虚拟形象制作全流程

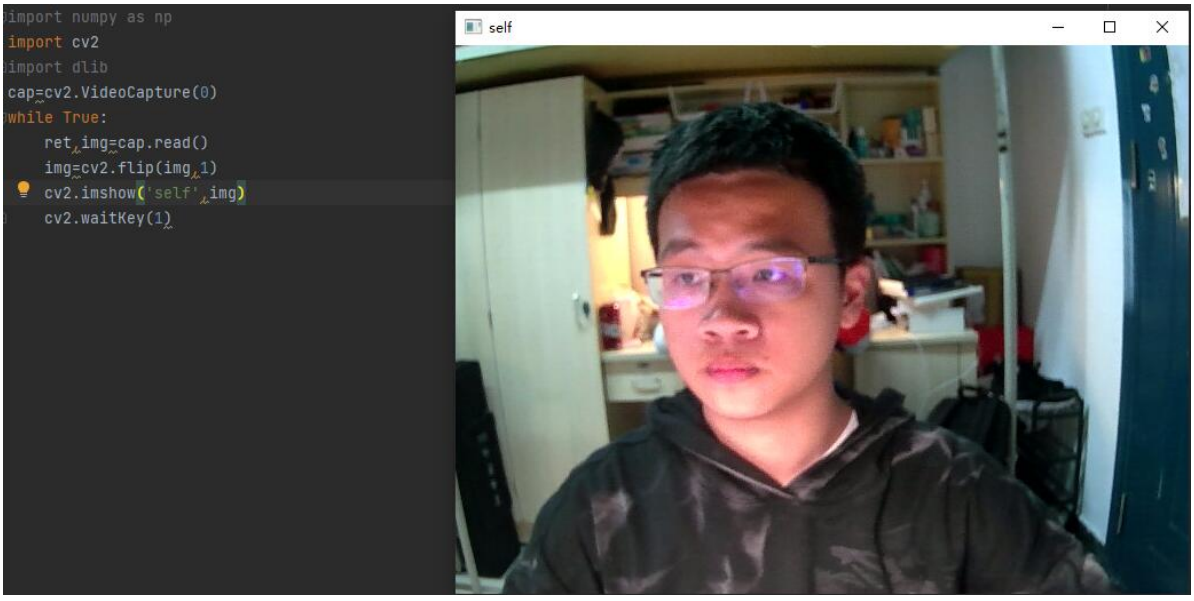
项目起源

评委老师大家好，我们是MetaVlinker虚拟形象赛道组，寓意元宇宙时代中的虚拟链接者，Live2D形象是当前直播界广泛采用的框架，我们希望能够借助Python，创作出一个元宇宙时代的Vtuber live2D形象，本项目主要采用的工具是python的dlib库和opencv库，借助矩阵的不断变形创造出一个能够面部识别的虚拟形象，项目的主角为我们的Linker，相信大家已经看到了实际效果，现在，就让我们看一下Linker是怎么创造出来的吧！

读取真实人脸

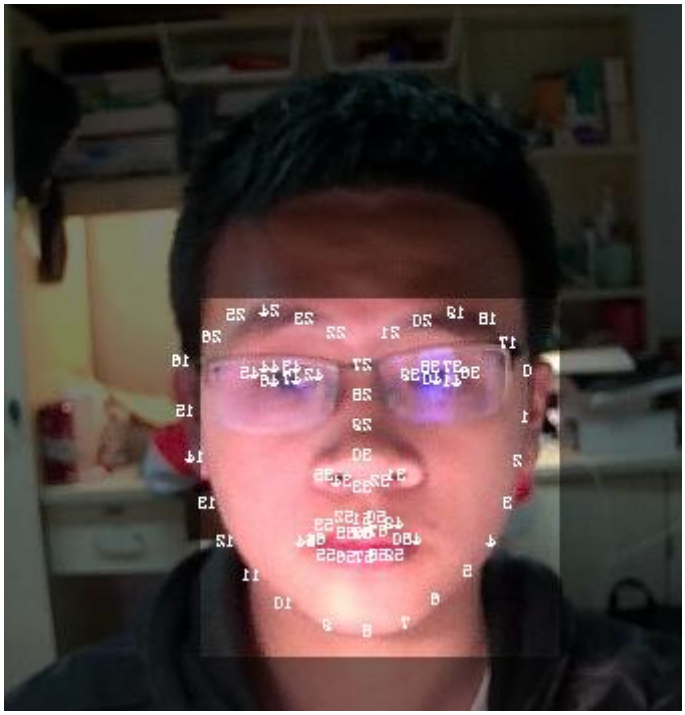
首先，我们需要调用OpenCV库来调取摄像头，以取得视频流，并且将其播放到窗口上,代码和具体效果如下：

```
cap=cv2.VideoCapture(0)
while True:
    ret,img=cap.read()
    img=cv2.flip(img,1)
    cv2.imshow('self',img)
    cv2.waitKey(1)
```



然后，我们需要从中捕捉关键点，作为面部捕捉的识别来源，而在此之前，我们需要对框架进行简单的训练，通过正常和闭眼的面部表情，来为框架做识别，在这里，我选择了两张平时照片：

在简单训练完之后，引入Dlib库的shape_predictor_68_face_landmarks.dat从实时摄像头提取68个关键点，如图所示：

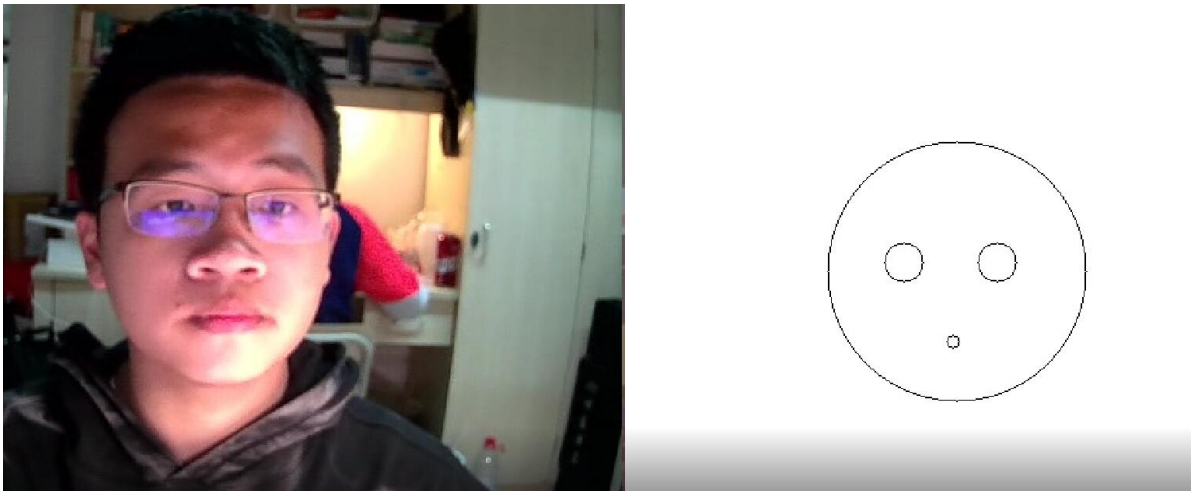


构筑这样一个框架后，需要计算面部特征，根据关键点计算出它们的坐标，然后就可以调用Dlib库为人脸的动作进行一个简单的描边：

```
def 生成构造点(关键点):
    def 中心(索引数组):
        return sum([关键点[i] for i in 索引数组]) / len(索引数组)
    左眉 = [18, 19, 20, 21]
    右眉 = [22, 23, 24, 25]
    下巴 = [6, 7, 8, 9, 10]
    鼻子 = [29, 30]
    return 中心(左眉 + 右眉), 中心(下巴), 中心(鼻子)
```

```
def 画图(横旋转量, 竖旋转量):
    img = np.ones([512, 512], dtype=np.float32)
    脸长 = 200
    中心 = 256, 256
    左眼 = int(220 + 横旋转量 * 脸长), int(249 + 竖旋转量 * 脸长)
    右眼 = int(292 + 横旋转量 * 脸长), int(249 + 竖旋转量 * 脸长)
    嘴 = int(256 + 横旋转量 * 脸长 / 2), int(310 + 竖旋转量 * 脸长 / 2)
    cv2.circle(img, 中心, 100, 0, 1)
    cv2.circle(img, 左眼, 15, 0, 1)
    cv2.circle(img, 右眼, 15, 0, 1)
    cv2.circle(img, 嘴, 5, 0, 1)
    return img
```

具体效果如图所示，可以通过面部的动效，得到一个镜面反转的简单面部捕捉实时反馈可视化图案：

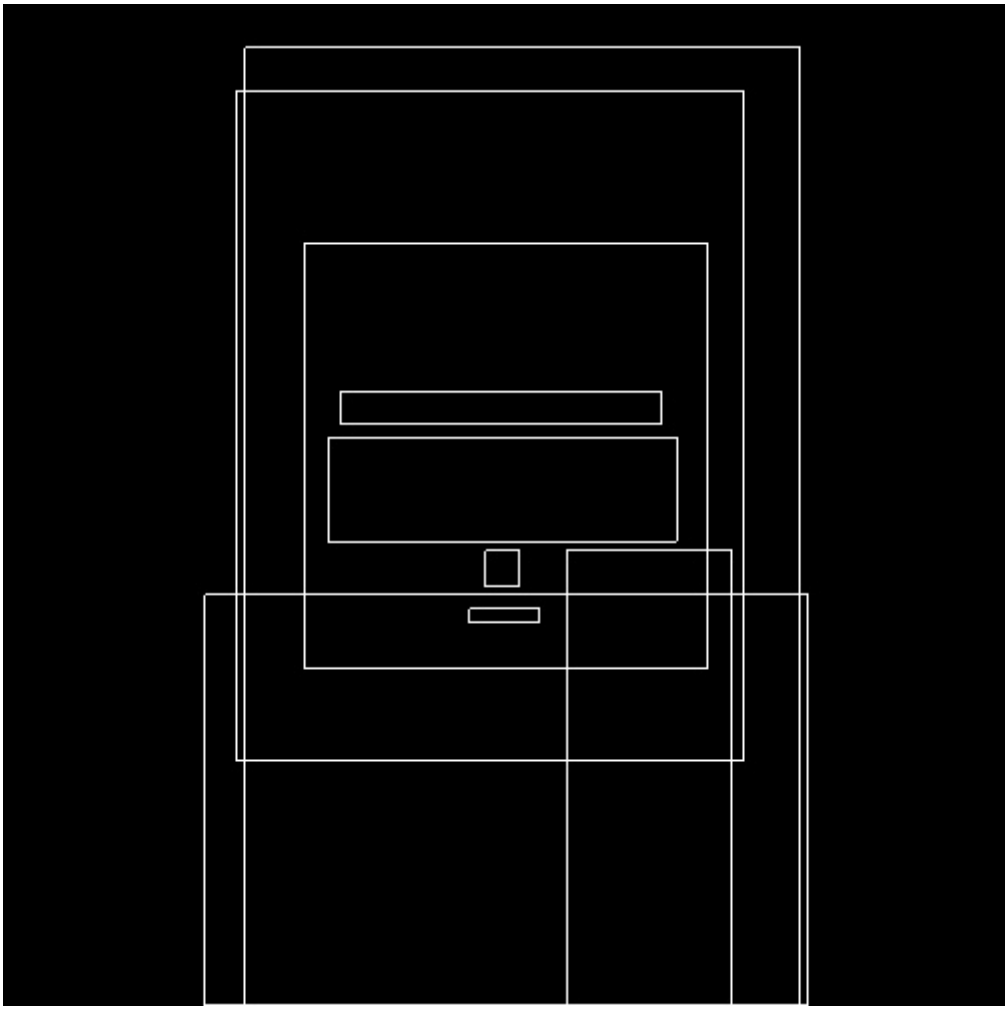


Dlib库描边.mp4

画图和绘图

我们进行了立绘的绘制，导成PSD文件，如图所示

然后通过Pip库中的psd-tool将PSD文件读入python中，继而通过OpenCV将这些图层组合起来，按照顺序写入图层，就可以通过OpenGL绘图，OpenGL中的坐标是四维坐标（x,y,z,w),(x,y)为屏幕坐标，z为深度坐标，然后进行适当的调参，就可以对PSD文件进行描边，效果图如下：



然后再绘制每个图层的时候，将纹理绑定到对应的纹理编号上，在OpenGL窗口上的图像如下：

但是图像有一点模糊，所以要把纹理放大一点，代码如下：

```
for 图层数据 in 所有图层:
    a, b, c, d = 图层数据['位置']
    q, w = 图层数据['纹理座标']
    p1 = np.array([a, b, 0, 1, 0, 0])
    p2 = np.array([a, d, 0, 1, w, 0])
    p3 = np.array([c, d, 0, 1, w, q])
    p4 = np.array([c, b, 0, 1, 0, q])
    model = matrix.scale(2 / psd尺寸[0], 2 / psd尺寸[1], 1) @ \
        matrix.translate(-1, -1, 0) @ \
        matrix.rotate_ax(-math.pi / 2, axis=(0, 1))
    glBindTexture(GL_TEXTURE_2D, 图层数据['纹理编号'])
    glColor4f(1, 1, 1, 1)
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL)
    glBegin(GL_QUADS)
    for p in [p1, p2, p3, p4]:
        a = p[:4]
        b = p[4:6]
        a = a @ model
        glTexCoord2f(*b)
        glVertex4f(*a)
    glEnd()
glfw.swap_buffers(window)
```

得到的效果图如下：

图片相对来说就很清晰了，能成功被读取并且在python中被捕获

让虚拟形象动起来

我们需要写一个YAML文档，识别出图像的信息，把所有需要动的图层的坐标乘上一个绕轴旋转的矩阵，就能使虚拟形象动起来，代码和效果如下：

```
for p in [p1, p2, p3, p4]:
    a = p[:4]
    b = p[4:]
    a = a @ model
    z = a[2]
    a[0:2] *= z
    b *= z
```

之后，就通过特征缓冲的方法，让虚拟形象可以根据面部捕捉动起来：代码和演示视频如下：

```
a = a @ matrix.translate(0, 0, -1) \
      @ matrix.rotate_ax(横旋转量, axis=(0, 2)) \
      @ matrix.rotate_ax(竖旋转量, axis=(2, 1)) \
      @ matrix.translate(0, 0, 1)
```

僵硬.mp4

然后就是训练表情的过程，通过撰写不同的YAML文件来表现不同的表情和动作

左转：

张嘴：

通过这些来读取出一些纹理，从而得到张嘴、左转、右转、睁眼、闭眼等一系列动作

最后得到一个可以说话的效果，主要代码如下：

```
def 获取截图(self, 反转颜色=True):
    while True:
        self.启用截图 = True
        if self.截图:
            img = np.frombuffer(self.截图, dtype=np.uint8).reshape((*Vtuber尺寸,
4)).copy()
            if 反转颜色:
                img[:, :, :3] = img[:, :, :3][:, :, ::-1]
            img = img[::-1]
            return img
        time.sleep(0.01)

def 附加变形(self, 变形名, 图层名, a, b, f):
    变形 = self.变形组[变形名]
    if 图层名 not in 变形:
        return a, b
    if '位置' in 变形[图层名]:
        d = 变形[图层名]['位置']
        if type(d) is str:
            d = eval(d)
        d = np.array(d)
        a[:, :2] += d.reshape(a.shape[0], 2) * f
    return a, b

def 多重附加变形(self, 变形组, 图层名, a, b):
    for 变形名, 强度 in 变形组:
```

```
        a, b = self.附加变形(变形名, 图层名, a, b, 强度)
    return a, b

def opengl绘图循环(self, window, 数据源, line_box=False):
    def 没有状态但是却能均匀变化的随机数(范围=(0, 1), 速度=1):
        now = time.time()*速度
        a, b = int(now), int(now)+1
        random.seed(a)
        f0 = random.random()
        random.seed(b)
        f1 = random.random()
        f = f0 * (b-now) + f1 * (now-a)
        return 范围[0] + (范围[1]-范围[0])*f

    def 锚击(x, a, b):
        x = sorted([x, a, b])[1]
        return (x-a)/(b-a)
```

把代码封装，得到完整的模型，完整代码如下：

```
import time
import math
import random
import logging
import functools
import pyvirtualcam
import numpy as np
import yaml
import threading
import glfw
import OpenGL
from OpenGL.GL import *
from OpenGL.GLU import *
from rimo_utils import matrix
from rimo_utils import 计时
import psd_tools

import 现实
def start(vtuber, size):
    r, c = size
    def q():
        with pyvirtualcam.Camera(width=r, height=c, fps=30) as cam:
            base = np.zeros(shape=(c, r, 3), dtype=np.uint8)
            while True:
                img = vtuber.获取截图(False)
                base[:, (r-c)//2:(r-c)//2+c] = img[:, :, :3]
                cam.send(base)
                time.sleep(0.01)
    t = threading.Thread(target=q)
    t.setDaemon(True)
    t.start()



Vtuber尺寸 = 1100,1100
```

```
def 相位转移(x):
    if x is None:
        return x
    if type(x) is str:
        return 相位转移(eval(x))
    if type(x) in [int, float]:
        return np.array([[x, x], [x, x]])
    else:
        return np.array(x)


class 图层类:
    def __init__(self, 名字, bbox, z, 物理, npdata):
        self.名字 = 名字
        self.npdata = npdata
        self.纹理编号, 纹理座标 = self.生成opengl纹理()
        self.变形 = []
        self.物理 = 相位转移(物理)
        深度 = 相位转移(z)
        assert len(深度.shape) == 2
        self.shape = 深度.shape

        q, w = 纹理座标
        a, b, c, d = bbox
        [[p1, p2],
         [p4, p3]] = np.array([
             [[a, b, 0, 1, 0, 0, 0, 1], [a, d, 0, 1, w, 0, 0, 1]],
             [[c, b, 0, 1, 0, q, 0, 1], [c, d, 0, 1, w, q, 0, 1]],
         ])
        x, y = self.shape
        self.顶点组 = np.zeros(shape=[x, y, 8])
        for i in range(x):
            for j in range(y):
                self.顶点组[i, j] = p1 + (p4-p1)*i/(x-1) + (p2-p1)*j/(y-1)
                self.顶点组[i, j, 2] = 深度[i, j]

    def 生成opengl纹理(self):
        w, h = self.npdata.shape[:2]
        d = 2**int(max(math.log2(w), math.log2(h)) + 1)
        纹理 = np.zeros([d, d, 4], dtype=self.npdata.dtype)
        纹理[:, :, :3] = 255
        纹理[:w, :h] = self.npdata
        纹理座标 = (w / d, h / d)

        width, height = 纹理.shape[:2]
        纹理编号 = glGenTextures(1)
        glBindTexture(GL_TEXTURE_2D, 纹理编号)
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_BGRA,
GL_FLOAT, 纹理)
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR_MIPMAP_LINEAR)
        glGenerateMipmap(GL_TEXTURE_2D)

        return 纹理编号, 纹理座标

    def 顶点组导出(self):
        return self.顶点组.copy()
```

```
class vtuber:
    def __init__(self, psd路径, 切取范围=(1024, 1024), 信息路径='信息.yaml', 变形路径
='变形.yaml'):
        psd = psd_tools.PSDImage.open(psd路径)
        with open(信息路径, encoding='utf8') as f:
            信息 = yaml.safe_load(f)
        with open(变形路径, encoding='utf8') as f:
            self.变形组 = yaml.safe_load(f)

        def 再装填():
            while True:
                time.sleep(1)
                try:
                    with open(变形路径, encoding='utf8') as f:
                        self.变形组 = yaml.safe_load(f)
                except Exception as e:
                    logging.exception(e)
        import threading
        t = threading.Thread(target=再装填)
        t.setDaemon(True)
        t.start()

        self.所有图层 = []
        self.psd尺寸 = psd.size
        self.切取范围 = 切取范围

        def dfs(图层, path=''):
            if 图层.is_group():
                for i in 图层:
                    dfs(i, path + 图层.name + '/')
            else:
                名字 = path+图层.name
                if 名字 not in 信息:
                    logging.warning(f'图层「{名字}」找不到信息，丢了！')
                    return
                a, b, c, d = 图层.bbox
                npdata = 图层.numpy()
                npdata[:, :, :3] = npdata[:, :, :3][:, :, ::-1]
                self.所有图层.append(图层类(
                    名字=名字,
                    z=信息[名字]['深度'],
                    物理=信息[名字].get('物理'),
                    bbox=(b, a, d, c),
                    npdata=npdata
                ))
        for 图层 in psd:
            dfs(图层)
        self.截图 = None
        self.启用截图 = False
        self._记忆 = {}

    def 获取截图(self, 反转颜色=True):
        while True:
            self.启用截图 = True
            if self.截图:
                img = np.frombuffer(self.截图, dtype=np.uint8).reshape((*Vtuber尺
寸, 4)).copy()
```

```
                if 反转颜色:
                    img[:, :, :3] = img[:, :, :3][:, :, ::-1]
                img = img[::-1]
                return img
            time.sleep(0.01)

    def 附加变形(self, 变形名, 图层名, a, b, f):
        变形 = self.变形组[变形名]
        if 图层名 not in 变形:
            return a, b
        if '位置' in 变形[图层名]:
            d = 变形[图层名]['位置']
            if type(d) is str:
                d = eval(d)
            d = np.array(d)
            a[:, :2] += d.reshape(a.shape[0], 2) * f
        return a, b

    def 多重附加变形(self, 变形组, 图层名, a, b):
        for 变形名, 强度 in 变形组:
            a, b = self.附加变形(变形名, 图层名, a, b, 强度)
        return a, b

    def 动(self, 图层, t):
        if 图层.物理 is None:
            return t
        res = t
        q, 上次时间 = self._记忆.get(id(图层), (None, 0))
        现在时间 = time.time()
        if q is not None:
            时间差 = min(0.1, 现在时间-上次时间)
            物理缩小 = 0.05
            w = 图层.物理.reshape(t.shape[0], 1)
            w = w * 物理缩小 + 1 * (1-物理缩小)
            ww = -((1-w)**时间差)+1
            v = t - q
            res = q + v * ww
        self._记忆[id(图层)] = res, 现在时间
        return res

    def opengl绘图循环(self, window, 数据源, line_box=False):
        def 没有状态但是却能均匀变化的随机数(范围=(0, 1), 速度=1):
            now = time.time()*速度
            a, b = int(now), int(now)+1
            random.seed(a)
            f0 = random.random()
            random.seed(b)
            f1 = random.random()
            f = f0 * (b-now) + f1 * (now-a)
            return 范围[0] + (范围[1]-范围[0])*f

        def 锚击(x, a, b):
            x = sorted([x, a, b])[1]
            return (x-a)/(b-a)

        @functools.lru_cache(maxsize=16)
        def model(xz, zy, xy, 脸大小, x偏移, y偏移):
            model_p = \
```

```
                matrix.translate(0, 0, -0.9) @ \
                matrix.rotate_ax(xz, axis=(0, 2)) @ \
                matrix.rotate_ax(zy, axis=(2, 1)) @ \
                matrix.translate(0, 0.9, 0.9) @ \
                matrix.rotate_ax(xy, axis=(0, 1)) @ \
                matrix.translate(0, -0.9, 0) @ \
                matrix.perspective(999)
            f = 750/(800-脸大小)
            extra = matrix.translate(x偏移*0.6, -y偏移*0.8, 0) @ \
                    matrix.scale(f, f, 1)
            return model_p, extra

        model_g = \
            matrix.scale(2 / self.切取范围[0], 2 / self.切取范围[1], 1) @ \
            matrix.translate(-1, -1, 0) @ \
            matrix.rotate_ax(-math.pi / 2, axis=(0, 1))

        def draw(图层):
            源 = 图层.顶点组导出()
            x, y, _ = 源.shape

            所有顶点 = 源.reshape(x*y, 8)

            a, b = 所有顶点[:, :4], 所有顶点[:, 4:]
            a = a @ model_g
            z = a[:, 2:3]
            z -= 0.1
            a[:, :2] *= z
            眼睛左右 = 横旋转量*4 + 没有状态但是却能均匀变化的随机数((-0.2, 0.2), 速度
=1.6)
            眼睛上下 = 竖旋转量*7 + 没有状态但是却能均匀变化的随机数((-0.1, 0.1), 速度=2)
            闭眼强度 = 锚击(左眼大小+右眼大小, -0.001, -0.008)
            眉上度 = 锚击(左眉高+右眉高, -0.03, 0.01) - 闭眼强度*0.1
            闭嘴强度 = 锚击(嘴大小, 0.05, 0) * 1.1 - 0.1
            a, b = self.多重附加变形([
                ['永远', 1],
                ['眉上', 眉上度],
                ['左眼远离', 眼睛左右],
                ['右眼远离', -眼睛左右],
                ['左眼上', 眼睛上下],
                ['右眼上', 眼睛上下],
                ['左眼闭', 闭眼强度],
                ['右眼闭', 闭眼强度],
                ['闭嘴', 闭嘴强度],
            ], 图层.名字, a, b)

            xz = 横旋转量 / 1.2
            zy = 竖旋转量 / 1.4
            xy = Z旋转量 / 5
            if not 图层.名字.startswith('头/'):
                xz /= 8
                zy = 0

            model_p, extra = model(xz, zy, xy, 脸大小, x偏移, y偏移)
            a = a @ model_p
            a = self.动(图层, a)
            a = a @ extra
```

```
            b *= z

            所有顶点 = np.concatenate([a, b], axis=1).reshape([x, y, 8])

            glBegin(GL_QUADS)
            for i in range(x-1):
                for j in range(y-1):
                    for p in [所有顶点[i, j], 所有顶点[i, j+1], 所有顶点[i+1, j+1],
所有顶点[i+1, j]]:
                        glTexCoord4f(*p[4:])
                        glVertex4f(*p[:4])
            glEnd()

        while not glfw.window_should_close(window):
            with 计时.帧率计('绘图'):
                glfw.poll_events()
                glClearColor(0, 0, 0, 0)
                glClear(GL_COLOR_BUFFER_BIT)
                横旋转量, 竖旋转量, Z旋转量, y偏移, x偏移, 嘴大小, 脸大小, 左眼大小, 右眼
大小, 左眉高, 右眉高 = 数据源()
                for 图层 in self.所有图层:
                    glEnable(GL_TEXTURE_2D)
                    glBindTexture(GL_TEXTURE_2D, 图层.纹理编号)
                    glColor4f(1, 1, 1, 1)
                    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL)
                    draw(图层)
                    if line_box:
                        glDisable(GL_TEXTURE_2D)
                        glColor4f(0.3, 0.3, 1, 0.2)
                        glPolygonMode(GL_FRONT_AND_BACK, GL_LINE)
                        draw(图层)
                glfw.swap_buffers(window)
                if self.启用截图:
                    glReadBuffer(GL_FRONT)
                    self.截图 = glReadPixels(0, 0, *Vtuber尺寸, GL_RGBA,
GL_UNSIGNED_BYTE)


缓冲特征 = None


def 特征缓冲(缓冲比例=0.8):
    global 缓冲特征
    新特征 = 现实.获取特征组()
    if 缓冲特征 is None:
        缓冲特征 = 新特征
    else:
        缓冲特征 = 缓冲特征 * 缓冲比例 + 新特征 * (1 - 缓冲比例)
    return 缓冲特征


def init_window():
    def 超融合():
        glfw.window_hint(glfw.DECORATED, False)
        glfw.window_hint(glfw.TRANSPARENT_FRAMEBUFFER, True)
        glfw.window_hint(glfw.FLOATING, True)
    glfw.init()
    超融合()
```

```
    glfw.window_hint(glfw.SAMPLES, 4)
    # glfw.window_hint(glfw.RESIZABLE, False)
    window = glfw.create_window(*Vtuber尺寸, 'Vtuber', None,None)
    glfw.make_context_current(window)
    monitor_size = glfw.get_video_mode(glfw.get_primary_monitor()).size
    glfw.set_window_pos(window, monitor_size.width - Vtuber尺寸[0],
monitor_size.height - Vtuber尺寸[1])
    glViewport(0, 0, *Vtuber尺寸)
    glEnable(GL_TEXTURE_2D)
    glEnable(GL_BLEND)
    glEnable(GL_MULTISAMPLE)
    glEnable(GL_CULL_FACE)
    glCullFace(GL_FRONT)
    glBlendFuncSeparate(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA, GL_ONE,
GL_ONE_MINUS_SRC_ALPHA)
    return window


if __name__ == '__main__':
    现实.启动()
    window = init_window()

   Linker = vtuber('Linker.psd')
    import sys
    sys.path.append('..')
    from utils import 虚拟摄像头
    虚拟摄像头.start(Linker, (1280, 720))
    Linker.opengl绘图循环(window, 数据源=特征缓冲)
```

最后，在合适的场景下，一个基于Python、能够稳定30帧运行、有多种表情、可以实现面部捕捉识别的Linker Live2D形象就诞生啦！