Informatik
Hauptcampus

H O C H
S C H U L E
T R I E R

Mosaic Generator Project

Bearbeiter: Mahmoud Al Jarad
Matrikelnr: 975671
Betreuer: Prof. Dr. Christof Rezk-Salama

Ausarbeitung zur Vorlesung Tool- and Pluginprogrammierung

Trier, den 31.08.2022

Inhaltsverzeichnis

1

Einleitung

Mosaik ist eine besondere Art, ein Bild zu machen. Werden dazu kleine Kacheln in verschiedenen Farben genommen. Die Kacheln so platziert werden können, dass die Farben ein Bild ergeben. Dann sorgt man in der Regel dafür, dass die Fliesen immer an Ort und Stelle bleiben, indem man sie zum Beispiel verklebt.
Es gibt eine andere Möglichkeit, ein Mosaik zu machen. Die Fliesen werden auf ein festes Stück Pappe gelegt, wo sie geklebt werden. Das resultierende Bild ist eigentlich ein Spiegelbild. Anschließend wird das gesamte Bild mit der freien Seite an der Wand befestigt. Die Kiste wird dann entfernt. Das hat einen großen Vorteil: die Fliesen müssen nicht vor Ort zusammensetzt werden.
Schließlich werden die Fugen zwischen den Steinen mit Gips oder einer ähnlichen Masse gefüllt. Anstelle von Steinen können Sie auch andere Materialien verwenden, zum Beispiel Glas. Auch Windows kann daraus erstellt werden. Auch Keramik, Ton, Kunststoff und vieles mehr werden heute verwendet. Mosaike sind seit der Antike bekannt. Viele andere Bilder aus dieser Zeit sind nicht mehr erhalten. Andererseits können Mosaike auf dem Boden eines alten Hauses platziert worden sein. Die Abbildungen 1.1 und 1.2 zeigen zwei Beispiele von Mosaik .

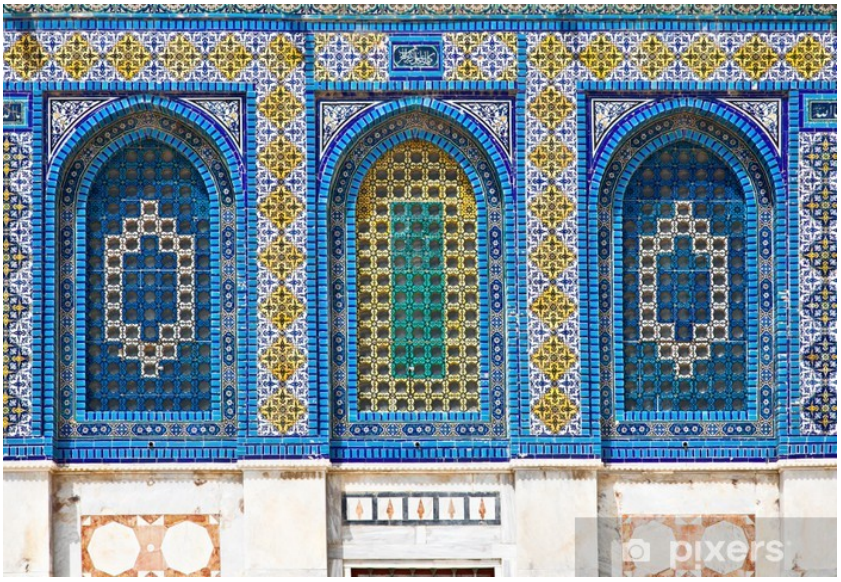


Abb. 1.1. Islamisches Muster, Mosaik auf Moschee.

Abb. 1.2. Mosaik an einer Wand.

2

Mosaik-Generator Programm

Dem modernen Fotomosaik liegt eine einfache Idee zugrunde: ein großes Bild aus vielen kleinen bunten Bildern. Mit einem Unterschied zum „klassischen“ Mosaik – aus „Mosaiksteinen“ – haben auch die Fotos jeweils ein eigenes Bild.
Auf diese Weise hat das Bild zwei Dimensionen. Das Hauptbild, das dem Mosaik seine Gesamtwirkung verleiht, und die vielen Einzelbilder aus einem Fotoverzeichnis entsteht ein Foto.

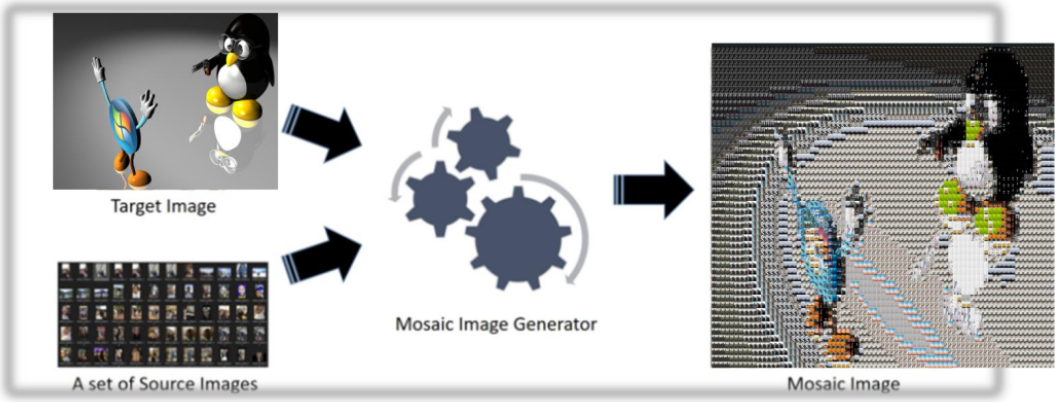


Abb. 2.1. Mosaik-Generator Input/Output.

2.1 Aufführung des Programms

Das Programm funktioniert in zwei Arten:
Erste Art wird durch Konsole mit dem folgenden Befehl und wird das Programm ohne GUIFenster ausgeführt:

```
1  python run.py -i <t_image> -o <output> -t <tiles_directory> -x <x> -y <y>
```

wobei t_image ist das Bild, das ein Mosaik gemacht wird, output ist das Mosaik vom t_image, tiles_directory ist der Verzeichnis von Bildern, die als Kacheln benutzt werden, und x bzw. y sind die Größe der Kacheln im Mosaik-Bild.

Die andere Art ist ohne Argumente:
Entweder anhands des Python-Kompilers oder mit dem folgenden Befehl durch den Terminal.

```
1    python run.py
```

In diesem Fall wird ein GUI-Fenster mit Eingabeinformationen gezeigt.
Hinweis: es muss die Bibliotheken „numpy, scipy, pillow und glob“ und für das Fenster GUI „PySimpleGUI“ installiert. Siehe die folgenden Befehle.

```
1  pip3 install numpy
2  pip3 install scipy
3  pip3 install pillow
4  pip3 install glob
5  pip3 install PySimpleGUI
```

Die Abbildung 2.2 zeigt das Fenster des Programms nach einer der zweite Ausführungsarten vor der Implementierung der Eingaben. Wenn man Run-Taste ohne input-Eingabe oder „View Result“-Taste vor dem Ende der Mosaikbearbeitung klickt, dann wird ein kleines Error-Fenster ausgegeben. Das Klicken von „View Result“-Taste zeigt das aktuelle Mosaik Image. Finishd-Progress ist rot vor dem Start und wird nach der Ausführung mit dunkel Grau ausgefüllt. Das hat die Folge „Operation erledigt“. Alle Eingaben kann man beliebig ausfällen.

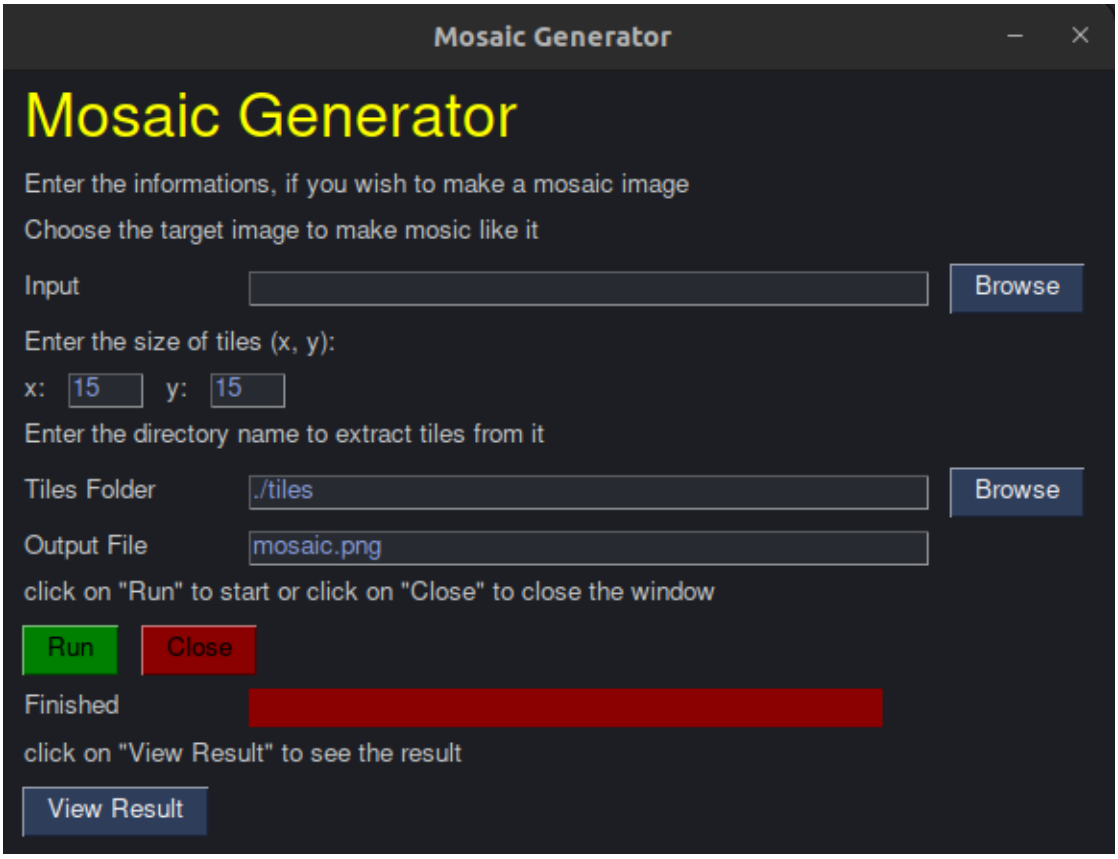
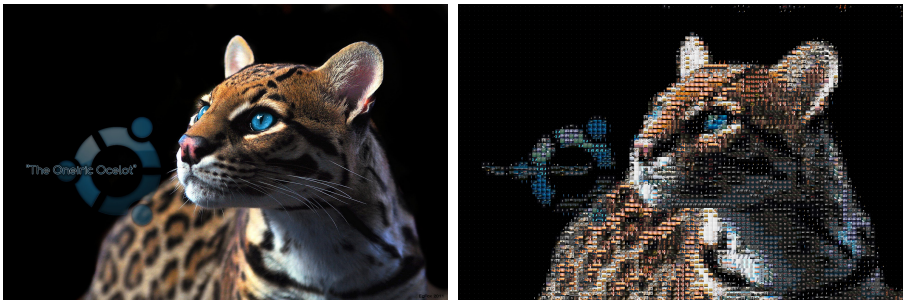


Abb. 2.2. Mosaik-Generator Programm Fenster mit default-Eingaben.

Die folgenden Abbildungen zeigen einen Beispiel von Mosaik



(a) Zielbild (b) Mosaikbild

Abb. 2.3. Input und Output Bilder in Mosik-Generator

Und die Einstellungen des Mosaikbilds finden in der Abbildung 2.4

Abb. 2.4. Mosaicbild mit tiles_size(20, 20) zeigen mit dem Klick auf View Result.

Ein Teil des Mosaikbilds wird vergrößert und in der Abbildung 2.5 wird gezeigt

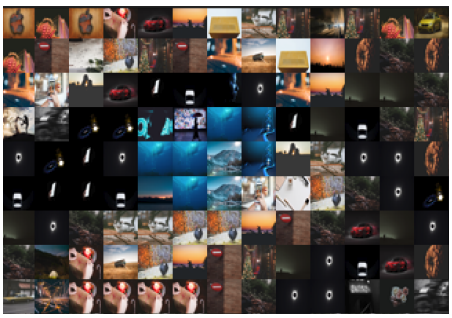


Abb. 2.5. Die Darstellung von Auge der Katze mit tiles_size(20, 20) in der Mosikbild.

3

Beschreibung der Codes

Nachdem die Eingaben implementiert wurde, wird ein Objekt mosaic von der Klasse „MosaicGenerator“ erstellt und die Methode „createMosaic()“ aufgeruft. Dann wird in der Methode „createMosaic()“ die Methode „create()“ aufgeruft die folgenden Operationen durchführen.

- Die Größe der Kacheln wird durch die Methode „get_size()“ eingesetzt.
- Und danach werden die Namen der Kacheln mithilfe der Bibliothek "glob"durch die Methode „get_paths()“ in der Klasse „MosaicGenerator“ aus dem eingegebenen Verzeichnis gesammelt.
- Die Kacheln werden als Images-Objekte in der Klasse „Tile_Img“ durch die Methode „import_all_tiles()“ importiert.
- Laut der eingegeben Größe werden die Größen aller importierten Kacheln durch die Methode „resize_tiles()“, die in der Klasse „TileImg“ gefunden ist, ändern. Diese Änderung wird mithilfe des Objects „Image“ aus der Bibliotheke „pillow“ erledigt.
- Um die Kacheln in der Mosaik richtig zu platzieren, müssen die Durchschnitte der Farben der Kacheln durch die Methode „find_avg_colors()“ berechnet. Jedes Bild wird als Array von [r, g, b] dargestellt und dann werden die Durchschnitte mithilfe „numpy“ gefunden.

```
1     avg_colors = []
2     for img in imgs:
3         colors = numpy.array(img)
4         avg_color_per_row = numpy.mean(colors, axis=0)
5         avg_color = numpy.mean(avg_colors_per_row, axis=0)
6         avg_colors.append(avg_color)
```

- Um das Hauptbild zu teilen, wird seine Größe mit der Methode „resize_target_img()“ ändert. Die Änderung wird so geschieht, dass jedes Pixel in den geänderten Kacheln genau die Größe der Kachelquadrat im Orginalbild hat.
- Jetzt muss für jedes Pixel im skalierten Bild das Quadrat mit der nächsten durchschnittlichen Farbe gefunden. Das wird in der Methode „find_nearst_tile()“. Hier wird KDTree-Klasse für die schnelle Suche nach dem nächsten Nachbaren benutzt. Diese Klasse stellt einen Index für einen Satz von k-Dimensionalen Punkten bereit, der verwendet werden kann, um schnell die nächsten Nachbarn eines beliebigen Punktes nachzuschlagen. Documentation der Klasse KDTree.

Und Method „query()“ wird nach dem nächsten nähersten Nachbarn im Baum gesucht. Es wurde das Argument „K=5“ für die KDTree-Abfrage angegeben. Das endgültige Bild hat viele Wiederholungen, da Pixel, die nahe beieinander liegen, einen sehr ähnlichen Wert haben. Also gibt der Baum die nächsten 30 Werte für jedes Pixel zurück, und es wird verwendet dann eine zufällige Ganzzahl von 0 bis 4, um auszuwählen, welche den Schnitt machen würde. Jetzt gibt es viel weniger Wiederholungen. Das folgende Code zeigt, wie diese Operation passiert.

```
1     w = image.size[0]
2     h = image.size[1]
3     map_nearst = np.zeros((w, h), dtype=np.uint32) # map = {pixel:nearst_tile}
4     tree = spatial.KDTree(avg_color)
5     for i in range(w):
6        for j in range(h):
7           # find nearst tile in tree for pixel (i,j)
8           pixel = image.getpixel((i, j))
9           nearst_tiles = tree.query(pixel, k=5)
10          rnd = random.randint(0, 4)
11          map_nearst[i, j] = nearst_tiles[1][rnd]
```

- Schließlich werden die Kacheln in der passenden Stelle zum Zielbild durch die Methode „draw_tiles()“ platziert und dann wird das erledigte Mosaikbild im gewünschten Ort durch die Methode „save_mosaic()“ gespeichert.

- Die Abbildung 3.1 zeigt das GUI-Fenster des Programms nach dem Ende der Erstellung vom Mosaik.
  Wenn man an die „View Result“-Taste klickt, wird das Mosaikbild gezeigt.
  In Bezug auf die Eingabefelder, indem man einen Text schreiben oder eine Taste klickt kann, wurde so wie folgenden Beispiel mit Nutzung von PySimpleGUI geschriben.

```
1     layout = [
2       [sg.Text('Input', size=(15, 1)), sg.In(size=(50, 1), default_text="",
3       enable_events=True, key="target_img"),
4       sg.FileBrowse()
5       ],
6       [
7        sg.ProgressBar(100, orientation='h', bar_color='dark Red',
8        size=(30, 20), key='progress_mosaic')
9       ]
10    ]
```

sg.Text(...) zeigt den Text in Gui und sg.In(...) gibt den Input-Feld in Gui, aber sg.FileBrowse() liefert die Browse-Tast, um den verzeichnis auszuwählen.Außerdem wird Fortschrittsanzeige mit sg.ProgressBar(...) dargestellt, Wobei sg = PySimpleGUI(). Siehe Abbildung 3.1
Für die Fehlerfenestern soll die Methode popup_error benutzt, wie zum Beispiel:

```
1      if event == 'Yes' and not view_ready:
2          sg.popup_error('Not yet created Mosaic!')
```

Die Argumente Eingaben, din durch Gui-Fenster implementiert, werden durch argparse festgestellt. Siehe das folgendes code.

```
1       parser = argparse.ArgumentParser()
2       parser.add_argument('-i', dest='target_img')
3       parser.add_argument('-o', dest='output_img')
4       parser.add_argument('-t', dest='tiles_dir')
5       parser.add_argument('-x', dest='x', default=15)
6       parser.add_argument('-y', dest='y', default=15)
7       args = parser.parse_args()
8       # ohne GuiFenster
9       args.target_img = str(args.input_img)
10      args.output_img = str(args.output_img)
11      args.tiles_dir = str(args.tiles_dir)
12      x = int(str(args.x))
13      y = int(str(args.y))
14      # mit GuiFenster
15      args.target_img = values["target_img"]
16      args.tiles_dir = values['tiles_dir']
17      args.output_img = str(values['output_img'])
18      args.x = str(int(values["x"]))
19      args.y = str(int(values["y"]))
```

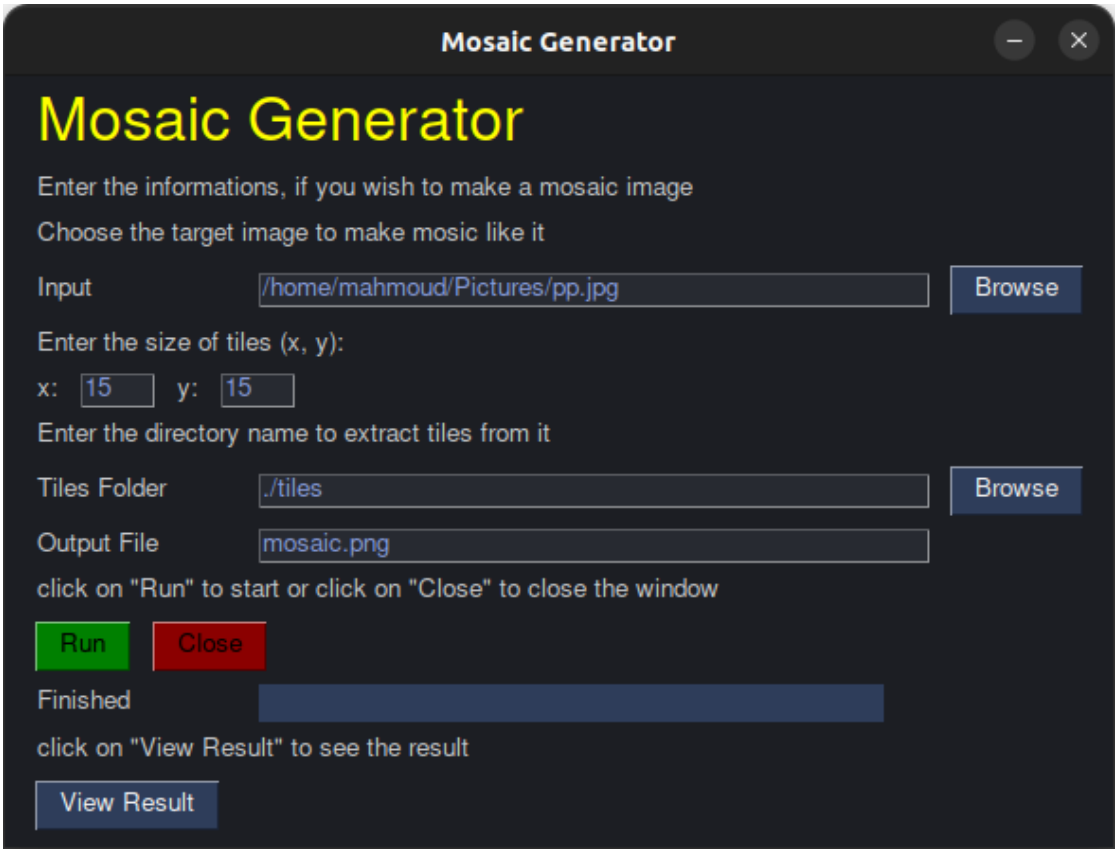


Abb. 3.1. Mosaik-Generator Programm Fenster nach der Ausführung.

4

Fazit

Bei einem Mosaik-Generator-Programm werden mehrere kleine Einzelbilder so platziert, dass das zuvor ausgewählte Hauptbild erscheint. Kleine Einzelbilder werden farblich analysiert und nach einer speziellen Logik angeordnet. Diese Darstellung der Mosaikbilder macht die einzelnen Bilder sehr deutlich.
Das Mosaikbild ist also eine ganz besondere Überraschung für einen beschenkten. Mosaikbilder werden sehr häufig als besonderes und persönliches Geschenk verwendet. Sei es zur Hochzeit, Geburtstag oder einem Firmenjubiläum.