Face 3D Network

Face 3D Network

- Routine Experiment Effect
- Code Explanation
  - Code structure
    - Import related dependency classes
    - Custom face detection task class
    - Custom face mesh task class
    - Define the face mesh post-processing class
    - Define the main class of 3D face mesh
    - Execution routine

Routine Experiment Effect

In this section, we will learn how to use K230 to draw a 3D face network.

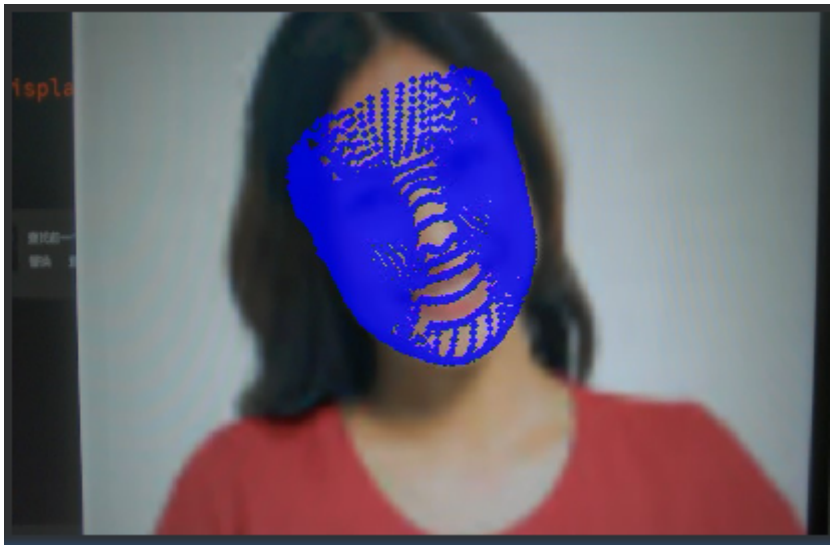
The code is in [Source code summary/07.Face/04.face_mesh.py]

After connecting to the IDE, run the sample code in this section and aim the K230 at the face. You can see that the face on the screen will be covered with the 3D network.

> 1. The model that needs to be loaded for the face 3D network operation is relatively large, so it is normal that faces cannot be recognized in the first few seconds of operation.
> 2. If too many faces appear on the screen at the same time, it may cause the system to freeze. In this case, press the RST key to restart.
>   - If you want to avoid this, you can consider adding restrictions in the run method to process only a certain number of faces.
>   - This part is only for reference, no detailed sample code is provided

【Original image】

[K230 runs the 3D face network routine]



Code Explanation

Code structure

FaceDetApp Class

- Responsible for face detection
- Use AI models to detect the location of faces in images
- Contains image pre-processing and post-processing functions

FaceMeshApp Class

- Responsible for generating 3D face mesh
- Extract features from detected face regions
- Use statistical parameters (mean and variance) for numerical adjustments

FaceMeshPostApp Class

- Post-processing of 3D meshes
- Converting mesh data into final visualization format
- Handling coordinate system transformations

FaceMesh Class

- Integrate the above three components
- Coordinate the entire process
- Managing resources and memory

Import related dependency classes

```
# 导入必要的库和模块
# Import necessary libraries and modules

# PipeLine用于构建AI处理Pipeline,ScopedTiming用于统计运行时间
# PipeLine is used to build AI processing pipeline, ScopedTiming is used for
runtime statistics
from libs.PipeLine import PipeLine, ScopedTiming

# AIBase是AI应用的基类
# AIBase is the base class for AI applications
from libs.AIBase import AIBase
```

```
# Ai2d用于2D图像处理操作
# Ai2d is used for 2D image processing operations
from libs.AI2D import Ai2d

# 导入系统操作相关模块
# Import system operation related modules
import os  # 操作系统接口 / Operating system interface
import ujson  # JSON数据处理 / JSON data processing

# 导入媒体处理相关模块
# Import media processing related modules
from media.media import *  # 媒体处理函数 / Media processing functions
from time import *  # 时间相关函数 / Time related functions

# 导入AI推理相关模块
# Import AI inference related modules
import nncase_runtime as nn  # neural network推理运行时 / Neural network inference
runtime
import ulab.numpy as np  # 类numpy数组操作 / Numpy-like array operations

# 导入图像处理和功能模块
# Import image processing and utility modules
import time  # 时间处理 / Time processing
import image  # 图像处理 / Image processing
import aidemo  # AI演示功能 / AI demo functions
import random  # 随机数生成 / Random number generation
import gc  # 垃圾回收 / Garbage collection
import sys  # 系统功能 / System functions

# 导入显示相关模块
# Import display related modules
from media.display import *  # 显示功能 / Display functions
import _thread  # 线程操作 / Thread operations
```

Custom face detection task class

```
# 自定义人脸检测任务类
# Custom face detection task class
class FaceDetApp(AIBase):
    # 初始化函数，设置模型和图像处理的各项参数
    # Initialization function, set parameters for model and image processing
    def
__init__(self,kmodel_path,model_input_size,anchors,confidence_threshold=0.25,nms
_threshold=0.3,rgb888p_size=[1280,720],display_size=[640,360],debug_mode=0):
        # 调用父类AIBase的初始化函数
        # Call parent class AIBase's initialization function
        super().__init__(kmodel_path,model_input_size,rgb888p_size,debug_mode)

        # kmodel文件的路径，包含训练好的神经网络模型
        # Path to kmodel file, which contains the trained neural network model
        self.kmodel_path=kmodel_path

        # 模型要求的输入图像尺寸[宽,高]
        # Required input image size for the model [width, height]
        self.model_input_size=model_input_size
```

```
        # 置信度阈值：检测结果的分数需要高于此值才被认为有效（范围0-1）
        # Confidence threshold: detection score must be above this value to be
considered valid (range 0-1)
        self.confidence_threshold=confidence_threshold

        # 非极大值抑制阈值：用于去除重叠框（范围0-1）
        # Non-Maximum Suppression threshold: used to remove overlapping boxes
(range 0-1)
        self.nms_threshold=nms_threshold

        # 模型预定义的锚框参数，用于检测时的边界框生成
        # Predefined anchor box parameters for the model, used for generating
bounding boxes
        self.anchors=anchors

        # 从图像传感器获取的输入图像分辨率，宽度需要16字节对齐
        # Input image resolution from image sensor, width needs to be aligned to
16 bytes
        self.rgb888p_size=[ALIGN_UP(rgb888p_size[0],16),rgb888p_size[1]]

        # 显示输出的分辨率，宽度需要16字节对齐
        # Display output resolution, width needs to be aligned to 16 bytes
        self.display_size=[ALIGN_UP(display_size[0],16),display_size[1]]

        # 调试模式标志：0关闭调试输出，>0 开启调试输出
        # Debug mode flag: 0 disable debug output, >0 enable debug output
        self.debug_mode=debug_mode

        # 创建AI2D实例，用于图像预处理
        # Create AI2D instance for image preprocessing
        self.ai2d=Ai2d(debug_mode)

        # 设置AI2D处理的数据格式：NCHW格式，uint8类型
        # Set AI2D data format: NCHW format, uint8 type

 self.ai2d.set_ai2d_dtype(nn.ai2d_format.NCHW_FMT,nn.ai2d_format.NCHW_FMT,np.uin
t8, np.uint8)

    # 配置图像预处理流程
    # Configure image preprocessing pipeline
    def config_preprocess(self,input_image_size=None):
        with ScopedTiming("set preprocess config",self.debug_mode > 0):
            # 设置预处理的输入尺寸，如果没有指定则使用默认的rgb888p尺寸
            # Set preprocessing input size, use default rgb888p size if not
specified
            ai2d_input_size=input_image_size if input_image_size else
self.rgb888p_size

            # 配置padding操作，使用指定的填充值[104,117,123]
            # Configure padding operation with specified padding values
[104,117,123]
            self.ai2d.pad(self.get_pad_param(), 0, [104,117,123])

            # 配置双线性插值的resize操作
            # Configure resize operation with bilinear interpolation
            self.ai2d.resize(nn.interp_method.tf_bilinear,
nn.interp_mode.half_pixel)
```

```
            # 构建预处理Pipeline，设置输入输出的尺寸
            # Build preprocessing pipeline, set input and output dimensions
            self.ai2d.build([1,3,ai2d_input_size[1],ai2d_input_size[0]],
[1,3,self.model_input_size[1],self.model_input_size[0]])

    # 处理模型的原始输出，得到最终的检测结果
    # Process raw model output to get final detection results
    def postprocess(self,results):
        with ScopedTiming("postprocess",self.debug_mode > 0):
            # 调用aidemo库进行人脸检测后处理
            # Call aidemo library for face detection post-processing
            res =
aidemo.face_det_post_process(self.confidence_threshold,self.nms_threshold,self.m
odel_input_size[0],self.anchors,self.rgb888p_size,results)
            # 返回检测结果，如果没有检测到人脸返回空列表
            # Return detection results, return empty list if no face detected
            if len(res)==0:
                return res
            else:
                return res[0]

    # 计算padding参数，确保图像等比例缩放并居中
    # Calculate padding parameters to ensure proportional scaling and centering
    def get_pad_param(self):
        dst_w = self.model_input_size[0]
        dst_h = self.model_input_size[1]
        # 计算宽高缩放比例，选择较小的比例保持长宽比
        # Calculate width and height scaling ratios, choose smaller ratio to
maintain aspect ratio
        ratio_w = dst_w / self.rgb888p_size[0]
        ratio_h = dst_h / self.rgb888p_size[1]
        ratio = ratio_w if ratio_w < ratio_h else ratio_h

        # 计算缩放后的新尺寸
        # Calculate new dimensions after scaling
        new_w = (int)(ratio * self.rgb888p_size[0])
        new_h = (int)(ratio * self.rgb888p_size[1])

        # 计算需要填充的边距，使图像居中
        # Calculate padding margins to center the image
        dw = (dst_w - new_w) / 2
        dh = (dst_h - new_h) / 2

        # 返回padding参数：上下左右的填充像素数
        # Return padding parameters: number of pixels to pad on top, bottom,
left, right
        top = (int)(round(0))
        bottom = (int)(round(dh * 2 + 0.1))
        left = (int)(round(0))
        right = (int)(round(dw * 2 - 0.1))
        return [0,0,0,0,top, bottom, left, right]
```

Custom face mesh task class

```
# 自定义人脸网格任务类，用于检测人脸的3D网格结构
# Custom face mesh detection class, used for detecting 3D mesh structure of
faces
class FaceMeshApp(AIBase):
    def __init__(self,kmodel_path,model_input_size,rgb888p_size=
[640,360],display_size=[640,360],debug_mode=0):
        # 调用父类AIBase的初始化方法
        # Call parent class AIBase's initialization method
        super().__init__(kmodel_path,model_input_size,rgb888p_size,debug_mode)

        # 初始化基本参数 / Initialize basic parameters
        self.kmodel_path=kmodel_path          # 模型文件路径 / Model file path
        self.model_input_size=model_input_size # 模型输入尺寸 / Model input size
        self.rgb888p_size=[ALIGN_UP(rgb888p_size[0],16),rgb888p_size[1]]  # 输入
图像尺寸(16字节对齐) / Input image size (16-byte aligned)
        self.display_size=[ALIGN_UP(display_size[0],16),display_size[1]]   # 显示
尺寸(16字节对齐) / Display size (16-byte aligned)
        self.debug_mode=debug_mode            # 调试模式标志 / Debug mode flag

        # 人脸网格模型的统计参数(训练集统计得到) / Statistical parameters for face mesh
model (obtained from training set)
        # 均值数组 / Mean array
        self.param_mean = np.array([0.0003492636315058917,2.52790130161884e-
07,-6.875197868794203e-07,60.1679573059082,-6.295513230725192e-
07,0.0005757200415246189,-5.085391239845194e-
05,74.2781982421875,5.400917189035681e-07,6.574138387804851e-
05,0.0003442012530285865,-66.67157745361328,-346603.6875,-67468.234375,46822.265
625,-15262.046875,4350.5888671875,-54261.453125,-18328.033203125,-1584.328857421
875,-84566.34375,3835.960693359375,-20811.361328125,38094.9296875,-19967.8554687
5,-9241.3701171875,-19600.71484375,13168.08984375,-5259.14404296875,1848.6478271
484375,-13030.662109375,-2435.55615234375,-2254.20654296875,-14396.5615234375,-6
176.3291015625,-25621.919921875,226.39447021484375,-6326.12353515625,-10867.2509
765625,868.465087890625,-5831.14794921875,2705.123779296875,-3629.417724609375,2
043.9901123046875,-2446.6162109375,3658.697021484375,-7645.98974609375,-6674.452
63671875,116.38838958740234,7185.59716796875,-1429.48681640625,2617.366455078125
,-1.2070955038070679,0.6690792441368103,-0.17760828137397766,0.05672552809119224
5,0.03967815637588501,-0.13586315512657166,-0.09223993122577667,-0.1726071834564
209,-0.015804484486579895,-0.1416848599910736],dtype=np.float)
        # 人脸mesh参数方差
        self.param_std = np.array([0.00017632152594160289,6.737943476764485e-
05,0.0004470848944038152 7,26.55023193359375,0.0001231376954820007,4.493021697271
615e-05,7.923670636955649e-
05,6.982563018798828,0.0004350444069132209,0.0001231489004567 2655,0.000174000015
24947584,20.80303955078125,575421.125,277649.0625,258336.84375,255163.125,150994
.375,160086.109375,111277.3046875,97311.78125,117198.453125,89317.3671875,88493.
5546875,72229.9296875,71080.2109375,50013.953125,55968.58203125,47525.50390625,4
9515.06640625,38161.48046875,44872.05859375,46273.23828125,38116.76953125,28191.
162109375,32191.4375,36006.171875,32559.892578125,25551.1171875,24267.509765625,
27521.3984375,23166.53125,21101.576171875,19412.32421875,19452.203125,17454.9843
75,22537.623046875,16174.28125,14671.640625,15115.6884765625,13870.0732421875,13
746.3125,12663.1337890625,1.5870834589004517,1.5077009201049805,0.58813577890396
12,0.5889744758605957,0.21327851712703705,0.2630201280117035,0.2796429395675659,
0.38030216097831726,0.16162841022014618,0.2559692859649658],dtype=np.float)
        # 初始化AI2D图像预处理器 / Initialize AI2D image preprocessor
        self.ai2d=Ai2d(debug_mode)
```

```
        # 设置AI2D数据格式(NCHW格式，uint8类型) / Set AI2D data format (NCHW format, uint8 type)

 self.ai2d.set_ai2d_dtype(nn.ai2d_format.NCHW_FMT,nn.ai2d_format.NCHW_FMT,np.uint8, np.uint8)

    # 配置预处理操作 / Configure preprocessing operations
    def config_preprocess(self,det,input_image_size=None):
        with ScopedTiming("set preprocess config",self.debug_mode > 0):
            # 设置输入尺寸 / Set input size
            ai2d_input_size=input_image_size if input_image_size else self.rgb888p_size

            # 根据人脸检测框计算ROI区域 / Calculate ROI area based on face detection box
            roi = self.parse_roi_box_from_bbox(det)
            # 设置裁剪区域 / Set crop area
            self.ai2d.crop(int(roi[0]),int(roi[1]),int(roi[2]),int(roi[3]))
            # 设置双线性插值缩放 / Set bilinear interpolation scaling
            self.ai2d.resize(nn.interp_method.tf_bilinear, nn.interp_mode.half_pixel)
            # 构建预处理流程 / Build preprocessing pipeline
            self.ai2d.build([1,3,ai2d_input_size[1],ai2d_input_size[0]],

[1,3,self.model_input_size[1],self.model_input_size[0]])
            return roi

    # 后处理函数:应用统计参数进行数值调整 / Postprocessing function: apply statistical parameters for value adjustment
    def postprocess(self,results):
        with ScopedTiming("postprocess",self.debug_mode > 0):
            # 应用标准差和均值进行反归一化 / Apply standard deviation and mean for denormalization
            param = results[0] * self.param_std + self.param_mean
            return param

    # 从人脸边界框计算ROI区域 / Calculate ROI area from face bounding box
    def parse_roi_box_from_bbox(self,bbox):
        # 解析边界框坐标 / Parse bounding box coordinates
        x1, y1, w, h = map(lambda x: int(round(x, 0)), bbox[:4])

        # 计算ROI区域,包含额外边距 / Calculate ROI area with extra margins
        old_size = (w + h) / 2
        center_x = x1 + w / 2
        center_y = y1 + h / 2 + old_size * 0.14  # 略微向下偏移以包含下巴 / Slight downward offset to include chin
        size = int(old_size * 1.58)  # 扩大区域以包含整个面部 / Enlarge area to include whole face

        # 计算新的边界框坐标 / Calculate new bounding box coordinates
        x0 = center_x - float(size) / 2
        y0 = center_y - float(size) / 2
        x1 = x0 + size
        y1 = y0 + size

        # 确保坐标不超出图像边界 / Ensure coordinates don't exceed image boundaries
        x0 = max(0, min(x0, self.rgb888p_size[0]))
        y0 = max(0, min(y0, self.rgb888p_size[1]))
```

```
        x1 = max(0, min(x1, self.rgb888p_size[0]))
        y1 = max(0, min(y1, self.rgb888p_size[1]))

        # 返回ROI参数(x,y,width,height) / Return ROI parameters(x,y,width,height)
        roi = (x0, y0, x1 - x0, y1 - y0)
        return roi
```

Define the face mesh post-processing class

```
# 人脸网格后处理类,用于生成最终的3D网格数据
# Face mesh post-processing class, used to generate final 3D mesh data
class FaceMeshPostApp(AIBase):
    def __init__(self,kmodel_path,model_input_size,rgb888p_size=
[640,360],display_size=[640,360],debug_mode=0):
        super().__init__(kmodel_path,model_input_size,rgb888p_size,debug_mode)
        # 初始化基本参数,与FaceMeshApp类似 / Initialize basic parameters, similar to
FaceMeshApp
        self.kmodel_path=kmodel_path
        self.model_input_size=model_input_size
        self.rgb888p_size=[ALIGN_UP(rgb888p_size[0],16),rgb888p_size[1]]
        self.display_size=[ALIGN_UP(display_size[0],16),display_size[1]]
        self.debug_mode=debug_mode
        self.ai2d=Ai2d(debug_mode)

 self.ai2d.set_ai2d_dtype(nn.ai2d_format.NCHW_FMT,nn.ai2d_format.NCHW_FMT,np.uin
t8, np.uint8)

    # 预处理函数:解析面部参数并转换为模型输入格式
    # Preprocessing function: parse face parameters and convert to model input
format
    def preprocess(self,param):
        with ScopedTiming("set preprocess config",self.debug_mode > 0):
            param = param[0]
            # 定义参数维度 / Define parameter dimensions
            trans_dim, shape_dim, exp_dim = 12, 40, 10

            # 解析变换矩阵和偏移量 / Parse transformation matrix and offset
            R_ = param[:trans_dim].copy().reshape((3, -1))
            R = R_[:, :3].copy()
            offset = R_[:, 3].copy().reshape((3, 1))

            # 解析形状和表情参数 / Parse shape and expression parameters
            alpha_shp = param[trans_dim:trans_dim +
shape_dim].copy().reshape((-1, 1))
            alpha_exp = param[trans_dim + shape_dim:].copy().reshape((-1, 1))

            # 转换为神经网络张量 / Convert to neural network tensors
            R_tensor = nn.from_numpy(R)
            offset_tensor = nn.from_numpy(offset)
            alpha_shp_tensor = nn.from_numpy(alpha_shp)
            alpha_exp_tensor = nn.from_numpy(alpha_exp)

            return [R_tensor,offset_tensor,alpha_shp_tensor,alpha_exp_tensor]

    # 后处理函数:生成最终的人脸网格数据
```

```
    # Postprocessing function: generate final face mesh data
    def postprocess(self,results,roi):
        with ScopedTiming("postprocess",self.debug_mode > 0):
            # 解析ROI参数 / Parse ROI parameters
            x, y, w, h = map(lambda x: int(round(x, 0)), roi[:4])

            # 将坐标从输入分辨率转换到显示分辨率
            # Convert coordinates from input resolution to display resolution
            x = x * self.display_size[0] // self.rgb888p_size[0]
            y = y * self.display_size[1] // self.rgb888p_size[1]
            w = w * self.display_size[0] // self.rgb888p_size[0]
            h = h * self.display_size[1] // self.rgb888p_size[1]

            # 创建ROI数组并调用后处理函数
            # Create ROI array and call post-processing function
            roi_array = np.array([x,y,w,h],dtype=np.float)
            aidemo.face_mesh_post_process(roi_array,results[0])
            return results[0]
```

Define the main class of 3D face mesh

```
# 3D人脸网格主类，整合了检测、网格生成和后处理功能
# 3D Face Mesh main class, integrating detection, mesh generation and post-processing
class FaceMesh:
    def __init__(self,face_det_kmodel,face_mesh_kmodel,mesh_post_kmodel,det_input_size,mesh_input_size,anchors,confidence_threshold=0.25,nms_threshold=0.3,rgb888p_size=[640,360],display_size=[640,360],debug_mode=0):
        # 初始化三个模型的路径 / Initialize paths for three models
        self.face_det_kmodel=face_det_kmodel        # 人脸检测模型路径 / Face detection model path
        self.face_mesh_kmodel=face_mesh_kmodel      # 人脸3D网格模型路径 / Face 3D mesh model path
        self.mesh_post_kmodel=mesh_post_kmodel      # 人脸3D网格后处理模型路径 / Face 3D mesh post-processing model path

        # 初始化模型参数 / Initialize model parameters
        self.det_input_size=det_input_size          # 人脸检测模型输入尺寸 / Face detection model input size
        self.mesh_input_size=mesh_input_size        # 人脸3D网格模型输入尺寸 / Face 3D mesh model input size
        self.anchors=anchors                        # 检测模型的锚框 / Anchor boxes for detection model

        # 初始化阈值参数 / Initialize threshold parameters
        self.confidence_threshold=confidence_threshold   # 置信度阈值 / Confidence threshold
        self.nms_threshold=nms_threshold                 # 非极大值抑制阈值 / Non-maximum suppression threshold

        # 初始化图像尺寸参数 / Initialize image size parameters
        self.rgb888p_size=[ALIGN_UP(rgb888p_size[0],16),rgb888p_size[1]]    # 输入图像尺寸(16字节对齐) / Input image size (16-byte aligned)
        self.display_size=[ALIGN_UP(display_size[0],16),display_size[1]]     # 显示尺寸(16字节对齐) / Display size (16-byte aligned)
```

```
        self.debug_mode=debug_mode                                      # 调
试模式标志 / Debug mode flag

        # 创建处理实例 / Create processing instances
        # 人脸检测实例 / Face detection instance

 self.face_det=FaceDetApp(self.face_det_kmodel,model_input_size=self.det_input_s
ize,

anchors=self.anchors,confidence_threshold=self.confidence_threshold,

nms_threshold=self.nms_threshold,rgb888p_size=self.rgb888p_size,
                              display_size=self.display_size,debug_mode=0)

        # 人脸网格实例 / Face mesh instance

 self.face_mesh=FaceMeshApp(self.face_mesh_kmodel,model_input_size=self.mesh_inp
ut_size,

rgb888p_size=self.rgb888p_size,display_size=self.display_size)

        # 人脸网格后处理实例 / Face mesh post-processing instance

 self.face_mesh_post=FaceMeshPostApp(self.mesh_post_kmodel,model_input_size=self
.mesh_input_size,

 rgb888p_size=self.rgb888p_size,display_size=self.display_size)

        # 配置人脸检测预处理 / Configure face detection preprocessing
        self.face_det.config_preprocess()

    # 执行推理的主函数 / Main inference function
    def run(self,input_np):
        # 执行人脸检测 / Perform face detection
        det_boxes=self.face_det.run(input_np)
        mesh_res=[]

        # 对每个检测到的人脸执行网格生成 / Generate mesh for each detected face
        for det_box in det_boxes:
            # 配置预处理并执行网格生成和后处理
            # Configure preprocessing and perform mesh generation and post-
processing
            roi=self.face_mesh.config_preprocess(det_box)          # 配置ROI区域 /
Configure ROI area
            param=self.face_mesh.run(input_np)                     # 执行网格生成 /
Perform mesh generation
            tensors=self.face_mesh_post.preprocess(param)          # 预处理网格参数
/ Preprocess mesh parameters
            results=self.face_mesh_post.inference(tensors)         # 执行后处理推理
/ Perform post-processing inference
            res=self.face_mesh_post.postprocess(results,roi)       # 后处理结果 /
Post-process results
            mesh_res.append(res)
        return det_boxes,mesh_res

    # 绘制结果的函数 / Function to draw results
    def draw_result(self,pl,dets,mesh_res):
```

```
        pl.osd_img.clear()    # 清除上一帧的显示内容 / Clear previous frame's
display content
        if dets:              # 如果检测到人脸 / If faces are detected
            # 创建绘图缓冲区 / Create drawing buffer
            draw_img_np =
np.zeros((self.display_size[1],self.display_size[0],4),dtype=np.uint8)
            draw_img = image.Image(self.display_size[0], self.display_size[1],
image.ARGB8888,
                                   alloc=image.ALLOC_REF,data = draw_img_np)

            # 绘制每个人脸的网格 / Draw mesh for each face
            for vertices in mesh_res:
                aidemo.face_draw_mesh(draw_img_np, vertices)
            pl.osd_img.copy_from(draw_img)    # 复制到显示缓冲区 / Copy to display
buffer
```

Execution routine

```
# 执行演示的主函数 / Main function to execute demonstration
def exce_demo(pl):
    global fm
    display_mode = pl.display_mode
    rgb888p_size = pl.rgb888p_size
    display_size = pl.display_size

    # 设置模型路径 / Set model paths
    face_det_kmodel_path="/sdcard/kmodel/face_detection_320.kmodel"
    face_mesh_kmodel_path="/sdcard/kmodel/face_alignment.kmodel"
    face_mesh_post_kmodel_path="/sdcard/kmodel/face_alignment_post.kmodel"
    anchors_path="/sdcard/utils/prior_data_320.bin"

    # 设置基本参数 / Set basic parameters
    rgb888p_size=[640,360]
    face_det_input_size=[320,320]
    face_mesh_input_size=[120,120]
    confidence_threshold=0.5
    nms_threshold=0.2
    anchor_len=4200
    det_dim=4

    # 加载锚框数据 / Load anchor data
    anchors = np.fromfile(anchors_path, dtype=np.float)
    anchors = anchors.reshape((anchor_len,det_dim))

    # 创建人脸网格处理实例 / Create face mesh processing instance

 fm=FaceMesh(face_det_kmodel_path,face_mesh_kmodel_path,face_mesh_post_kmodel_pa
th,

 det_input_size=face_det_input_size,mesh_input_size=face_mesh_input_size,
               anchors=anchors,confidence_threshold=confidence_threshold,

 nms_threshold=nms_threshold,rgb888p_size=rgb888p_size,display_size=display_size
)

    # 主循环 / Main loop
    try:
```

```
        while True:
            with ScopedTiming("total",0):
                img=pl.get_frame()                          # 获取视频帧 / Get video
frame
                det_boxes,mesh_res=fm.run(img)              # 执行推理 / Perform
inference
                fm.draw_result(pl,det_boxes,mesh_res)       # 绘制结果 / Draw results
                pl.show_image()                             # 显示图像 / Display image
                gc.collect()                                # 垃圾回收 / Garbage
collection
                time.sleep_us(10)                           # 短暂延时 / Brief delay
    except Exception as e:
        print("人脸网格功能退出")                          # 打印退出信息 / Print exit
message
    finally:
        # 清理资源 / Clean up resources
        fm.face_det.deinit()
        fm.face_mesh.deinit()
        fm.face_mesh_post.deinit()
```

Face 3D recognition networks have a wide range of application scenarios in many fields:

1. Secure Authentication - A more accurate facial recognition system that is harder to spoof than 2D recognition, suitable for high-security places or device unlocking.
2. Medical - Can be used for plastic surgery planning, facial abnormality analysis, and medical research.
3. Augmented Reality/Virtual Reality - Accurately capture user facial expressions to provide a more immersive virtual experience.
4. Smart Retail - Facial analysis of customers can provide personalized shopping experience and recommendations.
5. Sentiment Analysis - Analyze user emotions by capturing subtle changes in facial expressions.
6. Video Games and Entertainment - Mapping the player's facial expressions onto the game character.
7. Telemedicine - Doctors can make initial diagnoses through 3D facial scans.
8. Virtual makeup/glasses try-on - Provides a more accurate virtual try-on experience.