

Klausur Computergrafik und Bildverarbeitung

WS 2021/22

- Es gibt insgesamt 140 Punkte für die folgenden Aufgaben.
- Für die Notenberechnung werden 100 Punkte als Maximum verwendet, jeweils 5 Punkte pro Notenstufe, d.h. 95-100 Punkte = 1.0 .
- Sie können also die Aufgaben bzw. Teilaufgaben auswählen, die sie besonders gut beherrschen.
- Sie dürfen beliebiges Material verwenden, aber sich nicht mit Kommilitonen austauschen.
- Wenn Sie Code aus Online-Quellen verwenden, müssen Sie die Quelle angeben.
- Code aus Vorlesungsmaterialien benötigt keine Quellenangabe.
- Die Aufgaben sind grob nach Kniffligkeit geordnet (bis auf die allerletzte Aufgabe).
- Es gibt Punkte auf Teillösungen und sinnvolle Lösungsansätze.
- Codequalität wird nicht bewertet.
- Für die Bewertung wird der Code der Reihe nach ausgeführt. Prüfen Sie vor Abgabe, ob das auch klappt, indem Sie den Kernel neustarten.

Bitte tragen Sie im folgenden Feld Name und Matrikelnummer ein

Name, Matrikelnummer

In []: *# Bitte verwenden Sie nach Möglichkeit die folgenden Namen für importierte Mod*

```
import cv2
import matplotlib
from matplotlib import pyplot as plt
import numpy as np
from numpy import matrix as M
from math import sin, cos, pi
from PIL import Image
import collections

# optional
matplotlib.rcParams['figure.figsize'] = [12, 8]
```

In []: **def** show(img):
 plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))

def showg(img):
 plt.imshow(img, cmap='gray')

def showb(img):
 plt.imshow(img, cmap='Greys', interpolation='nearest')

1. Grundwissen (11 P.)

1.1 Aussagen

Bitte geben Sie für jede der folgenden Aussagen an, ob sie wahr oder falsch ist, indem Sie "W" oder "F" davorschreiben.

- Der Bresenham-Algorithmus dient dazu, Kanten in Bildern zu finden.
- Filter-Kernel können u.a. zum Weichzeichnen oder zum Schärfen eines Bildes verwendet werden.
- Mit der Hough-Transformation kann man beliebige Konturen in einem Bild finden.
- Der Painter's Algorithm ist relativ ineffizient.
- Die WebGL-API bietet Funktionen zum Erzeugen eines Szenengraphen.

1.2 Außenseiter


In jeder Zeile gehört ein Begriff nicht zu den anderen. Markieren Sie diesen **fett**.

1. OpenGL - Vulkan - Metal - Glacier
2. Flat - Ambient - Specular - Diffuse
3. Euler Rotation - Quaternions - Rodrigues Vectors - Gauss Axes
4. RANSAC - Raytracing - Radiosity - Phong Shading
5. Vertex - Voxel - Vektor - Volume
6. ORB - ATOL - SURF - SIFT

2. Bildverarbeitung (30 P.)

Laden Sie das Bild "bild1.png" (5 P.) und modifizieren Sie es mittels OpenCV-Funktionen, so dass die Farben korrekt aussehen (5 P.). Dann verbessern Sie das Bild indem Sie den Kontrast anpassen (10 P.) (hartkodierte Parameter sind ok) und es schärfen (10 P.).

Verwenden Sie nur Numpy-/OpenCV-Funktionen. (Tipp: es gibt keine eingebauten spezifischen Funktionen zur Kontrastanpassung oder zum Schärfen des Bildes)

Zeigen Sie das Bild im Notebook an (es muss nicht gespeichert werden) Es sollte dann ungefähr so aussehen (nur größer):  Lösung

```
In [ ]: # Hier könnte Ihre Lösung stehen!  
show(img)
```

3. 2D/3D-Grafik (59 P.)

Untenstehender Code soll einen sich drehenden 3D-Würfel zeigen, dessen Farbe sich kontinuierlich ändert (siehe `cube.mp4`).

 Ergebnis

Leider sind einige Bugs und Probleme im Code. Beheben Sie diese:

- Beheben Sie die drei Syntax-/Tippfehler (9 P.)

- Die verwendete orthographische Projektion sieht nicht schön aus. Verwenden Sie stattdessen die perspektivische Projektion (10 P.)
- Der Würfel ist unpraktisch definiert, so dass er sich nicht um seinen Mittelpunkt dreht. Korrigieren Sie dies (10 P.)

Außerdem fehlt noch etwas Code. Ergänzen Sie diesen:

- Verschieben Sie den Würfel mittels einer geeigneten Matrix, so dass er ganz zu sehen ist. (15 P.)
- Sorgen Sie dafür, dass der Würfel kontinuierlich, sanft und loopend seine Farbe ändert (*Tipp 1: HSV macht das einfacher. Tipp 2: wenn Sie fremden Code verwenden, achten Sie darauf, welchen Wertebereich die zurückgegebenen Werte haben. Tipp 3: Sie können die Werte in einem Numpy-Array einfach skalieren, indem Sie dieses mit einer Zahl multiplizieren*) (15 P.)

```
In [ ]: from IPython.display import Video
        Video('cube.mp4', embed=True)
```

```
In [ ]: # Define our simple 3D cube

#   1 - - - - - 7
#   - \         - \
#   - 3 - - - - - 5
#   - -         - -
#   - -         - -
#   - -         - -
#   0 - - - - - 6   -
#       \ -         \ -
#       2 - - - - - 4

cube_corners=[[-0.5, -0.5, 0.0], # 0
               [-0.5, -0.5, 1],  # 1
               [-0.5, 0.5, 0.0],  # 2
               [-0.5, 0.5, 1],    # 3
               [0.5, 0.5, 0.0],   # 4
               [0.5, 0.5, 1],     # 5
               [0.5, -0.5, 0.0],  # 6
               [0.5, -0.5, 1]]   # 7

cube_edges = [(0, 2), (2, 4), (4, 6), (6, 0),
              (1, 3), (3, 5), (5, 7), (7, 1),
              (0, 1), (2, 3), (4, 5), (6, 7)]

def transform(points, angle=45):
    cosa = cos(pi*angle/180)
    sina = sin(pi*angle/180)
    R = M([[cosa, 0, sina, 0],
           [0, 1, 0, 0],
           [-sina, 0, cosa, 0],
           [0, 0, 0, 1]])

    transformed_points = []
    for p in points:
        p_hom = p + [1]
        p_new = R @ p_hom
```

```

        transformed_points.append(p_new)
    return transformed_points

def project(points, width=200, height=200):
    P = M([[1, 0, 0, 0],
           [0, 1, 0, 0],
           [0, 0, 0, 0],
           [0, 0, 0, 1]])
    projected_points = []
    for p in points:
        p_new = P @ np.transpose(p)
        # assume points between -0.5 and 0.5
        x = int((float(p_new[0]/p_new[3]) + 0.5) * width)
        y = int((float(p_new[1]/p_new[3]) + 0.5) * height)
        projected_points.append((x,y))
    return projected_points

def cube(rotation=0, color=(255,255,255))
    projected_points = project(transform(cube_corners, rotation))
    cube_preview = np.zeros((200,200,3), np.uint8)
    for edge in cube_edges:
        p1 = projected_points[edge[0]]
        p2 = projected_points[edge[1]]
        cv2.line(cube_preview, p1, p2, color, 2)
    cube_preview = cv2.cvtColor(cube_preview, cv2.COLOR_BGR2RGB) # Converting
    return(Image.fromarray(cube_preview))

```

```

In [ ]: from IPython.display import DisplayHandle
        from time import sleep
        d = DisplayHandle()
        d.display(cube())

```

```

In [ ]: angle = 0
        while True:
            angle += 5
            if angle > 360:
                angle = 0
            color = (255,255,255)
            d.update(cube(angle, color))
            sleep(0.016)
        # you need to send a KeyboardInterrupt here in order to continue
        # you can do this via the ■ icon in the toolbar

```

4. Bildverarbeitung (35 P.)

Schreiben Sie Code, der folgende Informationen (korrekt) aus Bildern extrahiert. Der Code muss für die Bilder `bild2.png` und `bild3.png` korrekte Ergebnisse liefern.

- Wieviele Kreise sind im Bild? (Tipp: sinnvolle Start-Parameter für die benötigte Funktion sind z.B. `dp=1, param1=100,param2=30`) (10 P.)
- Zeichnen Sie die Kreise ins Bild ein. (5 P.)
- Welche Farbe kommt im Bild am zweithäufigsten vor? Geben Sie diese aus (Tipp: erstellen Sie eine Liste mit allen Pixelwerten und verwenden Sie `Counter.most_common()` aus dem `collections`-Modul) (15 P.)
- Zeichnen Sie ein Rechteck mit dieser Farbe ins linke obere Eck (5 P.)

```
In [ ]: orig = cv2.imread("bild2.png")
        show(orig)
```

5. Feedback (5 P.)

Wie empfanden Sie die Klausur? Was war gut, was könnten wir noch verbessern? (5 P.)

(hier ausfüllen)

ENDE

```
In [ ]:
```