorRoof: haeufigste Dachfarbe pro Gebaeude.
# modeColorWall: haeufigste Fassadenfarbe pro Gebaeude.

```

```

# colorvalueRGB: endgueltiger Farbwert im RGB-Format als Tuple.
# colorvalueString: endgueltiger Farbwert in der Syntax fuer CityGML (relative RGB-
Werte als String, getrennt durch Leerzeichen).
# X3DMaterial: Hilfsvariable zur Erstellung eines neuen Materials fuer die Abspei-
cherung des berechneten Farbwerts.
# element: Hilfsvariable zur Erstellung eines neuen Materials fuer die Abspeiche-
rung des berechneten Farbwerts.
# ende: Endzeit des Skripts.
#####

####---Postfixes fuer Variablennamen:---####
# Postfix TX: fuer jeden Namen, der in beiden CityGML-Dateien vorkommt und die Da-
tei mit der Texturierung betrifft
# Postfix CL: fuer jeden Namen, der in beiden CityGML-Dateien vorkommt und die Da-
tei mit den Farbwerten betrifft
#####

# Import der benoetigten Librarys:
from collections import Counter
import color
import colorconverter
import math
import numpy as np
from optparse import OptionParser
import os
from PIL import Image
from statistics import mode, mean
from scipy.spatial import ConvexHull
import sys
import time
import xml.etree.ElementTree as ET
#-----

start = time.time()

# Implementierung eines Option Parsers:

parser = OptionParser()

parser.add_option("-d", "--directory", dest = "directory", default = "directory",
help = "directory with folders containing tiles (.gml files)", metavar = "direc-
tory")
parser.add_option("-f", "--filepath", dest = "filepath", default = "filepath", help =
"directory (folder) containing the .gml file", metavar = "filepath")
parser.add_option("-r", "--resolution", dest = "resolution", default = "16", help =
"resolution of input image (original resolution is divided by this number)", me-
tavar = "image resolution")

(options, args) = parser.parse_args()

directory = options.directory
filepath = options.filepath
inputResolution = int(options.resolution)

# Bei "directory" und "filepath" gibt es keine brauchbaren Defaultwerte - diese
fuehren daher zu einer Fehlermeldung:
if (directory == "directory" and filepath == "filepath"):
    sys.exit("\nPlease enter a directory!\n\nUse -d DIRECTORY WHICH CONTAINS FOL-
DERS THAT CONTAIN A GML FILE\nto set the folder where the script should iterate o-
ver the folders with tiles.\n\nUse -f DIRECTORY WHICH CONTAINS A GML FILE to set the path to
the .gml file.")
# .....

# Abfangen eines ungueltigen Werts fuer den Divisor, welcher spaeter die Aufloesung
veraendern soll:
if inputResolution < 1:
    inputResolution = 16

```

```

    print("Ungueltiger Wert fuer -r/--resolution, default-Wert 16 wird uebernom-
men...")
# .....
#-----

# Vorbereitung fuer die automatische Abarbeitung aller Kacheln in einem Verzeich-
nis, um eine spaetere Iteration zu ermoeeglichen:
fileList = []

directory = directory.replace("\\", "/")
directoryName = directory.split("/")
directoryName = directoryName[-1]
if directory != "directory":
    for filepath in os.listdir(directory):
        filename = filepath.split("/")
        filename = filename[-1]
        fileList.append(directory + "/" + filepath + "/" + filename)

filepath = filepath.replace("\\", "/")
filename = filepath.split("/")
filename = filename[-1]

if filepath != "filepath" and directory == "directory":
    fileList.append(filepath + "/" + filename)
#-----

#BEGINN# Iteration durch alle Kacheln in einem Verzeichnis #BEGINN#
for file in fileList:

    print("Processing " + file + ".gml" + ":\n")

    # Kopieren der originalen CityGML-Datei und Veraenderung des Dateinamens:
    textureGML = open(file + ".gml")
    colorGML = open(file + "_color" + "_" + str(inputResolution) + "res" + ".gml",
"w")
    colorGML.write(textureGML.read())

# .....

# Registrierung der Namespaces, damit sie beim Schreiben des XML-Files erhalten
bleiben:
ET.register_namespace('xal', 'urn:oasis:names:tc:ciq:xsd:schema:xAL:2.0')
ET.register_namespace('gml', 'http://www.opengis.net/gml')
ET.register_namespace('app', 'http://www.opengis.net/citygml/appearance/1.0')
ET.register_namespace('bldg', 'http://www.opengis.net/citygml/building/1.0')
ET.register_namespace('gen', 'http://www.opengis.net/citygml/generics/1.0')
ET.register_namespace('xlink', 'http://www.w3.org/1999/xlink')
ET.register_namespace('xsi', 'http://www.w3.org/2001/XMLSchema-instance')
ET.register_namespace('schemaLocation', 'http://www.opengis.net/citygml/buil-
ding/1.0 http://schemas.opengis.net/citygml/building/1.0/building.xsd http://www.o-
pengis.net/citygml/cityfurniture/1.0 http://schemas.opengis.net/citygml/cityfurni-
ture/1.0/cityFurniture.xsd http://www.opengis.net/citygml/appearance/1.0
http://schemas.opengis.net/citygml/appearance/1.0/appearance.xsd http://www.open-
gis.net/citygml/cityobjectgroup/1.0 http://schemas.opengis.net/citygml/cityobject-
group/1.0/cityObjectGroup.xsd http://www.opengis.net/citygml/texturedsurface/1.0
http://schemas.opengis.net/citygml/texturedsurface/1.0/texturedSurface.xsd
http://www.opengis.net/citygml/generics/1.0 http://schemas.opengis.net/citygml/ge-
nerics/1.0/generics.xsd http://www.opengis.net/citygml/relief/1.0 http://schemas.o-
pengis.net/citygml/relief/1.0/relief.xsd http://www.opengis.net/citygml/vegeta-
tion/1.0 http://schemas.opengis.net/citygml/vegetation/1.0/vegetation.xsd
http://www.opengis.net/citygml/waterbody/1.0 http://schemas.opengis.net/citygml/wa-
terbody/1.0/waterBody.xsd http://www.opengis.net/citygml/transportation/1.0
http://schemas.opengis.net/citygml/transportation/1.0/transportation.xsd
http://www.opengis.net/citygml/landuse/1.0 http://schemas.opengis.net/citygml/lan-
duse/1.0/landUse.xsd')

# .....

# Parsen der originalen CityGML-Datei:

```

```

treeTX = ET.parse(file + ".gml")
CityModelTX = treeTX.getroot()

# .....

# Parsen der kopierten CityGML-Datei:
treeCL = ET.parse(file + "_color" + "_" + str(inputResolution) + "res" +
".gml")
CityModelCL = treeCL.getroot()

# .....
# Angabe der Namespaces als Dictionary, damit sie mit der Suchfunktion von et-
ree (findall) verwendet werden koennen:
namespaces = {'xal': 'urn:oasis:names:tc:ciq:xsd:schema:xAL:2.0', 'gml':
'http://www.opengis.net/gml', 'app': 'http://www.opengis.net/ci-
tygml/appearance/1.0', 'xmlns': 'http://www.opengis.net/citygml/1.0',
'tex': 'http://www.opengis.net/citygml/texturedsurface/1.0',
'dem': 'http://www.opengis.net/citygml/relief/1.0', 'veg': 'http://www.open-
gis.net/citygml/vegetation/1.0', 'wtr': 'http://www.opengis.net/citygml/wa-
terbody/1.0',
'tran': 'http://www.opengis.net/citygml/transportation/1.0',
'bldg': 'http://www.opengis.net/citygml/building/1.0', 'frn': 'http://www.open-
gis.net/citygml/cityfurniture/1.0', 'grp': 'http://www.opengis.net/citygml/cityob-
jectgroup/1.0',
'gen': 'http://www.opengis.net/citygml/generics/1.0', 'xlink':
'http://www.w3.org/1999/xlink', 'luse': 'http://www.opengis.net/citygml/lan-
duse/1.0', 'xsi': 'http://www.w3.org/2001/XMLSchema-instance',
'schemaLocation': 'http://www.opengis.net/citygml/building/1.0
http://schemas.opengis.net/citygml/building/1.0/building.xsd http://www.open-
gis.net/citygml/cityfurniture/1.0 http://schemas.opengis.net/citygml/cityfurni-
ture/1.0/cityFurniture.xsd http://www.opengis.net/citygml/appearance/1.0
http://schemas.opengis.net/citygml/appearance/1.0/appearance.xsd http://www.open-
gis.net/citygml/cityobjectgroup/1.0 http://schemas.opengis.net/citygml/cityobject-
group/1.0/cityObjectGroup.xsd http://www.opengis.net/citygml/texturedsurface/1.0
http://schemas.opengis.net/citygml/texturedsurface/1.0/texturedSurface.xsd
http://www.opengis.net/citygml/generics/1.0 http://schemas.opengis.net/citygml/ge-
nerics/1.0/generics.xsd http://www.opengis.net/citygml/relief/1.0 http://schemas.o-
pengis.net/citygml/relief/1.0/relief.xsd http://www.opengis.net/citygml/vegeta-
tion/1.0 http://schemas.opengis.net/citygml/vegetation/1.0/vegetation.xsd
http://www.opengis.net/citygml/waterbody/1.0 http://schemas.opengis.net/citygml/wa-
terbody/1.0/waterBody.xsd http://www.opengis.net/citygml/transportation/1.0
http://schemas.opengis.net/citygml/transportation/1.0/transportation.xsd
http://www.opengis.net/citygml/landuse/1.0 http://schemas.opengis.net/citygml/lan-
duse/1.0/landUse.xsd'}

# .....

# Erstellen eines Dictionarys zur Abfrage der Surface-Art:
surfaceDict = {}
for boundedByTX in CityModelTX.iter(r'{http://www.opengis.net/citygml/buil-
ding/1.0}boundedBy'):
    if boundedByTX[0].tag[45:] == "RoofSurface":
        surfaceDict[boundedByTX[0].attrib['{http://www.opengis.net/gml}id'] +
'_poly'] = 2
    elif boundedByTX[0].tag[45:] == "WallSurface":
        surfaceDict[boundedByTX[0].attrib['{http://www.opengis.net/gml}id'] +
'_poly'] = 1
    elif boundedByTX[0].tag[45:] == "GroundSurface":
        surfaceDict[boundedByTX[0].attrib['{http://www.opengis.net/gml}id'] +
'_poly'] = 0
#-----

#BEGINN# Erstellen eines Dictionarys zur Abfrage der Exposition #BEGINN#

expositionDict = {}
for PolygonTX in CityModelTX.iter(r'{http://www.opengis.net/gml}Polygon'):

```

```

# Abfrage und Aufbereitung der Koordinaten:
koordinaten = PolygonTX[0][0][0].text

coord_split_string = koordinaten.split(" ")

coords = []

for css in coord_split_string:
    coords.append(float(css))

coordsArray = np.array(coords)

coordsArrayXY = np.delete(coordsArray, np.arange(2, coordsArray.size, 3))
coordsArrayXZ = np.delete(coordsArray, np.arange(1, coordsArray.size, 3))
coordsArrayYZ = np.delete(coordsArray, np.arange(0, coordsArray.size, 3))

coordsArray = coordsArray.reshape(-1, 3)

coordsArrayXY = coordsArrayXY.reshape(-1, 2)
coordsArrayXZ = coordsArrayXZ.reshape(-1, 2)
coordsArrayYZ = coordsArrayYZ.reshape(-1, 2)

# .....

# Die Exposition ergibt sich aus der "Drehrichtung" der Punkte eines Poly-
gons.
# Damit diese immer die Selbe bleibt, muessen die Punkte konvex angeordnet
sein.

# Berechnung der konvexen Huellen fuer 3 Ansichten:
fortsetzung = True
while fortsetzung == True:
    # XY:
    try:
        convexXY = ConvexHull(coordsArrayXY)
        fortsetzung = False
    except:
        try:
            convexXY = convexXZ
            fortsetzung = False
        except:
            try:
                convexXY = convexYZ
                fortsetzung = False
            except:
                pass
    # XZ:
    try:
        convexXZ = ConvexHull(coordsArrayXZ)
    except:
        try:
            convexXZ = convexXY
            fortsetzung = False
        except:
            try:
                convexXZ = convexYZ
                fortsetzung = False
            except:
                pass
    # YZ:
    try:
        convexYZ = ConvexHull(coordsArrayYZ)
    except:
        try:
            convexYZ = convexXY
        except:
            try:
                convexYZ = convexXZ
            except:

```

```

pass

# .....

    # Annahme fuer 3D-Gebaeude: konkave Punkte erscheinen in jeder Ansicht konkav (Ausnahme: Draufsicht)
    # Daher werden nur jene Punkte behalten, welche in jeder Ansicht konvex sind.
    # Ist ein Punkt in einer Ansicht konkav, so wird er entfernt.

    # Nur jene Indizes werden uebernommen, welche in allen 3 Arrays vorhanden sind:
    commonIndices = set(convexXY.vertices) & set(convexXZ.vertices) & set(convexYZ.vertices)

    # Mit diesen Indizes ("commonIndices") koennen aus dem originalen 3D-Array (hier: "coordsArray") alle konvexen Punkte heraus gefiltert werden.

    index = 0
    coordsIndicesConvex = []

    # Hinzufuegen der konvexen Indizes zur Liste "coordsIndicesConvex", wenn Index eines Elements von "coordsArray" in "commonIndices" vorkommt:
    for element in coordsArray:
        if index in commonIndices:
            coordsIndicesConvex.append(True)
        else:
            coordsIndicesConvex.append(False)
        index += 1

# .....
    coordsIndicesConvex = np.array(coordsIndicesConvex)
    # Elemente mit Index aus "coordsIndicesConvex" werden in "coordsConvex" verschoben:
    coordsConvex = coordsArray[coordsIndicesConvex]

    # Berechnung der beiden Vektoren, welche die Flaeche aufspannen:
    try:
        r = coordsConvex[1] - coordsConvex[0]
        s = coordsConvex[2] - coordsConvex[0]
    except:
        r = coordsArray[1] - coordsArray[0]
        s = coordsArray[2] - coordsArray[0]

# .....

    # Berechnung der Zielkoordinaten des Normalvektors "n":
    n = np.cross(r, s)

    # Berechnung des Expositionswinkels alpha (a) aus dem Winkel beta (b):
    b = abs(math.atan(n[0] / n[1]) * (180 / math.pi))

    if n[0] > 0 and n[1] > 0:
        a = b
    elif n[0] > 0 and n[1] < 0:
        a = 180 - b
    elif n[0] < 0 and n[1] < 0:
        a = 180 + b
    elif n[0] < 0 and n[1] > 0:
        a = 360 - b

# .....

    try:
        expositionDict[PolygonTX.attrib['{http://www.opengis.net/gml}id']] = a
    except:
        expositionDict[PolygonTX.attrib['{http://www.opengis.net/gml}id']] = 0

```

```

# Updaten des Exposition-Dictionarys (ersetzen der Exposition der Dachflaechen
durch jene des Attributs "Dachorientierung"):
for roofSurfaceTX in CityModelTX.iter(r'{http://www.opengis.net/citygml/build-
ing/1.0}RoofSurface'):
    for doubleAttributeTX in roofSurfaceTX.findall('gen:doubleAttribute', name-
spaces):
        if doubleAttributeTX.attrib['name'] == 'Dachorientierung':
            if float(doubleAttributeTX[0].text) == -1.0:
                expositionDict[roofSurfaceTX.attrib['{http://www.open-
gis.net/gml}id'] + '_poly'] = 180.0
            else:
                expositionDict[roofSurfaceTX.attrib['{http://www.open-
gis.net/gml}id'] + '_poly'] = abs(float(doubleAttributeTX[0].text))

# .....
#ENDE# Erstellen eines Dictionarys zur Abfrage der Exposition #ENDE#
#-----

# Farbwertberechnung und Erstellen eines Dictionarys zur Abfrage der Farben:
colorDict = {}
for ParameterizedTextureTX in CityModelTX.iter(r'{http://www.opengis.net/ci-
tygml/appearance/1.0}ParameterizedTexture'):

    ## Oeffnen der Texturdatei:
    for imageUri in ParameterizedTextureTX.findall('app:imageURI', namespaces):
        textureUri = imageUri.text
        texturefileTX = file[:-12] + "/" + textureUri
        print(texturefileTX)
        # Import der Textur-Datei:
        textureTX = Image.open(texturefileTX)
        textureTX = textureTX.rotate(270, expand = 1)

# .....

# Auslesen der Texturkoordinaten und Berechnung der Farbwerte:
for targetTX in ParameterizedTextureTX.findall('app:target', namespaces):
    surfaceUri = targetTX.get('uri')[1:]
    textureCoordinatesListTX = []
    for texCoordListTX in targetTX.findall('app:TexCoordList', namespaces):
        for textureCoordinatesTX in texCoordListTX.findall('app:tex-
tureCoordinates', namespaces):
            textureCoordinatesListTX.append(textureCoordinatesTX.text)

    try:
        surface = surfaceDict[surfaceUri]

        exposition = expositionDict[surfaceUri]

        # Berechnung der Farbwerte:
        colorDict[surfaceUri] = color.extract(inputResolution, textureTX,
textureCoordinatesListTX, surface, exposition)
    except:
        # Falls eine ID das Wort "citygrid" beinhaltet:
        print("citygrid error: colorDict")

# .....
#-----

#BEGINN# Ersetzen der Defaultwerte durch den haeufigsten Farbwert pro Dachflae-
che oder Fassade je Gebaeude und Schreiben der Farbwerte in die neu erstellte Ci-
tyGML-Datei #BEGINN#

for cityObjectMemberTX, cityObjectMemberCL in zip(CityModelTX, CityModelCL):

    for AppearanceTX, AppearanceCL in zip(cityObjectMemberTX.i-
ter(r'{http://www.opengis.net/citygml/appearance/1.0}Appearance'), cityObjectMem-
berCL.iter(r'{http://www.opengis.net/citygml/appearance/1.0}Appearance')):

```

```

# Aendern des Namens von "Bildflug 2013" auf "Farbwertextraktion":
for themeCL in AppearanceCL.findall('app:theme', namespaces):
    themeCL.text = "Farbwertextraktion"

#.....

# Ermittlung des haeufigsten Farbwertes pro Dach oder Fassade je Ge-
baeude:
roofColorList = []
wallColorList = []

for surfaceDataMemberTX in AppearanceTX.findall('app:surfaceDataMem-
ber', namespaces):

    ParameterizedTextureTX = surfaceDataMemberTX.find('app:Paramete-
rizedTexture', namespaces)

    # Befuellung der Liste "roofColorList" oder "wallColorList"
wenn Farbe ein Defaultwert ist:
    for targetTX in ParameterizedTextureTX.findall('app:target',
namespaces):
        surfaceUri = targetTX.get('uri')[1:]

        try:
            surface = surfaceDict[surfaceUri]

            if surface == 2 and colorDict[surfaceUri][1] == 0:
                roofColorList.append(colorDict[surfaceUri][0])
            elif surface == 1 and colorDict[surfaceUri][1] == 0:
                wallColorList.append(colorDict[surfaceUri][0])
        except:
            # Falls eine ID das Wort "citygrid" beinhaltet:
            print("citygrid error: modeColorRoof/modeColorWall")

#.....

# Ermittlung des haeufigsten Wertes:
try:
    if roofColorList != []:
        modeColorRoof = Counter(roofColorList).most_common(1)[0][0]
    else:
        modeColorRoof = tuple((20, 4, 60))

    if wallColorList != []:
        modeColorWall = Counter(wallColorList).most_common(1)[0][0]
    else:
        modeColorWall = tuple((33, 4, 80))
except:
    modeColorRoof = tuple((0, 100, 100))
    modeColorWall = tuple((0, 100, 100))

#.....
#-----

# Schreiben der Farbwerte in die neue CityGML-Datei:
for surfaceDataMemberTX in AppearanceTX.findall('app:surfaceDataMem-
ber', namespaces):

    ParameterizedTextureTX = surfaceDataMemberTX.find('app:Paramete-
rizedTexture', namespaces)

    # Loeschen des Bereichs, in dem die Texturen referenziert werden -
wird in der CityGML-Datei, welche die Farbwerte beinhalten soll, nicht benoetigt:
    for surfaceDataMemberCL in AppearanceCL.findall('app:surfaceData-
Member', namespaces):
        AppearanceCL.remove(surfaceDataMemberCL)

# .....
```

```

# Auslesen der Surface Uri:
positionIndex = 1
for targetTX in ParameterizedTextureTX.findall('app:target', name-
spaces):
    surfaceUri = targetTX.get('uri')[1:]

    # Entscheidung, ob tatsaechlicher Farbwert genommen werden
    kann, oder ob auf den haeufigsten Wert pro Gebaeude zurueckgegriffen werden muss,
    weil der Farbwert ein Defaultwert ist:
    try:
        surface = surfaceDict[surfaceUri]

        if colorDict[surfaceUri][1] == 1:
            if surfaceDict[surfaceUri] == 2:
                colorvalueRGB = colorconverter.hsv2rgb(modeColor-
Roof[0], modeColorRoof[1], modeColorRoof[2])
            elif surfaceDict[surfaceUri] == 1:
                colorvalueRGB = colorconverter.hsv2rgb(modeColor-
Wall[0], modeColorWall[1], modeColorWall[2])
            elif colorDict[surfaceUri][1] == 0:
                colorvalueRGB = colorconverter.hsv2rgb(color-
Dict[surfaceUri][0][0], colorDict[surfaceUri][0][1], colorDict[surfaceUri][0][2])
        except:
            colorvalueRGB = tuple((35, 5, 70))
            # Falls eine ID das Wort "citygrid" beinhaltet:
            print("citygrid error: Farbzuzuweisung")

    # Umwandeln des Farbwertes in das fuer CityGML erforderliche
    Format:

    rValue = str(round((colorvalueRGB[0]/255), 2))
    gValue = str(round((colorvalueRGB[1]/255), 2))
    bValue = str(round((colorvalueRGB[2]/255), 2))

    if float(rValue) > 1:
        rValue = str(1)
    else:
        pass

    if float(gValue) > 1:
        gValue = str(1)
    else:
        pass

    if float(bValue) > 1:
        bValue = str(1)
    else:
        pass

    colorvalueString = str(rValue + ' ' + gValue + ' ' + bValue)

# .....
#-----

    # Erstellen eines Materials in der neuen GML-Datei fuer die
    Speicherung der Farbwerte und Schreiben des Farbwertes "colorvalueString":
    surfaceDataMember = ET.Element("app:surfaceDataMember")
    AppearanceCL.insert(positionIndex, surfaceDataMember)
    X3DMaterial = ET.Element("app:X3DMaterial")
    surfaceDataMember.insert(0, X3DMaterial)

    # Eigenschaften vom Material:

    # Uri: stellt Verknuepfung zwischen Material (=Farbe) zur Flae-
    che am Gebaeude her:
    def newElement(name, content):
        element = ET.Element(str("app:" + name))
        element.text = content
        X3DMaterial.insert(0, element)

```

```

        newElement("target", "#" + surfaceUri)
        #newElement("isSmooth", "false")
        #newElement("transparency", "0.0")
        #newElement("shininess", "0.2")
        newElement("diffuseColor", colorvalueString)
        #newElement("ambientIntensity", "0.2")
        #-----

        positionIndex += 1

    #ENDE# Ersetzen der Defaultwerte durch den haeufigsten Farbwert pro Dachflaeche
    oder Fassade je Gebaeude und Schreiben der Farbwerte in die neu erstellte CityGML-
    Datei #ENDE#
    #-----

    # Funktion zum Aufbereiten der zuvor geaenderten CityGML-Datei (alle Elemente
    in eiener Zeile -> schwer lesbar):
    def indent(elem, level = 0):
        i = "\n" + level*"    "
        if len(elem):
            if not elem.text or not elem.text.strip():
                elem.text = i + "    "
            if not elem.tail or not elem.tail.strip():
                elem.tail = i
            for elem in elem:
                indent(elem, level + 1)
            if not elem.tail or not elem.tail.strip():
                elem.tail = i
        else:
            if level and (not elem.tail or not elem.tail.strip()):
                elem.tail = i
        return elem
    #-----

    # Anwenden der zuvor definierten Funktion "indent":
    indent(CityModelCL)

    # Schreiben aller Aenderungen in die zu Beginn kopierte Datei:
    treeCL.write(file + "_color" + "_" + str(inputResolution) + "res" + ".gml",
    xml_declaration = True)

    print("\n\n")
    #ENDE# Iteration durch alle Kacheln in einem Verzeichnis #ENDE#
    #-----

    # Berechnung und Ausgabe der Laufzeit dieses Skripts:
    ende = time.time()
    print('Die Ausführung des Programms dauerte {:.3f}s.'.format(ende-start), end='
    ')
    #-----
    #####
    #####

```

Erklärung

Hiermit versichere ich,

- dass ich die vorliegende Masterarbeit selbstständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubter Hilfe bedient habe,
- dass ich dieses Masterarbeitsthema bisher weder im In- noch im Ausland in irgendeiner Form als Prüfungsarbeit vorgelegt habe
- und dass diese Arbeit mit der vom Begutachter beurteilten Arbeit vollständig übereinstimmt.

Ort

Datum

Unterschrift