WIPP360

1. Idee/Ziel

Da ich neben normalen Fotos auch gerne 360° Fotos mache und bereits eine Fotoplattform namens Wipptal Backgrounds erstellt habe, möchte ich eine unabhängige Plattform eigens für 360° Fotos namens WIPP360 erstellen.

1.1 Wahrscheinlich benötigte Ressourcen

- HTML und CSS
- Webserver
- Datenbank ?
- 360° Fotos

2. Recherche

Nach einer längeren Recherche im Internet nach Open-Source Viewern bin ich auf **Google VR View for the Web** gestoßen, dies gibt es auf GitHub (Google Archive/VR View) kostenlos zum Herunterladen.

Da es nach einigem Herumprobieren einfach nicht funktioniert hat, habe ich mir ein YouTube Video angeschaut und entdeckt, dass ein kompletter Ordner fehlt. Im „Build“ Ordner wären einige wichtige Skriptdateien gewesen, ohne die nichts läuft.

Ich wollte schon nach einer anderen Lösung suchen, als ich auf eine weitere GitHub Seite (Theata360Developers/VR View) von einem anderen Unternehmen gestoßen bin, wo der vollständige Code und auch der fehlende Ordner enthalten war.

Anschließend habe ich den Code auf meinen Webserver hochgeladen, um diesen nutzen zu können. Der sogenannte Code, also das ganze Paket besteht aus ein paar HTML-Dateien, damit es im Browser angezeigt werden kann, aber hauptsächlich aus vielen Java-Script-Dateien.

build	26.08.2016 01:11	Dateiordner	
examples	26.08.2016 01:11	Dateiordner	
images	26.08.2016 01:11	Dateiordner	
img	08.04.2020 18:06	Dateiordner	
node_modules	26.08.2016 01:11	Dateiordner	
scripts	26.08.2016 01:11	Dateiordner	
src	26.08.2016 01:11	Dateiordner	
CONTRIBUTING	26.08.2016 01:11	Datei	1 KB
COPYING	26.08.2016 01:11	Datei	12 KB
index.html	05.04.2020 19:51	HTML-Datei	1 KB
package.json	26.08.2016 01:11	JSON-Datei	2 KB
README.md	26.08.2016 01:11	MD-Datei	2 KB
style.css	26.08.2016 01:11	Kaskadierendes St...	2 KB
video-sample.html	26.08.2016 01:11	HTML-Datei	1 KB

Abbildung 1|Struktur des Hauptordners

device-motion-sender.min.js	26.08.2016 01:11	JavaSkriptdatei	2 KB
vrview.js	26.08.2016 01:11	JavaSkriptdatei	1 100 KB
vrview.min.js	26.08.2016 01:11	JavaSkriptdatei	729 KB
vrview-analytics.js	26.08.2016 01:11	JavaSkriptdatei	1 102 KB
vrview-analytics.min.js	26.08.2016 01:11	JavaSkriptdatei	731 KB

Abbildung 2|Dateien im build Ordner

3. Programmierung

Am Anfang habe ich eine ganz simple HTML Seite programmiert und diese direkt auf meinem Webserver ausprobiert. Der Code für das Anzeigen eines 360° Fotos schaut so aus:

```
<iframe width="100%" height="700px" allowfullscreen frameborder="0"
src="index.html?image=img/bild.jpg&is_stereo=false"></iframe>
```

Danach habe ich mithilfe von W3CSS eine schönere Webseite programmiert, das derzeitige Ergebnis ist auf elektronikr.de/WIPP360 abrufbar. Was W3CSS genau ist steht im nächsten Absatz.

3.1 Programmieren mit W3CSS

W3CSS ist eine einfache und schöne Art HTML zu programmieren. Es ist ein modernes CSS Framework, das ganz einfach importiert werden kann, und mit vielen Parametern, schöne Design-Elemente erzeugt. Mithilfe von diesem Framework kann man auch ganz einfach Webseiten für Mobilgeräte optimieren.

Um es nutzen zu können, muss zuerst das Stylesheet im Head-Bereich des HTML-Codes importiert werden. Das geht ganz einfach, wie unten gezeigt mithilfe der URL, oder man lädt sich die CSS Datei herunter und hostet sie selbst.

```
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css">
```

Weiters müssen eventuell verschiedene Schriftarten importiert werden, die häufigste Methode ist via Google Fonts API. Die Schriftarten werden ebenfalls im Head-Bereich importiert.

```
<link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Roboto">
```

Um die Schriftarten nutzen zu können muss dies erst in einem weiteren Stylesheet oder direkt im HTML-Head-Bereich mit folgenden Zeilen deklariert werden.

```
<style>
body,h1,h2,h3,h4,h5,h6 {font-family: "Roboto", sans-serif}
.w3-bar,h1,button {font-family: "Fjalla One", sans-serif}
.fa-anchor,.fa-coffee {font-size:200px}
</style>
```

W3CSS schaut einerseits super schön aus und ist auch sehr gut für Mobilgeräte optimierbar, die Navigationsleiste kann so bei großen Bildschirmen über die ganze Breite gezogen sein und bei kleinen Bildschirmen ist sie eingeklappt, dass kann über hide-Befehle erzielt werden.

```
 <!-- Navbar -->
div class="w3-top">
 <div class="w3-bar w3-yellow w3-card w3-left-align w3-large">
  <a class="w3-bar-item w3-button w3-hide-medium w3-hide-large w3-right w3-padding-large w3-hover-white w3-large
w3-yellow" href="javascript:void(0);" onclick="myFunction()" title="Toggle Navigation Menu"><i class="fa fa-
bars"></i></a>
  <a href="index.html" class="w3-bar-item w3-button w3-hide-small w3-padding-large w3-hover-white">Start</a>
  <a href="#" class="w3-bar-item w3-button w3-padding-large w3-white">Galerie</a>
```

```
  <a href="mailto:kontakt@wipptal-backgrounds.com" class="w3-bar-item w3-button w3-hide-small w3-padding-large w3-hover-white">Kontakt</a>
 </div>

 <!-- Navbar on small screens -->
 <div id="navDemo" class="w3-bar-block w3-white w3-hide w3-hide-large w3-hide-medium w3-large">
  <a href="index.html" class="w3-bar-item w3-button w3-padding-large">Start</a>
  <a href="#" class="w3-bar-item w3-button w3-padding-large">Galerie</a>
  <a href="mailto:kontakt@wipptal-backgrounds.com" class="w3-bar-item w3-button w3-padding-large">Kontakt</a>
 </div>
</div>
```

W3CSS ermöglicht es, ganz einfach bei div-Elementen durch verschiedene Klassen, verschiedene Eigenschaften zuzuweisen, alle verfügbaren Befehle findet man auf der Homepage. Das ist ein Beispiel für einen Footer. w3-container legt ein Feld fest, w3-padding-32 ist der Abstand, w3-center legt fest, dass der Inhalt zentriert ist, w3-yellow legt die Farbe gelb des Felds fest und w3-mobile legt fest, dass auf Mobilgeräten die ganze Breite ausgenutzt wird.

```
<footer class="w3-container w3-padding-32 w3-center w3-yellow w3-mobile" style="max-width=100%">
 <a href="https://www.wipptal-backgrounds.com">
  <img class="w3-image w3-mobile" src="https://images.elektronikr.de/Wipptal_Backgrounds_Logo.png" alt="" width="30%" height="auto">
 </a>
 <p>Powered by <a href="https://www.elektronikr.de" target="_blank">Elektronikr</a></p>
</footer>
```

4. Fazit

WIPP360 funktioniert ausgezeichnet und ich habe gelernt, wie man ganz einfach mit W3CSS eine schöne, minimalistische Webseite programmieren kann.

Abbildung 3| WIPP360 Galerie Seite

X-Viewer

1. Idee

Auf Basis vom VR View Paket verwendet beim vorherigen Projekt WIPP360 soll ein Python Code programmiert werden, mit dem man 360° Fotos auf dem Computer öffnen kann, da es nicht sehr viel Freeware in dem Bereich gibt.

1.1 Wahrscheinlich benötigte Ressourcen

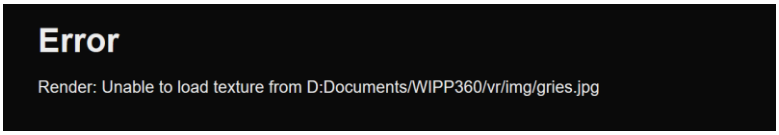
- VR View
- Python

2. Programmierung

Grundidee: Die VR View Umgebung kann man ja mit einer URL, oder in dem Fall lokal im Browser mit dem Dateipfad öffnen. So war ist die Idee, mit einem Python Programm einfach den Pfad von VR View und den Pfad vom Bild zu kombinieren und im Browser aufzurufen. Zum Beispiel:

file:\\\ D:\Documents\X Viewer\vr\index.html?image= D:\Images\bild.jpg

Allerdings hat das nicht ganz wie vorgestellt funktioniert, denn die vielen Java-Script Dateien können nicht einfach auf lokal gespeicherte Dateien zugreifen.



Lösung: Verwendung von der Python http-Server Bibliothek, um für eine kurze Zeit einen lokalen Server zu erstellen, wo die VR View Umgebung geladen und geöffnet werden kann.

2.1 Grundcode

Auf Basis der obigen Überlegungen ist folgender funktionsfähiger Code als Grundgerüst entstanden.

```
#Bibliotheken-----------
import http.server
import socketserver
import shutil
import webbrowser
#------------------------

#Deklarationen-----------------------------------------------------------------
PORT = 5555
Handler = http.server.SimpleHTTPRequestHandler
original = ''
target = r'cache\img.jpg'
url = 'http://127.0.0.1:5555/vr/index.html?image=/cache/img.jpg&is_stereo=false'
#------------------------------------------------------------------------------
#Hauptprogramm-----------------------------------------------------------------
original = input("Bitte geben Sie den Pfad des Bildes an.")
shutil.copyfile(original, target)
webbrowser.open_new_tab(url)

#Starte lokalen Server---------------------------------
with socketserver.TCPServer(("", PORT), Handler) as httpd:
    print("serving at port", PORT)
    httpd.serve_forever()
#-----------------------------------------------------
#Ende--------------------------------------------------------------------------
```

Als erstes wird vom Benutzer der Pfad der zu anzeigenden Datei abgefragt, diese Bilddatei wird anschließend in den **cache-Ordner** des Programms kopiert und die lokale URL mit Port 5555 und dem Speicherort im Cache mithilfe von der Bibliothek **webbrowser** geöffnet.

Anschließend startet das bereits standardmäßig installierte Modul **http.server** einen lokalen Server mit **Port 5555**. Dieser ist entweder über die Loopback-Adresse (**127.0.0.1**), oder der IPv4 bzw. IPv6 im internen Netz erreichbar.

2.2 Verbesserter Code

2.2.1 Beschränkte Zeit

Die erste und vorerst wichtigste Verbesserung soll sein, dass der http-Server nicht so lange läuft, bis das Programm gestoppt wird, sondern nach dem erfolgreichen Laden automatisch beendet wird.

Das war aber überhaupt nicht so einfach, wie gedacht, denn der Server wird mit httpd.serve_forever() gestartet und wie der Name schon sagt, läuft er, bis das Programm unterbrochen wird. In der Dokumentation von **socketserver** und **http.server** nachgeschaut, hat es keine Möglichkeit gegeben, den Server nach einer gewissen Zeit oder Ereignis automatisch auszuschalten. Nach langem Recherchieren und Nachdenken bin ich auf die Idee gekommen, Threading zu benutzen.

Mithilfe von der **Threading Bibliothek** kann man zwei Befehle sozusagen gleichzeitig ausführen, so habe ich eine Funktion erstellt, die den Server mit dem Befehl **shutdown()** stoppt, und diese mithilfe des **Threading-Timers** nach 10 Sekunden aufgerufen wir, in der Zwischenzeit läuft **serve_forever()** bis das shutdown-Event eingetreten ist.

```
#Funktionen------------------
def stop():
    httpd.server_close()
    httpd.shutdown()
#----------------------------
#Starte lokalen Server-------------------------------------
with socketserver.TCPServer(("", PORT), Handler) as httpd:
    print("serving at port", PORT)
    t = threading.Timer(10.0, stop)
    t.start()
    httpd.serve_forever(poll_interval=1)
#----------------------------------------------------------
```

2.2.2 Unterstützung von Videos

Für die Unterstützung zur Wiedergabe von Videos muss die Eingabe des Benutzers lediglich überprüft werden, ob sie den Text „jpg“ enthält, ist das nicht der Fall, wird eine veränderte URL für Videos aufgerufen.

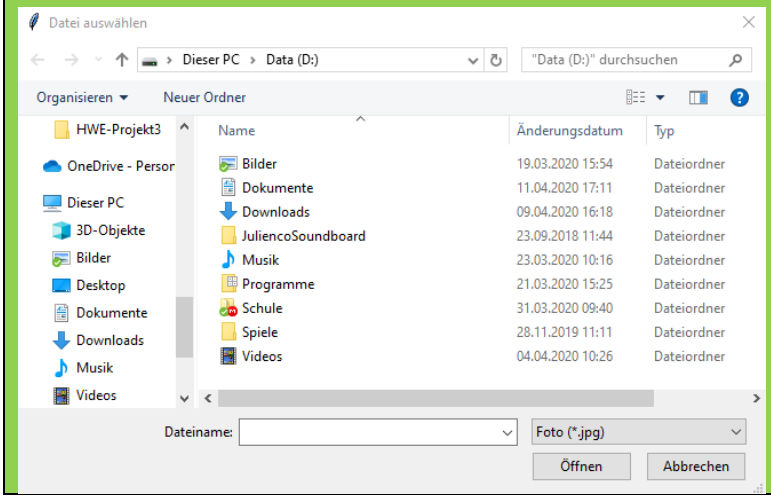
```
target_i = r'cache\img.jpg'
target_v = r'cache\vid.mp4'
url_i = 'http://127.0.0.1:5555/vr/index.html?image=/cache/img.jpg&is_stereo=false'
url_v = 'http://127.0.0.1:5555/vr/index.html?video=/cache/vid.mp4&is_stereo=false'
```

```
if original.find("jpg") != -1:
    shutil.copyfile(original, target_i)
    webbrowser.open_new_tab(url_i)
else:
    shutil.copyfile(original, target_v)
    webbrowser.open_new_tab(url_v)
```

2.2.3 GUI

Für die Bereitstellung einer grafischen Benutzeroberfläche (GUI) wird das bereits integrierte Modul **Tkinter** verwendet. Dieses muss als erstes importiert werden. Als erstes wird ein Dialog zum Datei öffnen mithilfe von **Tkinter Filedialog** erstellt, das geschieht mittels nur drei Zeilen Code.

```
root = Tk()
root.filename =  filedialog.askopenfilename(initialdir = "/",title = "Datei auswählen",
                                            filetypes = (("Foto","*.jpg"),("Video","*.mp4*")))
original = root.filename
```



Im Dialog kann man entweder ein Foto oder eben Video auswählen.

2.2.4 Ausführbare Datei

Bevor die GUI verbessert wird, möchte ich zuerst probieren, aus dem Python Skript und dem VR View Paket eine ausführbare EXE-Datei zu erstellen. Das mache ich mit **pyinstaller**, das als erstes mit folgendem Befehl installiert werden muss.

```
pip install pyinstaller
```

```
PS D:\Documents\X-Viewer> pyinstaller "xviewer.py" --onefile --windowed --name=X-Viewer --icon=icon.ico
122 INFO: PyInstaller: 3.6
123 INFO: Python: 3.7.7
123 INFO: Platform: Windows-10-10.0.18362-SP0
125 INFO: wrote D:\Documents\X-Viewer\X-Viewer.spec
126 INFO: UPX is not available.
131 INFO: Extending PYTHONPATH with paths
['D:\\Documents\\X-Viewer', 'D:\\Documents\\X-Viewer']
131 INFO: checking Analysis
132 INFO: Building Analysis because Analysis-00.toc is non existent
132 INFO: Initializing module dependency graph...
138 INFO: Caching module graph hooks...
151 INFO: Analyzing base_library.zip ...
3420 INFO: Processing pre-find module path hook   distutils
3420 INFO: distutils: retargeting to non-venv dir 'c:\\python\\python37\\lib'
5448 INFO: Caching module dependency graph...
5647 INFO: running Analysis Analysis-00.toc
5650 INFO: Adding Microsoft.Windows.Common-Controls to dependent assemblies of final executable
  required by c:\python\python37\python.exe
5790 INFO: Analyzing D:\Documents\X-Viewer\xviewer.py
6023 INFO: Processing module hooks...
6023 INFO: Loading module hook "hook-distutils.py"...
6026 INFO: Loading module hook "hook-encodings.py"...
6184 INFO: Loading module hook "hook-pydoc.py"...
6185 INFO: Loading module hook "hook-sysconfig.py"...
6189 INFO: Loading module hook "hook-xml.py"...
6538 INFO: Loading module hook "hook-_tkinter.py"...
6815 INFO: checking Tree
6816 INFO: Building Tree because Tree-00.toc is non existent
6821 INFO: Building Tree Tree-00.toc
6935 INFO: checking Tree
6936 INFO: Building Tree because Tree-01.toc is non existent
6938 INFO: Building Tree Tree-01.toc
6981 INFO: Looking for ctypes DLLs
6982 INFO: Analyzing run-time hooks ...
6988 INFO: Including run-time hook 'pyi_rth__tkinter.py'
6997 INFO: Looking for dynamic libraries
7528 INFO: Looking for eggs
7528 INFO: Using Python library c:\python\python37\python37.dll
7529 INFO: Found binding redirects:
[]
7552 INFO: Warnings written to D:\Documents\X-Viewer\build\X-Viewer\warn-X-Viewer.txt
7622 INFO: Graph cross-reference written to D:\Documents\X-Viewer\build\X-Viewer\xref-X-Viewer.html
7676 INFO: checking PYZ
7676 INFO: Building PYZ because PYZ-00.toc is non existent
7677 INFO: Building PYZ (ZlibArchive) D:\Documents\X-Viewer\build\X-Viewer\PYZ-00.pyz
8526 INFO: Building PYZ (ZlibArchive) D:\Documents\X-Viewer\build\X-Viewer\PYZ-00.pyz completed successfully.
8542 INFO: checking PKG
8542 INFO: Building PKG because PKG-00.toc is non existent
8543 INFO: Building PKG (CArchive) PKG-00.pkg
13668 INFO: Building PKG (CArchive) PKG-00.pkg completed successfully.
13706 INFO: Bootloader c:\python\python37\lib\site-packages\PyInstaller\bootloader\Windows-64bit\runw.exe
13707 INFO: checking EXE
13708 INFO: Building EXE because EXE-00.toc is non existent
13713 INFO: Building EXE from EXE-00.toc
13752 INFO: Copying icons from ['icon.ico']
13754 INFO: Writing RT_GROUP_ICON 0 resource with 20 bytes
13754 INFO: Writing RT_ICON 1 resource with 24783 bytes
13767 INFO: Updating manifest in D:\Documents\X-Viewer\build\X-Viewer\runw.exe.opy74ua7
13769 INFO: Updating resource type 24 name 1 language 0
13778 INFO: Appending archive to EXE D:\Documents\X-Viewer\dist\X-Viewer.exe
13799 INFO: Building EXE from EXE-00.toc completed successfully.
```

Nach dem Erhalt der EXE-Datei wird diese mit den Ordnern vr und cache in ein ZIP-Archiv gepackt und mit dem Programm NSIS zu einer installierbaren EXE gepakt.

2.2.5 Applikation auf Linux

Um eine Applikation für Linux zu erstellen, muss (gleich wie in Punkt 2.2.4) PyInstaller installiert werden und auf einem Linux-Gerät (in meinem Fall eine virtuelle Maschine mit Ubuntu) generiert werden.

Anschließend wird das Programm mit folgendem Befehl geöffnet:

```
sieller@sieller-virtual-machine:~/Dokumente/X-Viewer$ ./X-Viewer
```

Beim ersten Versuch, das Programm zu öffnen ist ein Fehler aufgetreten, dass Tkinter nicht installiert sei, es musste also erst nachinstalliert werden.

```
sudo apt-get install python3-tk
sudo apt-get upgrade
reboot -n
```

Die Entwicklung des Programms für Linux wurde vorübergehend pausiert, da es einfach nicht funktioniert hat.

2.3 Geplante Verbesserungen

- Öffnen von 360° Fotos und Videos in Windows direkt im Context-Menü
- Verbesserung der GUI

2.3.1 Context-Menü Eintrag

Den Code für einen Eintrag im Windows-Context-Menü habe ich schon, er muss in einer zukünftigen Version nur noch eingebunden werden.

```python
import winreg as _winreg

def define_action_on(filetype, registry_title, command, title=None):
    reg = _winreg.OpenKey(_winreg.HKEY_CURRENT_USER, "Software\\Classes", 0, _winreg.KEY_SET_VALUE)
    k1 = _winreg.CreateKey(reg, filetype)
    k2 = _winreg.CreateKey(k1, "shell")
    k3 = _winreg.CreateKey(k2, registry_title)
    k4 = _winreg.CreateKey(k3, "command")
    if title != None:
        _winreg.SetValueEx(k3, None, 0, _winreg.REG_SZ, title)
    _winreg.SetValueEx(k4, None, 0, _winreg.REG_SZ, command)
    _winreg.CloseKey(k3)
    _winreg.CloseKey(k2)
    _winreg.CloseKey(k1)
    _winreg.CloseKey(reg)

define_action_on(".jpg", "X-Viewer", "\"D:\\Programme\\Thonny\\thonny.exe\" \"D:\\Documents\\X-Viewer\\server3.py\" \"%1\"", title="Öffnen mit X-Viewer")
```