UNIVERSITÄT BERN
OESCHGER CENTRE | Mobiliar Lab für Naturrisiken

Visualisierung von Hochwassersimulationen in 3D und 4D

Machbarkeitsstudie

Rafael Wampfler

27.02.2018

Inhalt

Projekt

Dieses Dokument und die Implementationen wurden als „UP 20: Visualisierung von Hochwassersimulationen in 3D und 4D“ erstellt. Es dauerte 3 Monate (November 2017 bis Februar 2018).

Ausgangslage

Für einen risikobasierten oder ganzheitlichen Umgang mit Naturgefahren in einer Gemeinde ist eine umfassende Information der Bevölkerung notwendig. Gute Visualisierungen von Naturgefahren und Risiken unterstützen die Sensibilisierung von Betroffenen und Planern.

Bis heute werden dafür hauptsächlich historische Fotos, Karten von Überflutungssituationen (2D), oder Filme von Hochwassersimulationen (2D) verwendet. Die Hochwassersimulationen können auf Satellitenbilder (Abbildung 1) oder Reliefbilder (Abbildung 2) gelegt werden, um die Situation zu erklären.

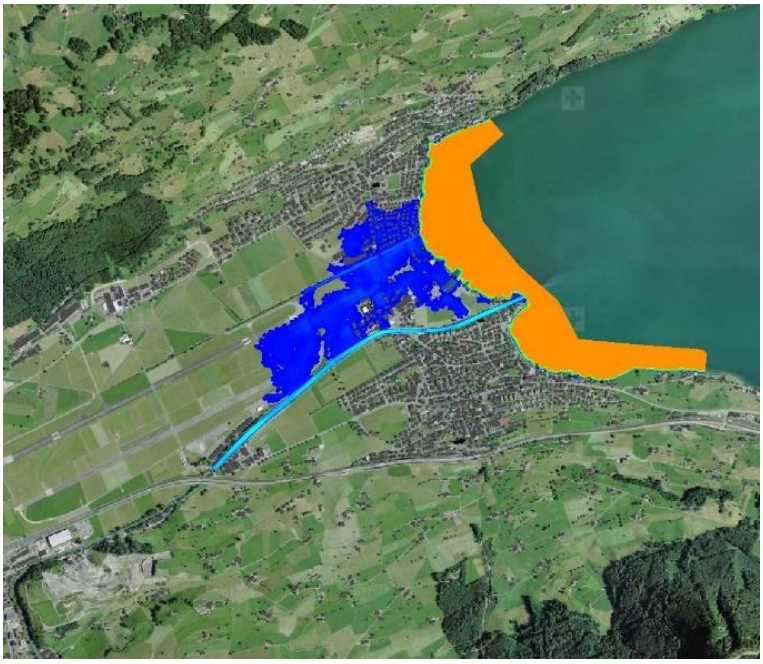

Abbildung 1 Simulation mit Satellitenbild

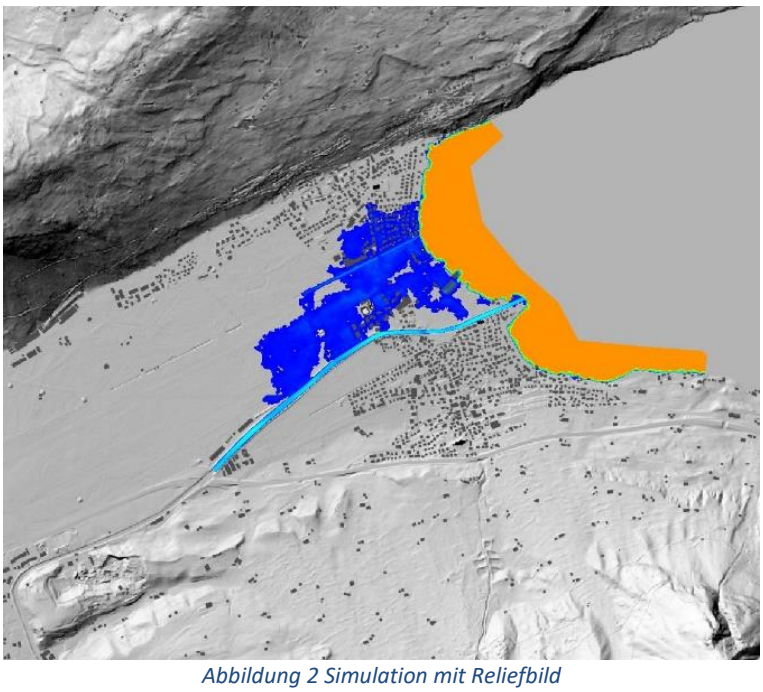

Abbildung 2 Simulation mit Reliefbild

Im Gegensatz zu historischen Fotos von Hochwasserereignissen in der jeweiligen Gemeinde machen Karten die Gefahr nicht „erlebbar“, sie sind für viele Menschen nicht einfach lesbar.

Die Hypothese ist, dass 3D-Simulationen oder virtuelle Realitäten die Hochwassergefahr besser begreifbar machen können.
Sie sollen dort eingesetzt werden, wo historische Fotos als Beweis und als Visualisierung der Gefahr fehlen.

Die Ziele dieses Projekts des Mobiliar Labs für Naturrisiken an der Universität Bern sind:

- die Entwicklung eines Prototyps zur Darstellung einer Hochwassersimulation in 3D unter Verwendung verschiedener Technologien
- darauf aufbauend eine Machbarkeitsanalyse für eine Anwendung in der Praxis

Die dafür notwendigen Daten und Simulationen sind vorhanden.

Prototypen

Im Projekt wurden zwei Prototypen basierend auf verschiedenen Technologien erstellt. Die Daten sind dieselben und die Funktion beider Prototypen ist sehr ähnlich. Die grössten Unterschiede sind technisch oder visuell andere Umsetzungen.

Web Framework

Mit WebGL[1] ist seit 2011 eine Technologie vorhanden, um im Web-Browser 3D Grafiken auf eine einfache Art darzustellen. Dabei kann der Browser auch auf Hardwarebeschleunigung zugreifen.
Darauf basiert auch Web Framework Cesium.js[2]. Dieses JavaScript Framework ist Open Source und kann durch die Apache 2.0 Lizenz auch kommerziell verwendet werden.
Cesium bietet bereits viele Funktionen, welche für GIS-Anwendungen sehr nützlich sind. Als Standard werden die Orthophotos von Bing[3] verwendet und ein eigenes freies 3D Geländemodell.
Der grosse Vorteil ist, dass es ohne Installation von Plug-Ins auf jedem modernen Web-Browser läuft. Daher wird JavaScript auch noch ein paar Jahre der gemeinsame Nenner von PC, Tablet, Smartphone usw. sein und viele Applikationen werden auf diesem Stack aufbauen.

[1] https://www.khronos.org/webgl/
[2] https://cesiumjs.org
[3] https://www.bing.com/maps

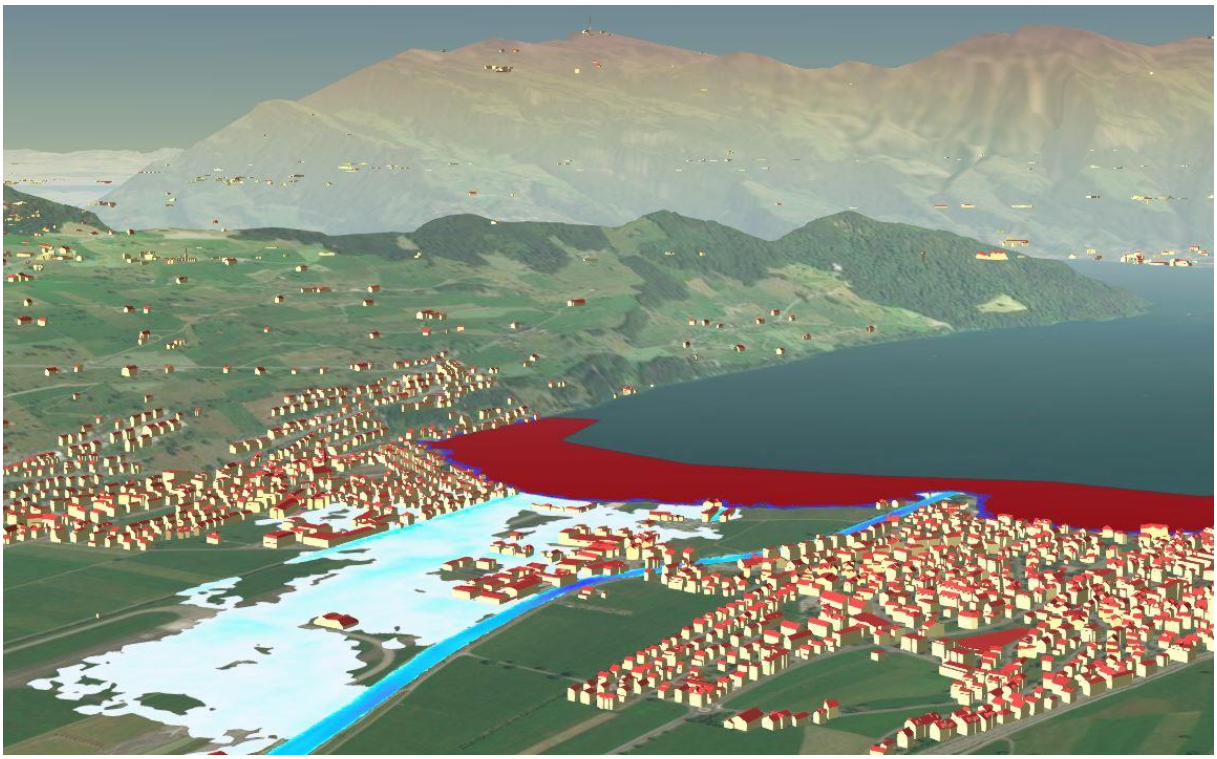

Abbildung 3 Screenshot Prototyp mit Web Framework

Game Engine

Die ursprüngliche Idee war die Verwendung von Unity[4] für eine 3D-Visualisierung. Unity ist eines der besten und populärsten Frameworks und hat eine grosse Community. Das Problem ist jedoch, dass Unity nicht Open Source ist. Die Kosten von $125 pro Monat wären verkraftbar, aber schliesslich der Grund, nach Alternativen Ausschau zu suchen.
Das andere populäre 3D-Framework ist die Unreal Engine[5]. Obwohl eher durch 3D-Shooter bekannt, liessen sich damit auch seriöse Applikationen erstellen. Stärken sind die hohe Rendering-Qualität und die gute Physik-Engine. Grosser Nachteil hier ist, dass Code in C++ geschrieben werden muss. Dies wäre für die Machbarkeitsstudie ein zu grosser Aufwand gewesen.
Die Wahl fiel dann auf die jMonkeyEngine[6]. Das ist eine Game Engine, welche vollständig in Java implementiert ist und intern OpenGL verwendet. Da der Quellcode offen ist, können auf einfache Art auch spezielle Anforderungen implementiert werden. Dies war in diesem Projekt sehr hilfreich, da die ganze Simulation aus Daten automatisch erstellt wird und nicht wie sonst üblich von Hand in einem Editor modelliert wird.
Obwohl Java in der Industrie als langsam und speicherhungrig gilt, haben die Entwickler diese Engine sehr performant implementiert.

[4] https://unity3d.com
[5] https://www.unrealengine.com
[6] http://jmonkeyengine.org

Abbildung 4 Screenshot Prototyp mit Game Engine

Implementation

Gelände

In einer 3D-Simulation ist das Gelände besonders wichtig. Dies ist schliesslich die weitere Dimension zur 2D-Karte.

Cesium verwendet standardmässig ein eigenes Geländemodell (STK World Terrain[7]). Dieses Modell deckt die ganze Welt in guter Qualität ab und kann frei verwendet werden.
Es ist auch einfach möglich, einen Service von swisstopo zu verwenden. Das Modell swissALTI3D[8] deckt die ganze Schweiz hoch aufgelöst ohne Bewuchs und Bebauung ab.

In jMonkeyEngine werden (wie auch in anderen Game Engines) Height Maps verwendet, um Gelände zu modellieren. Height Maps sind Graustufen-Bilder, wobei schwarz das Tal und weiss die Bergspitzen sind. Die Auflösung muss nicht besonders gut sein, denn Zwischenwerte werden interpoliert. In jMonkeyEngine muss das Gelände Quadratisch sein. Die Simulation verwendet ein Bild von der Grösse 512 x 512 Pixel. Das Gelände kann beliebig skaliert oder überhöht werden.
Die Height Maps wurden durch das swissALTI3D Modell erstellt. Das entstandene Geländemodell ist trotz Limitierung durch ein Bild genug genau.

Geländebilder

Das Gelände wird sehr viel besser erkennbar, wenn ein Orthophoto verwendet wird. Orthophotos sind Satellitenbilder, welche in guter Auflösung bestehen und über das Geländemodell gelegt werden können.

Cesium verwendet standardmässig die Karten von Bing. Es wird gebeten, ein eigener Bing API Key zu generieren und verwenden. Bing ist bis zu 125‘000 Abfragen pro Jahr kostenlos nutzbar.

[7] https://cesiumjs.org/data-and-assets/terrain/stk-world-terrain/
[8] https://shop.swisstopo.admin.ch/de/products/height_models/alti3D

Auf einfache Art kann auch ein Service von swisstopo verwendet werden. Die Qualität von swisstopo ist besser als Bing, jedoch werden die Bilder mit einem Wasserzeichen (Schweizerkreuz) versehen.

Mit jMonkeyEngine ist eine Verwendung von Orthophotos nicht vorgesehen. Die Geländetextur wird mit wiederholenden Texturen eingefärbt. Eine Orthophoto-Funktion könnte sicher implementiert werden, aber der Aufwand wäre vermutlich hoch.
Da nur ein begrenztes Gebiet (2km x 2km) modelliert ist, sind die Grenzen als klarer Abgrund erkennbar. Dahinter ist einfach nichts sichtbar.
Die Engine bietet nun die Möglichkeit, ein statisches Panoramabild als Landschaft und Himmel zu verwenden. Die Position ist fix, dreht aber mit der Kamera mit. So kann die Illusion erstellt werden, dass die Simulation im Gebiet Nidwalden stattfinden.
Das Panorama-Bild wurde in Google Earth erstellt und zusammengeschnitten. Das Format muss 1:2 sein, wobei der Horizont in der Mitte ist und die Ränder links und rechts nahtlos zusammengefügt werden können.

Hochwasser-Überlagerung

Die Hochwasserdaten sind das Resultat der Simulationsberechnung. Sie werden bisher durch Bilder visualisiert.
Der einfachste Weg für beide Prototypen ist, diese Bilder direkt auf das Gelände zu legen. So muss die Wassertiefe nicht als 3D-Objekt modelliert werden.
Dies ist jedoch nur eine Visualisierung. So klebt das Wasser auf dem Gelände und hat Fliesstiefe 0. Bei einer geringen Fliesstiefe bis 1m ist das nicht sichtbar, da die Kamera meist weiter entfernt ist. Bei grösseren Fliesstiefen stimmt diese Vereinfachung nicht mehr. Zum Beispiel das Aare-Hochwasser in der Matte Bern um 1480, als die Fliesstiefe etwa 3m betrug. Da wäre bei genauerer Betrachtung sichtbar, dass die Häuser nur am Boden im Wasser stehen und der Wasserstand nicht auf der korrekten Höhe steht.

Cesium ist sehr gut mit der Darstellung als Überlagerung von Bildern. Echte 3D-Objekte sind möglich, jedoch müsste die Wasserfläche als 3D-Tiles modelliert werden. Ein Versuch mit einfachen geometrischen Dreiecken funktionierte, aber die Performance war sehr schlecht.

In jMonkeyEngine könnte der Wasserstand als geometrische Figur modelliert werden. Das ist einfach möglich und kann schnell gezeichnet werden. Einzig die Wasserstruktur stellt ein Problem dar, da es nicht einfach ist, auf irregulären Dreiecken eine Textur korrekt anzuwenden. Dieses Problem mit den Texturkoordinaten ist jedoch lösbar.

Vierte Dimension

Als vierte Dimension wird meist die Zeit verwendet, und das ist auch hier so. Die Berechnung erfolgt in diskreten Zeitschritten. Die Berechnung verwendet 2 min oder 10 min Schritte zwischen zwei Bildern. So läuft die Simulation stark beschleunigt wie ein Film. Zu beachten ist jedoch, dass bei mehr als 2 Bildern pro Sekunde die Simulation ins Stocken kommt, da zu viele Daten nachgeladen werden müssen. In den folgenden fünf Beispielbildern sind Zeitschritte mit je 5 h Abstand dargestellt.
Da die Zeitschritte diskret sind, kann nicht beliebig fein aufgelöst werden, da die Bilder nicht existieren und auch nicht interpoliert werden.

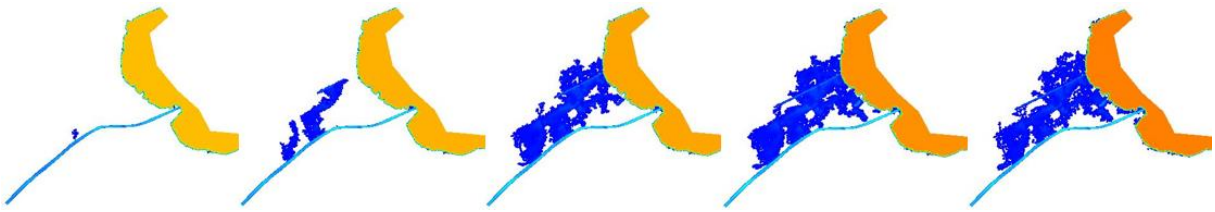

Abbildung 5 Zeitschritte der vierten Dimension

Datenaufbereitung

Da die verwendeten Datenformate nicht in üblicher 3D-Software benutzt werden, müssen sie entsprechend konvertiert werden.

Ausgangslage sind Mesh Files, welche automatisch oder teils auch von Hand erstellt werden. Diese Mesh Files werden zusammen mit weiteren Parametern in der Simulationssoftware Basement[9] durchgerechnet. Je nach Auflösung kann dies Stunden oder Tage dauern. Als Ausgabe werden diverse Werte wie Fliesstiefe, Geschwindigkeit oder Richtung in ein Solution File (.sol) geschrieben.

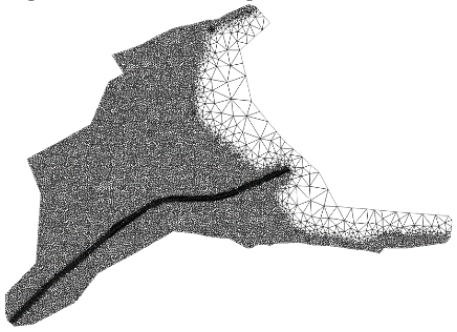

Abbildung 6 Mesh

Die Ausgabe kann auch mit dem QGis Plug-In Crayfish[10] visualisiert werden. Das funktioniert sehr gut und auch schnell. Crayfish rendert jeden Zeitschritt in einzelne Bilder und erstellt daraus ein Film. Leider werden die Einzelbilder danach wieder gelöscht. Durch eine kleine Änderung am Code können die Bilder jedoch in einem temporären Verzeichnis behalten und für spätere Visualisierungen verwendet werden. Am Anfang wurden diese Bilder verwendet und durch Georeferenzierung auf der Karte positioniert.

Abbildung 7 Render Crayfish

Jedoch war die Verwendung von Crayfish zu unhandlich, wodurch eine eigene Render-Applikation geschrieben wurde, die in etwa dasselbe macht. Jedoch können die Anforderungen exakt auf die Bedürfnisse der Simulation abgestimmt werden.

Für Cesium braucht es Graustufenbilder, welche mit einem Script in Kacheln geschnitten werden. Die Auflösung muss möglichst gut sein. Crayfish kann maximal 9999px rendern.
Der eigene Renderer könnte beliebige Auflösungen verarbeiten, verwendet aber 5000px, da sonst die Performance beim Kacheln erstellen stark einbricht und auch die zusätzliche Genauigkeit nichts bringt, wenn die Inputdaten nicht besser aufgelöst sind.

[9] BASEplane, http://www.basement.ethz.ch
[10] https://www.lutraconsulting.co.uk/products/crayfish/

Abbildung 8 Render Graustufen

Mit einem R-Script (Danke an die Hydrologiegruppe) werden die Bilder in Kacheln geschnitten. Die Zoomstufen 9 bis 16 werden durchgerechnet. Dies dauert auf einem üblichen Rechner etwas über einen Tag.

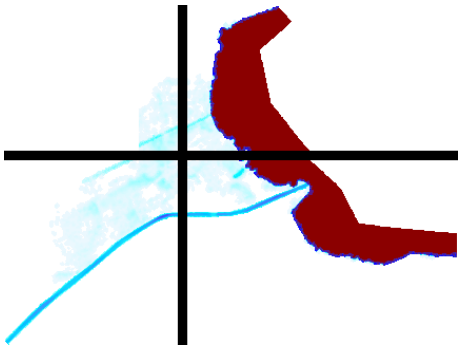

Abbildung 9 Zuschneiden Kacheln

Für die jMonkeyEngine müssen die Daten anders aussehen. Aktuell muss die Fläche quadratisch sein, daher schneidet der Renderer die Bilder auf ein Quadrat zu.

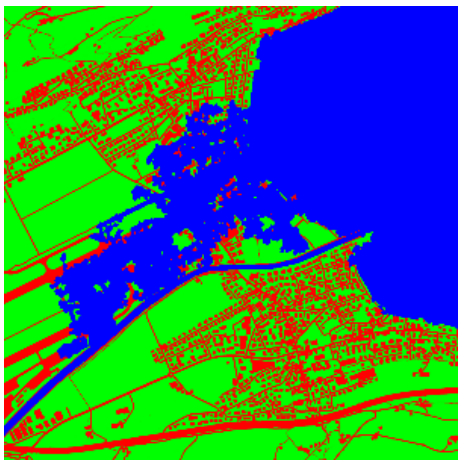

Abbildung 10 Texturkarte

Zudem müssen die Bilder speziell eingefärbt werden, damit die Engine später mittels „Texture Splatting“ die richtige Textur anwenden kann: Blaue Pixels werden mit Wassertextur gefärbt, grüne Pixels mit Grastextur, rote Pixels mit Asphalt. Eine Überblendung von Wasser und Gras Textur wäre auch möglich, um fliessende Grenzen zu ermöglichen.

Kacheln

Um grössere Gebiete detailliert darzustellen, kann nicht ein grosses Bild genommen und skaliert werden. Diese Bilder wären viel zu gross und unhandlich.
Also wird der gewünschte Kartenbereich in Kacheln aufgeteilt. Die Applikation lädt dabei immer nur die gerade sichtbaren Kacheln in den Speicher. Beim Zoomen wird die Zoomstufe der Kacheln erhöht,

indem jede Kachel durch vier genauere Subkacheln dargestellt wird. Beim Schieben der Karte werden die Nachbarskacheln geladen.

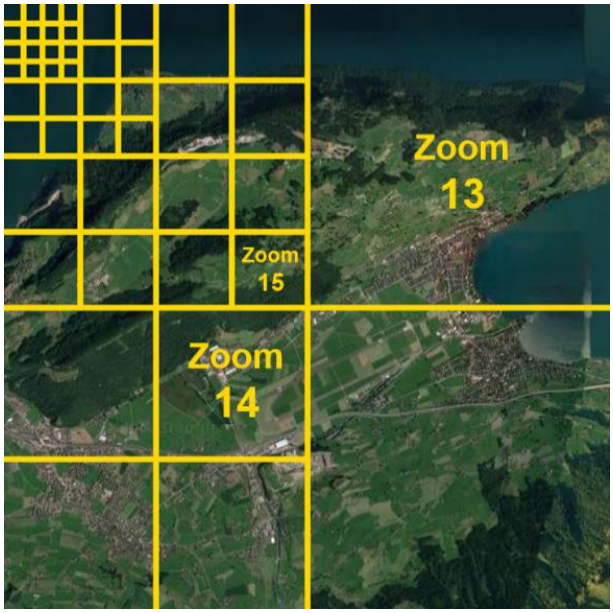

Abbildung 11 Zoomstufen Kacheln

Diese Technik wird in Webkarten wie Google Maps[11] oder OpenStreetMap[12] schon länger verwendet. Alternative Namen sind „Tiled web map“ oder „slippy maps“.
Je grösser jedoch die Zoomstufe ist, desto mehr Kacheln müssen bereitgestellt werden. Im Gebiet Nidwalden werden folgende Kacheln verwendet:

Zoom	Anzahl	Meter pro Pixel	Speicherplatz	Total Kacheln Ausschnitt
9	2	209	1.2kb	2x1
10	2	104	1.7kb	2x1
11	2	52	3.2kb	2x1
12	5	26	8.7kb	3x2
13	5	13	20kb	4x2
14	7	6.5	54kb	6x2
15	15	3.3	148kb	11x4
16	34	1.6	399kb	21x6

Tabelle 1 Zoomstufen

Leere Kacheln werden nicht generiert, wodurch die Anzahl Kacheln nicht der Anzahl Kacheln im Ausschnitt entspricht.
Die Tabelle 1 Zoomstufen ist für einen Zeitschritt. Bei 400 Zeitschritten ergibt es 28‘800 Bilder mit Speichergrösse von 248MB. Mit einer Zoomstufe mehr vervierfacht sich der Aufwand und Speicherbedarf beinahe.

[11] https://www.google.ch/maps
[12] https://www.openstreetmap.org

Cesium kann einen Web Map Tile Service[13] anbinden und diesen mit einer Zeitvariable parametrisieren. Dadurch kann das Bild der entsprechenden Zeit dynamisch geladen werden. Obwohl die Bilder gepuffert werden, sind beim Abspielen unscharfe Bereiche sichtbar und stören die Simulation. Das Problem ist bekannt und wird hoffentlich mit einem Update des Frameworks behoben.

3D-Gebäude

Swisstopo hat mit swissBUILDINGS3D 2.0[14] einen Datensatz mit allen Schweizer Gebäuden in 3D. Der frühere Datensatz 1.0 hatte jedes Gebäude nur als Flachdach. In der neuen Version ist die Dachform richtig abgebildet und meist korrekt. Die Fehlerquote liegt etwa bei 2%. Der Datensatz ist noch im Aufbau, sollte jedoch bis Mitte 2018 die ganze Schweiz komplett enthalten.

Für Cesium wird direkt der 3D Tiles[15] Service von Swisstopo verwendet. Dieser Service hat Swisstopo zusammen mit den Entwickler von Cesium erstellt und ist daher qualitativ sehr gut.

In jMonkeyEngine müssen die Gebäude zuerst konvertiert werden. Swisstopo bietet den Datensatz in diversen Formaten an, jedoch kann die Engine keines davon nativ lesen. Das Mobiliar Lab hat die Lizenz zur Verwendung im Format Multipatch Shapefile. Dieses kann mit FME[16] ins Format CityGML[17] konvertiert werden.
Da CityGML auf XML basiert, kann es einfach gelesen werden. Alle Flächen sind dabei schon trianguliert und können direkt als ein Custom Mesh[18] dargestellt werden. Für die korrekte Beleuchtung müssen jedoch noch die Normalvektoren der Flächen manuell berechnet werden. Für korrekte Texturen fehlen noch die Texturkoordinaten, auf welche verzichtet wurde. Darum haben die Häuser auch keine Textur, da sie über die Dreiecke falsch dargestellt würde. Die Simulation Nidwalden enthält 3960 Gebäude.

Vegetation

Swisstopo bietet mit swissTLM3D[19] auch einen Datensatz an, welcher Einzelbäume enthält. Die Bäume der Region wurden in ein GeoJson exportiert und in der jMonkeyEngine als Baum-Objekt dargestellt. Die Simulation Nidwalden enthält 1026 Bäume.

Virtual Reality (VR)

Die beiden Prototypen sind nicht wirklich 3D. Es sind nur 3D-Simulationen, welche auf dem Bildschirm als 2D wiedergegeben werden. Eine 3D-Tiefenschärfe ist nicht vorhanden.
Richtiges 3D ist nicht schwer zu erstellen, es müssen einfach je ein Bild pro Auge erzeugt werden, wobei die Kamera einen fixen Augenabstand hat. Aus den beiden leicht verschobenen Bildern kann das Gehirn ein 3D-Bild sehen. In der Natur funktioniert das genauso.

Cesium hat einen eingebauten VR-Modus. Dieser kann mit einem Knopf aktiviert werden und funktioniert am besten auf einem Handy im Panoramamodus.

[13] http://www.opengeospatial.org/standards/wmts
[14] https://shop.swisstopo.admin.ch/en/products/landscape/build3D2
[15] https://github.com/AnalyticalGraphicsInc/3d-tiles
[16] https://www.safe.com
[17] https://www.citygml.org
[18] https://jmonkeyengine.github.io/wiki/jme3/advanced/custom_meshes.html
[19] https://shop.swisstopo.admin.ch/de/products/landscape/tlm3D

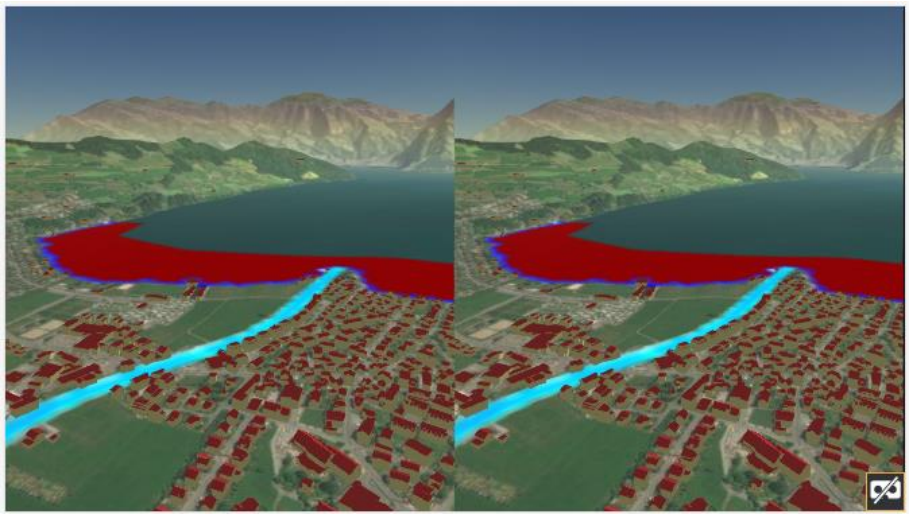

Abbildung 12 Virtual Reality Modus

Als zusätzliche Hardware ist nur ein Google Cardboard[20] nötig. In dieser Kartonbox wird das Handy eingesetzt. Mit zwei Linsen wird das Bild fürs Auge vergrössert. Durch die Bewegungssensoren kann die Kamera bewegt werden.

Abbildung 13 Google Cardboard

Besser ist jedoch zusätzliche Hardware mit einer dedizierten VR-Brille (zum Beispiel Oculus Rift[21] oder HTC Vive[22]). Diese kosten ab 500.- und haben hohe Anforderungen an die PC-Hardware. Als Resultat ist das VR-Erlebnis um weites besser.

Auch jMonkeyEngine hätte ein VR-Modul und könnte diverse 3D-Brillen nutzen, jedoch wurde dies bis jetzt nicht ausprobiert.

Augmented Reality (AR)

In Virtual Reality wird die gesamte Umgebung simuliert. Augmented Reality geht noch einen Schritt weiter und kombiniert Virtual Reality mit der Wirklichkeit. Dabei werden simulierte Teile über Kamerabilder gelegt oder direkt aufs Auge projiziert.

Im Jahr 2017 haben die beiden grossen Smartphone-Systemanbieter Apple (ARKit[23]) und Google (ARCore[24]) ein Framework veröffentlicht, um Augmented Reality auf Smartphones zu benutzen. Da

[20] https://vr.google.com/cardboard/
[21] https://www.oculus.com/rift/
[22] https://www.vive.com/de/
[23] https://developer.apple.com/arkit/
[24] https://developers.google.com/ar/

heute jeder ein leistungsfähiges Smartphone besitzt, ist nun auch die Software vorhanden, um auf einfache Art neue Visualisierungen zu erstellen.
Microsoft geht seit einigen Jahren eher den Weg von Mixed Reality. Ihre HoloLens[25] Hardware ist schon seit 2016 auf dem Markt, kostet aber $3000 bis $5000 und ist daher eher für den professionellen Markt geeignet.

Spielmodus

Bei der Verwendung einer Game Engine liegt es nahe, ein Spiel zu entwickeln. Zudem ist Gamification immer mehr ein Thema. Dabei werden komplexe Themen oder Lerninhalte als Spiel konzipiert und sind so für den Benutzer attraktiver und interessanter.
Als Spiel wurde ein kleines Rettungsboot erstellt, mit welchem die überschwemmten Häuser angefahren und so die Bewohner „gerettet" werden können. Das Haus färbt sich dabei grün. Das Rettungsboot fährt nur auf dem Wasser und lässt sich nur langsam manövrieren.

Folgende Eingaben werden verwendet:

- **Maus**: Kamera drehen
- **A**: Kamera links
- **D**: Kamera recht
- **W**: Kamera vorwärts
- **S**: Kamera rückwärts
- **Q**: Kamera steigen
- **Y**: Kamera sinken
- **Space**: Animation pausieren/abspielen
- **F**: Zeit schneller (wird mit Pause auf ursprüngliche Geschwindigkeit zurückgesetzt)
- **F5**: Statistik anzeigen
- **Esc**: Programm beenden

Steuerung im Spiel:

- **G**: Spiel starten (nur einmalig möglich)
- **U**: Vorwärts beschleunigen
- **J**: Bremsen, rückwärts
- **H**: nach links steuern
- **K**: nach rechts steuern

Farbcodierung der Gebäude:

- **Rot**: Haus steht im Wasser
- **Violett**: Haus steht erstmals im Wasser (pro Zeitschritt)
- **Gelb**: Haus stand schon im Wasser, aktuell jedoch nicht mehr
- **Grün**: Haus wurde „gerettet"

Als Indikator, ob ein Gebäude im Wasser steht, wird nur auf die Textur geschaut. Das Zentrum des Gebäudes muss auf einem blauen gefärbten Pixel stehen, dann ist es fürs Spiel im Wasser.
Da im Gebäudedatensatz nicht immer alle Gebäudeteile korrekt zusammengefasst sind, werden manchmal nur Teile davon eingefärbt und zu Beispiel die Garage als separates Gebäude betrachtet.

Das Rettungsboot ist als einfaches Fahrzeug mit vier unsichtbaren Rädern implementiert. Die Federung ist ganz weich und die Maximalgeschwindigkeit ist beschränkt. Daher ist das Fahrverhalten eher mit einem Buggy vergleichbar als mit einem Boot.
Eine Auswertung oder Ähnliches ist nicht implementiert. Die Kamera kann weiterhin mit der Maus gesteuert werden, dreht aber mit dem Boot mit.

[25] https://www.microsoft.com/de-ch/hololens

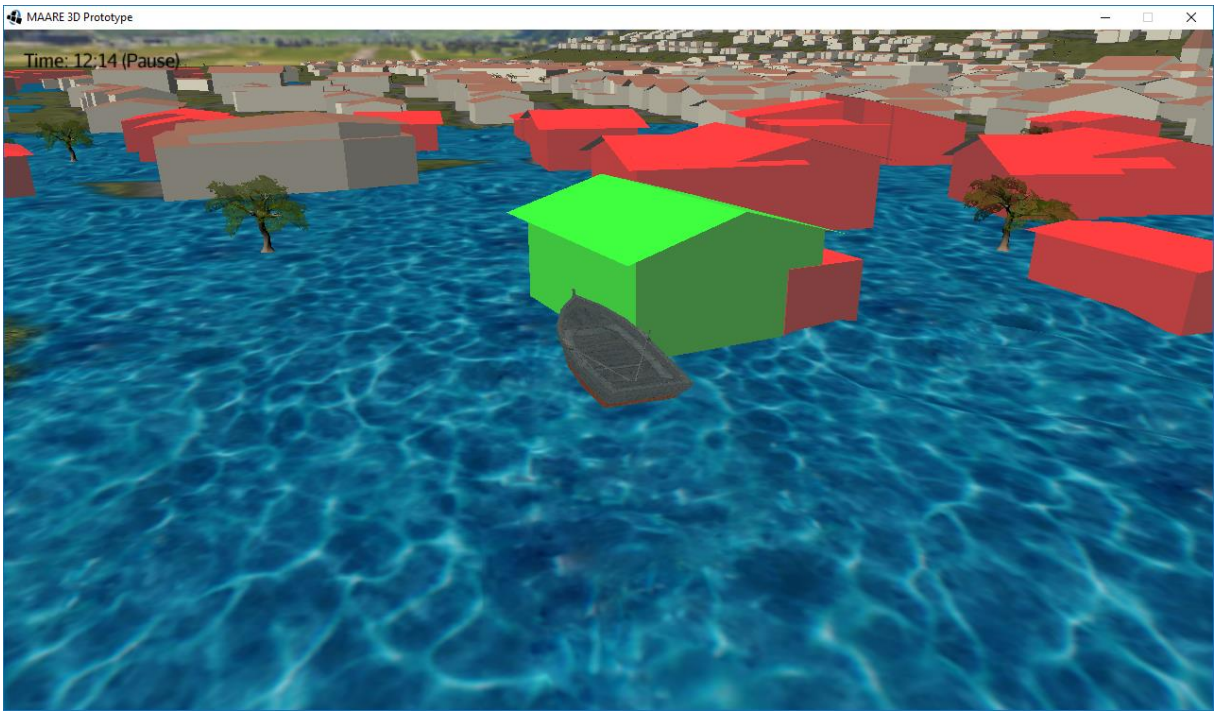

Abbildung 14 Screenshot Spielmodus

Quellcode

Der Quellcode wurde in drei verschiede Repositories aufgeteilt und mit Git[26] verwaltet. Als Programmiersprache sind primär Java und JavaScript verwendet worden.

Render-Projekt

Hilfsprogramm für die Generierung der Bilder aus den Basement Ausgabedateien. Die Applikation kann auch von der Kommandozeile mit entsprechenden Parametern gestartet werden:

```
java –jar javagis.jar \
c:/tmp/NW_2005_nds.2dm \
c:/tmp/NW_2005_2min_nds_depth.sol \
c:/tmp/see.geojson \
c:/tmp
```

Dabei müssen 4 Pfade als Parameter mitgegeben werden:

1. Pfad zur Mesh-Datei
2. Pfad zur Basement-Ausgabedatei
3. Pfad zur Seegrenze-Datei (GeoJson MultiPolygon)
4. Ausgabeverzeichnis (Unterverzeichnis wird automatisch erstellt)

Git Repo	http://git.giub.unibe.ch/mobilab_up20/javagis.git
Programmiersprache	Java 8
Anzahl Zeilen Code	etwa 600

Tabelle 2 Render Projekt

Game Engine-Projekt

Projekt mit der jMonkeyEngine. Enthält alle Daten offline und kann auch ohne Internetverbindung direkt gestartet werden.

Git Repo	http://git.giub.unibe.ch/mobilab_up20/jme.git

[26] https://git-scm.com

Programmiersprache	Java 8
Anzahl Zeilen Code	etwa 1500

Tabelle 3 Game Engine Projekt

Web Framework-Projekt

Webapplikation mit Cesium.js. Kann lokal gestartet oder auf einem Server deployed werden. Verwendet diverse Services von Swisstopo und lädt dabei viele Daten aus dem Internet. Zudem sind einige Hilfs-Skripte enthalten für die Erstellung und Bearbeitung der Kacheln.

Git Repo	http://git.giub.unibe.ch/mobilab_up20/cesium.git
Programmiersprache	JavaScript, R, Bash
Anzahl Zeilen Code	Etwa 60

Tabelle 4 Web Framework Projekt

Herausforderungen

In der 3D-Ansicht müssen die Daten immer in Echtzeit aufbereitet werden, da die Kameraführung nicht bekannt ist und vorausberechnet werden kann. Daher hat die Datenmenge einen direkten Einfluss auf die Performance der Visualisierung.
Die Datenmenge darf nicht zu gross sein wegen der Performance und nicht zu klein wegen der Qualität.

Ein weiterer Einfluss hat die Auflösung der Mesh-Modellierung. Im Mesh (Abbildung 15 Triangulierung Flusslauf) sind zum Beispiel der Flusslauf sehr detailliert (kleine Dreiecke), das umliegende Gebiet gröber (grössere Dreiecke). Für die Berechnung macht das Sinn, denn der Flusslauf muss sehr genau berechnet werden für gute Resultate. Die Umgebung ist aus Performance-Gründen weniger gut modelliert.
Für die Visualisierung ist es aber eher umgekehrt: Der Flusslauf führt immer Wasser und muss nicht fein modelliert sein. Interessant ist jedoch das Umland, welches weniger gut modelliert ist. Daher kann nicht beliebig reingezoomt werden ohne Qualitätsverlust.

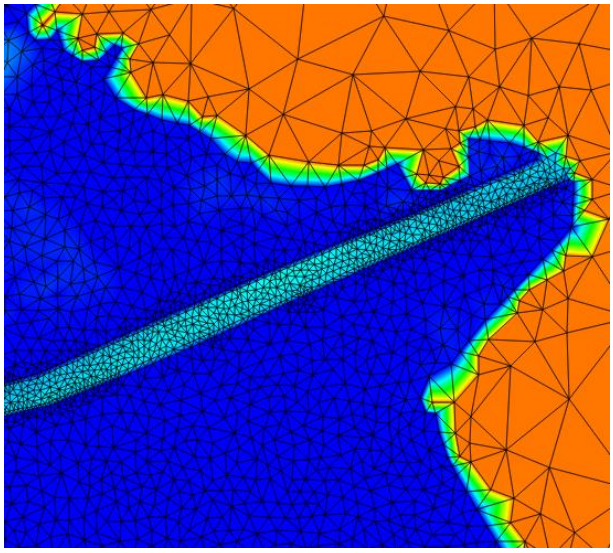

Abbildung 15 Triangulierung Flusslauf

Die Berechnung und Visualisierung erfolgt in spezieller GIS-Software, welche für Geo-Anwendungen konzipiert sind. Dabei sind auch Datenformate entstanden, die für diesen Zweck brauchbar sind. Viele Daten sind z.B. im ESRI Shapefile (.shp) Format, welches eigentlich ein natives Datenformat vom

Marktführer ESRI (ArcGIS[27]) ist. Weitere verwendete Formate sind Solution Files (.sol) oder 2D-Mesh Files (.2dm).
Ausserhalb der „GIS-Welt“ sind diese Formate jedoch meist unbekannt. Da sie teils auch noch proprietäre Binärformate sind, können sie durch Drittsoftware nur sehr schwer gelesen werden. QGis[28] und GDAL[29] können viele Formate umwandeln.

Für die verschiedenen Daten sind Kacheln gut geeignet, da sie skalieren und gute Performance bringen. Ein Nachteil ist jedoch die Datenmenge: schon ein kleines Gebiet braucht ein paar hundert MB an Platz. Die Berechnung der Kacheln dauert mehre Stunden oder Tage. Grössere Gebiete wie die ganze Schweiz oder Europa brauchen daher Faktoren mehr Platz, wobei durchaus die Festplattengrenze erreicht werden kann.

Für die Schweiz liefert das Bundesamt für Landestopografie swisstopo[30] die besten Daten. Am Mobiliar Lab für Naturrisiken sind auch Lizenzen vorhanden, um alle möglichen Arten von Daten für die Forschung zu verwenden.
Sobald jedoch eine Visualisierung im Internet veröffentlicht werden will, braucht es intensive Abklärungen, ob die Daten nun verwendet werden können oder ob das schon kommerzielle Nutzung ist.

Vergleich Technologien

Jede Technologie hat ihre Vor- und Nachteile. In der Tabelle 5 ist dieses für die beiden verwendeten Technologien aufgestellt. Der subjektive Gewinner der Zeile ist jeweils grün hinterlegt.

	Web Framework	**Game Engine**
Ziel-Plattform	Web Browser (Desktop, Smartphone, Tablet)	Primär Desktop, muss installiert werden
Aufwand	Gering, Framework kann viel	Gross, Welt muss modelliert werden
Freiheit, Möglichkeiten	Gering, ans Framework gebunden	Gross, fast alles ist möglich
Grösse Modell	Beliebig gross, ganze Welt möglich	Beschränkt durch Modellierung
Aussehen	Gelände, Häuser, Fototexturen	Gelände, Häuser
Konnektivität	Online, lädt viele Daten aus Internet	Offline, alles lokal vorhanden
Performance	Gut in modernen Browsern	Sehr gut dank OpenGL mit Grafikprozessor Unterstützung

Tabelle 5 Vergleich Technologien

[27] https://www.arcgis.com
[28] https://www.qgis.org
[29] http://www.gdal.org
[30] https://www.swisstopo.admin.ch

Empfehlung

Für zukünftige GIS-Anwendungen in 3D empfiehlt sich das Web Framework. Die Gründe dafür sind:

- Läuft in jedem Browser ohne zusätzliche Software
- Kann viele Funktionen schon ohne Aufwand
- JavaScript ist leicht verständlich und braucht wenig Code
- Besser geeignet für grössere Regionen

Obwohl eine Game Engine mehr Funktionen und Freiheiten bietet, ist es vermutlich den Aufwand nur begrenzt wert. Für Spezialfälle oder kleinere Gebiete könnte es aber trotzdem eine Alternative darstellen.

Dank

Der Autor dankt Andreas Zischg und Markus Mosimann vom Mobiliar Lab für Naturrisiken für die gute Betreuung und dafür, dass sie die nötigen Daten zur Verfügung gestellt haben.