1. 基于GAN的AR阴影生成

5.1 案例简介

增强现实（Augmented Reality，以下简称AR）技术将计算机生成的虚拟信息如图片、文字与三维物体等无缝地与真实环境进行叠加融合。其中，AR的光照一致性要求被插入物体能呈现出与背景真实环境相一致的明暗及其四周的阴影效果，与真实环境相一致的阴影使得合成图像更具有真实感（如图5-1）。



图 5-1 虚拟物体阴影使合成图像更具有真实感

本章案例属于AR阴影生成应用。旨在华为Atlas 200开发者套件（Atlas 200 Developer Kit，以下简称Atlas 200DK）上实现基于对抗生成网络（Generative Adversarial Network，以下简称GAN）的端到端AR阴影生成。该应用通过输入不含虚拟物体阴影的合成图像与虚拟物体掩码图像，输出包含虚拟物体阴影的图像。由于AR应用的实时性要求，网络模型推断的速度对于用户体验至关重要。如何得到速度快、参数少的阴影生成模型是一项富有挑战性的任务。

本案例完成的系统在华为Atlas 200DK上实现了基于GAN的AR阴影直接生成而不需要任何的逆渲染过程，便于灵活地运用到各种AR场景。

5.2 系统总体设计

本系统读取本地图像数据作为输入，由GAN网络推断生成虚拟物体的阴影，输出包含虚拟物体阴影的图像。整个系统的实现分为三个子模块：

(1)数据收集与数据集制作；

(2)网络模型定义与训练；

(3)网络模型的移植与推断。

三个模块的功能互相独立但通过数据与模型互相关联：模块(2)使用模块(1)输出的数据完成训练，模块(3)使用模块(2)训练好的网络模型完成移植与推断。

5.2.1 功能结构

基于GAN的AR阴影系统可以划分为数据集制作、网络模型构建、网络推断三个主要子模块。其中数据集制作需要完成原始图像采集、相机与光照标定、虚拟物体渲染等；网络模型构建需要完成网络模型定义、网络模型训练等；网络推断需要完成图像数据读取、图像数据处理、阴影图像生成等。为了说明各模块之间的结构关系，细化的系统整体功能结构图如图5-2所示。

基于GAN的AR阴影生成系统

数据集制作

网络模型构建

网络推断

原始图像采集

相机与光照标定

虚拟物体渲染

网络模型定义

网络模型训练

图像数据读取

阴影图像生成

图像数据处理

图 5-2 系统整体功能结构图

5.2.2 运行流程

按照运行流程划分，系统分成两个阶段，分别是网络模型训练阶段和网络推断阶段，如图5-3所示。训练阶段使用自主收集制作的数据集，将其适当地划分为训练集与测试集两部分。基于TensorFlow框架（1.12.0 版本）定义网络模型的结构与训练参数，本案例选用Phillip Isola等人2017年于文献[1]中提出的pix2pix网络作为基本架构进行适当的修改调整；接着进行网络的训练，得到能够实现阴影图像生成的网络模型；最后通过验证环节评估模型的质量，选择最合适的模型。

网络推断阶段主要包括5个步骤：

1. 数据读取：读取无虚拟物体阴影的图像与虚拟物体的掩码图像数据。
2. 归一化：转换图像尺寸并将其归一化到[-1.0, 1.0]的32位浮点数类型。
3. 封装：将图像数据封装成网络模型支持的batch作为网络模型的输入。
4. 阴影生成：网络模型生成包含虚拟物体阴影的图像数据作为输出。
5. 反归一化：将输出数据转换成[0, 255]的8位无符号整数类型并保存。

其中，归一化与封装这两个步骤在训练阶段和推断阶段都需要执行，基于公式：

*I’* = 2.0 \* (*I* / 255.0) – 1.0 (1)

实现。反归一化步骤为公式(1)的逆过程，在推断阶段保存结果时需要执行而训练阶段不需要。推断阶段输出的结果可保存或展示。

数据归一化

数据封装

网络结构

定义

网络模型

训练

测试评估

输入图像

数据归一化

数据封装

阴影生成

反归一化

输出图像

模型转换

**训练阶段（TensorFlow运行主机）**

**推断阶段（Atlas 200DK开发者板）**

图 5-3 系统流程图

整个系统的运行环境分为两部分，数据集制作与网络模型训练在主机端完成，网络推断在Atlas 200DK端完成。数据集制作要求运行环境支持OpenGL渲染流程，网络模型训练要求运行环境安装TensorFlow框架（版本1.12.0），推荐使用工作站配置计算加速卡。推断阶段运行在Atlas 200DK环境，能够支持卷积神经网络的加速。推断阶段所需要的GAN网络模型由训练阶段提供，并根据需要进行格式转换或加速重构生成Ascend 310支持的OM模型文件。

5.3 系统设计与实现

本节详细介绍系统的设计与实现。5.3.1节详细介绍案例的设计思想与AR阴影数据集的制作过程；5.3.2节讲述图像的预处理步骤，涉及像素值与尺寸的归一化；5.3.3节详细介绍如何基于TensorFlow（1.12.0版本）框架定义并训练AR阴影生成的GAN网络模型；5.3.4节详细介绍如何实现将TensorFlow保存的ckpt文件逐步地转换成Ascend 310支持的网络模型；5.3.5节则详细介绍如何用转换后的模型实现网络推断并将推断的结果保存为图像。

5.3.1 数据集制作

由于现有的开源数据集不满足AR阴影生成的需求，我们自主收集并制作相应的数据集以此完成网络模型的训练与测试。本案例的核心设计思想是：让GAN网络通过有监督方式利用AR图像中的真实环境线索学习自动推断虚拟物体的阴影。本案例定义的真实环境线索为真实阴影与相应的投射物，两者缺一不可，但可以部分地出现在图像中。为此，我们需要三种类型的图像数据，如图5-4所示：无虚拟物体阴影的图像（左），虚拟物体的掩码图像（中），包含虚拟物体阴影的图像（右）。注意，图像中必须有真实阴影与其投射物出现。

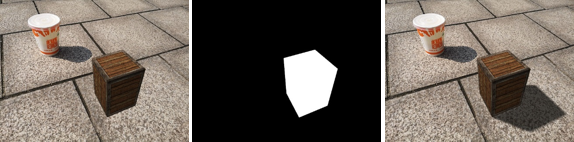


图 5-4 数据样例

无阴影图像与掩码图像作为网络模型的输入，阴影图像为网络模型学习生成的目标。掩码图像的作用主要是指明虚拟物体的其形状与目标阴影形状有一定的关联性。

制作数据集需要准备的条件如下：

1. 焦距固定的摄像头，本案例使用一个焦距为680像素的Logitech C920摄像头以640x480的分辨率拍摄原始图像。
2. 支持基于可编程着色器的OpenGL（OpenGL版本不低于3.3）的PC机。
3. PC机需要具备C++版开源库OpenCV、assimp与freeglut的运行环境。
4. 3D模型文件若干。
5. 方形黑白标记物，如图5-5。

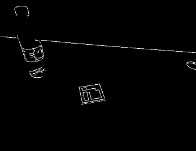


图 5-5 制作数据集所使用的方形黑白标记物

通过标记物对相机进行位姿估计与光照标定的原理为：以标记物的边为坐标轴，建立3D笛卡尔坐标系M，显然标记物4个角点的3D坐标是已知的。在拍摄的图像中检测与识别标记物可以计算出标记物4个角点在图像中的像素坐标。有了这4个点像素坐标与3D坐标之间的对应关系，可以结合相机内参数计算出当前相机在M坐标系中的位置与姿态。

假定室内光源为最主要的单个点光源，室外光源（太阳光）为无限远的点光源，即方向光源，可在M坐标系下测量光源的位置与方向信息。针对室内环境，可通过逐个关闭或遮挡找出产生最明显真实阴影的光源，并测量其几何中心的位置。针对室外环境，可以用预先测定好的直尺，通过直尺的边缘角点与其影子相应角点之间的匹配计算太阳光在M坐标系中的方向向量。

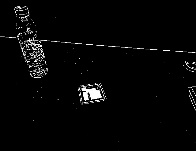
标记物的识别分为图像二值化、轮廓提取、角点检测、匹配识别这四大步骤。完整的流程如图5-6所示。图像二值化利用自适应阈值方法，将图片分为前景和背景两部分。本项目中，图像二值化的目的是为了将黑白方形标记的外观特征从RGB图像中提取出来，将背景统一变为纯色。轮廓提取将二值化图像中的所有的轮廓检测出来并放入列表，进而可以筛选出组成四边形的直线，实现角点的检测与提取。通过四个角点，可将提取得到的四边形用透视变换（Perspective Transformation）恢复为正方形，经过0/1编码后与黑白方形标记物的编码进行匹配。若两者编码的海明（Hamming）距离满足一定的阈值条件，则说明从图像中成功识别出了一个黑白方形标记物。黑白方形标记物的0/1编码为一个7x7的矩阵，如图5-7所示。



二值化

轮廓提取

角点检测



匹配识别

图 5-6 标记物识别流程图

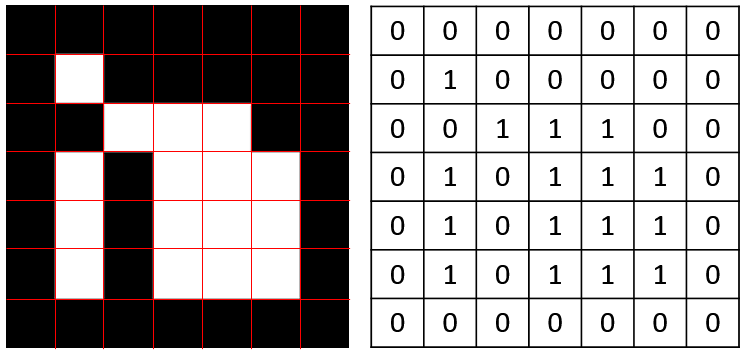


图 5-7 标记物编码示意图

标记物的检测与识别提供了4个角点的坐标对应关系，从而可以通过PnP算法计算出相机位姿与相机内参数。

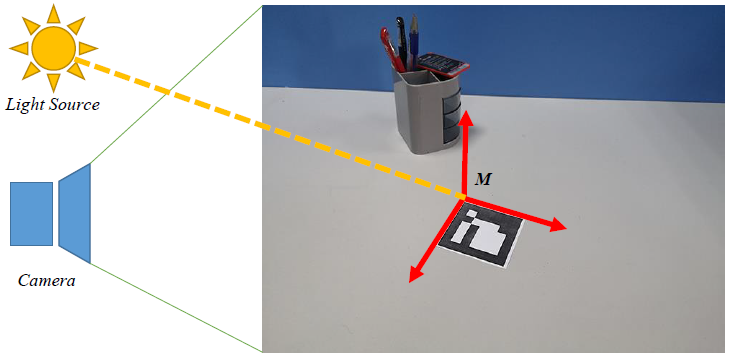


图 5-8 基于标记物制作数据

如图5-8，有了相机位姿与光照这些信息，即可用OpenGL渲染3D模型与阴影。AR合成图像通过以真实照片作为背景，以3D模型为前景的方式渲染得到。3D模型的阴影通过如下方式渲染得到：

1. 在3D模型底部放置一个与模型底部对齐的平面；
2. 利用ShadowMapping方法确定3D模型的阴影区域，该方法的核心思想为：从光源视角发出视线，不可见区域即认为属于阴影区域；
3. 设置平面的透明度（alpha）参数，非阴影区域设置为0.0（完全透明），阴影区域设置为某个位于(0.0, 1.0)区间的浮点数（半透明）；
4. 前景（3D模型与其阴影区域）与背景（真实照片）通过alpha混合得到具有虚拟阴影效果的图像。

渲染得到的包含虚拟阴影AR图像将作为本案例监督学习的Ground Truth被保存下来。通过控制是否投射阴影即可获得有虚拟阴影与无虚拟阴影的AR图像对。通过控制背景（原始图像）为黑色，前景（3D模型）为白色即可获取精准的虚拟物体掩码图像。大部分的渲染工作由OpenGL完成，我们直接提供数据制作工具，读者不需要详细掌握大量的细节知识。

数据集制作的整个过程涉及大量的三维重建与计算机图形学知识，这些知识不便在此展开，如果读者对原理感兴趣可自行下载源代码，结合注释并阅读相关的参考文献[2,3]学习。我们将自主收集制作的数据集划分为两部分：1500用于训练，200用于测试。

由于数据集制作流程较为复杂繁琐且需要受到标记物的限制，我们同时提供另一种无需标记物的数据制作工具方便读者制作自己的合成图像数据。我们将相机与标记物之间的相对位姿与投影矩阵设置为满足多数场景的定值，直接编译到可执行文件之中，通过调用可执行文件观察可视化效果进而调节光源位置参数。这种方式可以摆脱标记物的限制。可执行文件的调用命令如清单5-1所示，调用效果见图5-9，详细操作方法请参考我们提供的工具使用手册。

清单5-1 图像合成工具调用命令

ImageComposite[.exe] <imageFile> <modelFile> <configFile>

<imageFile>: 图像文件路径

<modelFile>: 3D模型文件路径

<configFile>: 配置文件路径，包括光源位置、模型缩放、位姿等参数

调用示例: ImageComposite.exe 007.jpg bunny/bunny.obj light.txt

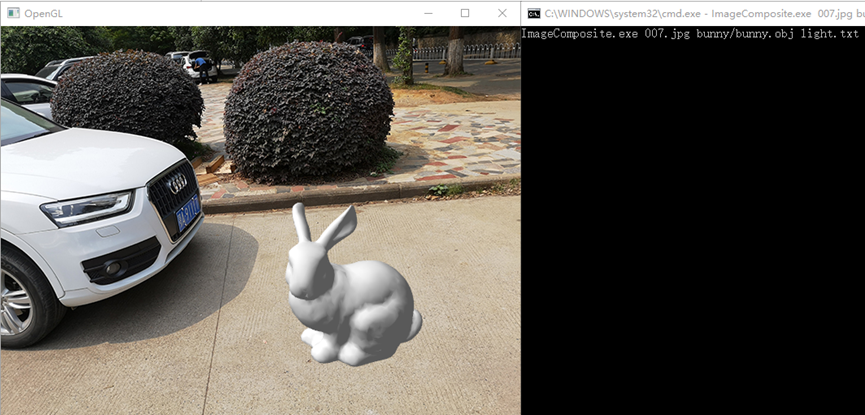


图 5-9 图像合成工具调用示例

本案例提供两种数据制作工具的源代码与可执行文件，读者可根据需要自行下载阅读或根据需求选择使用，如有特殊需求也可以修改编译。

5.3.2 图像预处理

图像的预处理操作包括尺寸与像素值的归一化，这些操作在网络模型训练与推断阶段都会执行。首先需要将图像尺寸缩放到网络支持的256x256尺寸，本案例用双三次插值实现图像的缩放。再根据5.2.2节的公式(1)完成像素值的归一化。通过调用numpy与opencv-python库封装的接口函数实现图像的预处理操作，具体操作如清单5-2所示。

清单5-2 图像预处理操作，详见train.py与test.py

import cv2 as cv

import numpy as np

# 以3通道BGR格式读取输入合成图像，以单通道灰度格式读取输入掩码

image = cv.imread(image\_path, cv.IMREAD\_COLOR)

mask = cv.imread(mask\_path, cv.IMREAD\_GRAYSCALE)

# 用双三次插值方法把图像缩放到256x256

image = cv.resize(image, (256, 256), interpolation=cv.INTER\_CUBIC)

mask = cv.resize(mask, (256, 256), interpolation=cv.INTER\_CUBIC)

# 根据公式(1)将像素值归一化到[-1.0, 1.0]，掩码图像取反

image = image.astype(np.float) / 127.5

mask = 1.0 - mask.astype(np.float) / 127.5

5.3.3 模型创建

AR阴影生成通过GAN网络学习完成。生成器是一个由5个下采样层与5个上采样层构成的U网络，每个下采样层由“卷积-ReLU激活-平均池化”操作序列组成，每个上采样层由“最邻近插值-卷积-ReLU激活”组成。每个上采样层的输出直接连接到对称的下采样层的输入。生成器的输出为残差式（Residual）结构，生成器直接生成阴影残差图，与输入图像相加得到最终的AR阴影图像。判别器则由4个下采样层组成，最后一层输出的特征图经过全局平均池化作为判别器输出的概率。网络结构定义如清单5-3所示，网络结构示意图如图5-10。

清单5-3 网络结构定义，详见model.py

import tensorflow as tf

import tf.image.resize\_nearest\_neighbor as resize

# 对卷积操作的简单封装，详见operators.py

def conv(input\_data, # 输入4维图像数据，NHWC格式的张量

output\_channels, # 输出通道数，即卷积核个数

k=3, # 卷积核的尺寸

s=1, # 卷积操作的步长

padding='SAME', # 填充方式，'SAME'或'VALID'

initializer=tf.keras.initializers.he\_normal()):

...

# 对最大池化操作的简单封装，详见operators.py

def max\_pool(input\_data):

...

# 对平均池化操作的简单封装，详见operators.py

def avg\_pool(input\_data):

...

#生成器定义

def generator(input\_image, input\_mask):

with tf.variable\_scope('generator', reuse=tf.AUTO\_REUSE):

output = tf.concat([input\_image, input\_mask], axis=3)

channels = [32, 64, 128, 256, 512] # 每层输出通道数

n = len(channels) # 5个上采样与下采样层

outputs = [] # 每个上采样测输出的特征图

for i in range(n): # 下采样层

with tf.variable\_scope('enc%d'%i, reuse=tf.AUTO\_REUSE):

output = conv(output, output\_channels=channels[i])

output = avg\_pool(tf.nn.relu(output))

outputs.append(output)

for i in range(n): # 上采样层

with tf.variable\_scope('dec%d'%i, reuse=tf.AUTO\_REUSE):

if i > 0:

output = tf.concat([output, outputs[n-1-i]], axis=3)

h, w = output.get\_shape().as\_list()[1: 3]

output = resize(output, (2\*h, 2\*w)) # 最邻近插值

output = conv(output, output\_channels=channels[n-1-i])

output = tf.nn.relu(output)

with tf.variable\_scope('output', reuse=tf.AUTO\_REUSE):

output = conv(output, output\_channels=3, k=1)

# 输出为残差式结构，阴影残差图与原图相加为最终结果

return tf.nn.tanh(input\_image + output)

# 判别器定义

def discriminator(input\_image, input\_mask, output):

with tf.variable\_scope('discriminator', reuse=tf.AUTO\_REUSE):

channels = [64, 128, 256, 512]

output = tf.concat([input\_image, input\_mask, output], axis=3)

for i in range(4):

with tf.variable\_scope('conv%d'%i, reuse=tf.AUTO\_REUSE):

output = conv(output, output\_channels=channels[i])

output = max\_pool(tf.nn.relu(output))

# 全局平均池化作为判别器的输出

return tf.reduce\_mean(tf.nn.sigmoid(output))

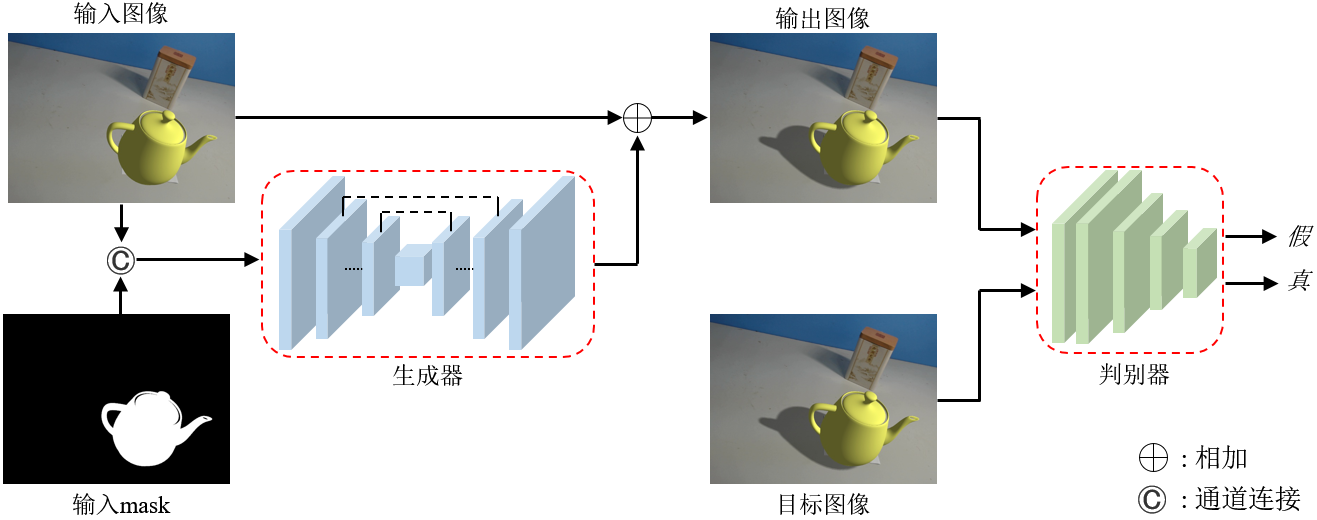


图5-10 GAN网络结构图

GAN网络训练所用的损失函数由三项组合而成，可为每一项赋值不同的权重选择性能最佳的配置：

其中，*L*1表示生成器输出图像与目标图像之间的1-范数误差：

*Lgrad*表示生成器输出图像与目标图像之间图像梯度的1-范数误差：

*Ladv*表示GAN网络的生成对抗误差，我们采用最小二乘生成对抗网络（Least Squares GANs，简称LSGANs）的损失函数作为生成对抗误差：

训练过程中，判别器尽可能使*Ladv*最小化，而生成器尽可能使*Ladv*最大化。损失函数的实现见清单5-4。

清单5-4 损失函数的实现，详见train.py

from tensorflow.losses import absolute\_difference as MAE

# 生成器输出

output\_image = generator(input\_image, input\_mask)

# 判别器输出

real = discriminator(input\_image, input\_mask, target\_image)

fake = discriminator(input\_image, input\_mask, output\_image)

# 计算输出图像与目标图像梯度

real\_grad = tf.image.image\_gradients(target\_image)

fake\_grad = tf.image.image\_gradients(output\_image)

# loss\_L1为1-范数误差，loss\_grad为图像梯度误差

loss\_L1 = MAE(output\_image, target\_image)

loss\_grad = MAE(real\_grad[0], fake\_grad[0])

+ MAE(real\_grad[1], fake\_grad[1])

# 优化判别器最小化loss\_D

loss\_D = tf.square(1.0 - real) + tf.square(fake)

# 优化生成器最小化loss\_G

loss\_DG = tf.square(1.0 - fake)

loss\_G = loss\_L1 + loss\_grad + loss\_DG

由于本案例提供的自主收集数据数据量不大，故可采用把所有图像数据一次性全部读入内存的方式减少磁盘读写带来的开销。训练结束后可直接在测试集上进行网络模型验证。在每个迭代（iteration）中，Adam优化器交替地优化判别器与生成器，初始学习率设置为10-5，batch size设置为1，GAN网络在训练集上训练100轮（epoch）后停止训练。

AR阴影生成的整体质量可通过RMSE图像相似度指标衡量。训练好的网络模型在测试集上计算生成图像与目标图像的平均RMSE，一般来说，RMSE越低，AR阴影生成的质量越好，反之，RMSE越高则AR阴影生成质量越低。本案例量化指标见表格1，典型的可视化结果如图5-11。

表1 网络模型在测试集上的平均RMSE指标

|  |  |
| --- | --- |
| Epoch | RMSE |
| 50 | 7.729 |
| 100 | 6.872 |

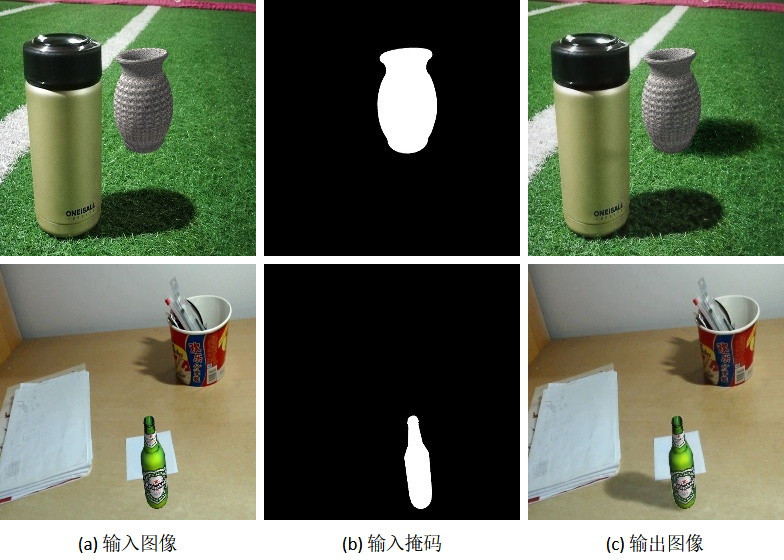


图 5-11 可视化测试结果示意

5.3.4 模型转换

TensorFlow为了方便训练而直接保存网络结构与权重数据分离的ckpt文件，因此首先要将ckpt文件转换成OMG工具支持转换的pb文件，可运行convert.py完成转换。将ckpt文件转换为pb文件的实现如清单5-5所示。

清单5-5 ckpt文件转pb模型，详见convert.py

from tensorflow.python.framework import graph\_util

...

# 定义了计算图

output\_image = generator(input\_image, input\_mask)

# 模型转换

config = tf.ConfigProto()

config.gpu\_options.allow\_growth = True

with tf.Session(config=config) as sess:

init\_op = tf.global\_variables\_initializer()

sess.run(init\_op)

ckpt = tf.train.latest\_checkpoint(ckpt\_dir) # ckpt保存目录

saver.restore(sess, ckpt) # 从ckpt文件加载模型权重

# 需要指明模型最终输出的节点名称，graph工具将自动推断计算图

output\_node\_names = 'generator/Tanh'

output\_graph\_def = graph\_util.convert\_variables\_to\_constants(

sess=sess,

input\_graph\_def=sess.graph\_def,

output\_node\_names=output\_node\_names.split(',')

)

# 保存pb模型文件

with tf.gfile.GFile('model.pb', 'wb') as f:

f.write(output\_graph\_def.SerializeToString())

值得注意的是，GAN网络的判别器只在训练阶段辅助生成器的学习与训练，在后续的推断阶段是不需要的，因此与判别器相关的输入输出不应出现在最终转换的计算图中。将ckpt文件转换为pb模型之后，从MindStudio IDE的菜单中选择“Tools-> Model Convert”打开OMG模型管理工具。如图5-12，选择刚刚保存的pb网络模型，其输入格式为NHWC。网络推断的Batch Mode参数设置为Static，输入图像设置为[1, 256, 256, 3]，输入掩码设置为[1, 256, 256, 1]。本案例网络输入与输出的格式为32位浮点数类型，不启用Quantization与AIPP而是通过代码实现浮点数转换与归一化再输入网络模型推断。参数设置完成后点击Finish，如果模型转换成功将会看到如图5-13所示的日志，Ascend 310支持的OM模型将被保存到~/modelzoo目录下。

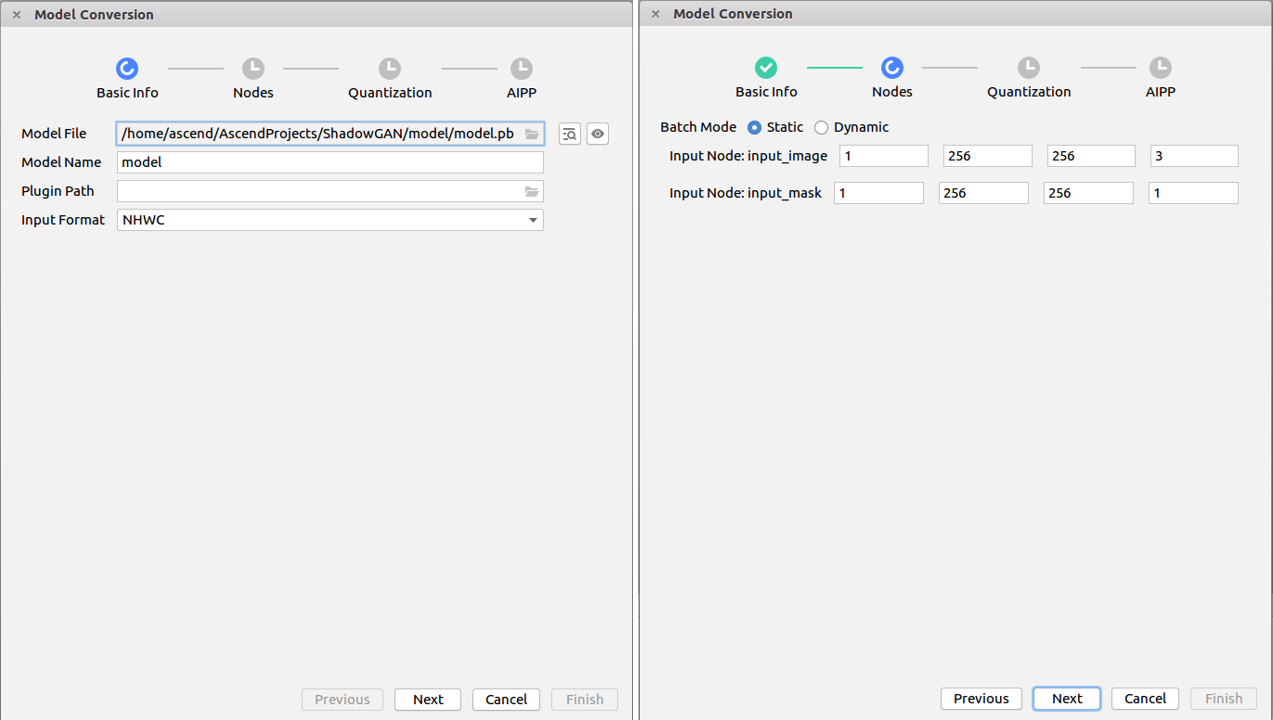


图 5-12 模型转换时的关键配置

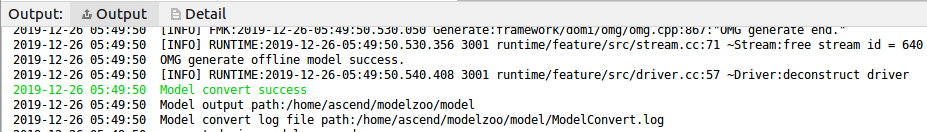


图 5-13 模型转换成功时的输出日志

5.3.5 网络推断

本案例使用python接口开发AR阴影生成的网络推断示例，示例的开发参考了python版的图像分类案例[4]。

首先根据OM模型文件创建计算图并安排网络推断流程，见清单5-6。

清单5-6 创建计算图，详见demo.py

import hiai

# 创建计算图与网络推断流程

def CreateGraph(model):

myGraph = hiai.hiai.\_global\_default\_graph\_stack.get\_default\_graph()

if myGraph is None:

print("[ERROR] Fail to get default graph.")

return None

nnTensorList = hiai.NNTensorList()

#安排网络推断流程

hiai.inference(nnTensorList, model, None)

if hiai.HiaiPythonStatust.HIAI\_PYTHON\_OK == myGraph.create\_graph():

print("[OK] Finish creating graph.")

return myGraph

else:

print("[ERROR] Fail to create graph, please check Davinc log.")

return None

# 根据OM模型创建计算图

omModelName = 'model.om'

inferenceModel = hiai.AIModelDescription('ShadowGAN', omModelName)

myGraph = CreateGraph(inferenceModel)

接下来，读取输入无虚拟阴影AR图像与输入虚拟物体掩码图像，完成浮点格式转换与归一化，如清单5-7所示。

清单5-7 图像数据读取与归一化，详见demo.py

import os

import os.path as osp

import numpy as np

import cv2 as cv

# 读取图像数据并转换为浮点数

def ReadImage(fileName, channels):

if channels != 1 and channels != 3:

print("[ERROR] Wrong image channel parameter.")

return None

if channels == 3: # 以3通道BGR格式读取输入无虚拟阴影图像

image = cv.imread(fileName, cv.IMREAD\_COLOR)

else: # 以单通道灰度格式读取输入虚拟物体掩码

image = cv.imread(fileName, cv.IMREAD\_GRAYSCALE)

# 尺寸缩放为网络支持的256x256

if image.shape[0] != 256 or image.shape[1] != 256:

image = cv.resize(image, (256, 256), cv.INTER\_CUBIC)

if image.ndim == 2:

image = np.expand\_dims(image, axis=2)

# 转换为浮点数返回

return image.astype(np.float32)

# 读取图像与掩码

inputImage = ReadImage(osp.join('noshadow', 'demo.jpg'), channels=3)

inputMask = ReadImage(osp.join('mask', 'demo.jpg'), channels=1)

# 归一化到[-1.0, 1.0]

inputImage = inputImage / 127.5 - 1.0

inputMask = 1.0 - inputMask / 127.5

然后归一化的图像数据需要转换成hiai接口函数支持的数据格式，形成输入网络模型的batch。注意，hiai.NNTensor() 只支持 NCHW的数据作为输入，因此需要用numpy. transpose().copy()接口函数对数据进行格式转换，具体实现如清单5-8所示。

清单5-8 数据格式转换与封装，详见demo.py

from hiai.nn\_tensor\_lib import DataType

# 数据类型从HWC转换成CHW

inputImage = inputImage.transpose([2, 0, 1]).copy()

inputMask = inputMask.transpose([2, 0, 1]).copy()

# 把格式转换好的数据封装成TensorList

inputImageTensor = hiai.NNTensor(inputImage,

256, 256, 3, # H, W, C

'input\_image', DataType.FLOAT32\_T,

inputWidth \* inputHeight \* 3

)

inputMaskTensor = hiai.NNTensor(inputMask,

256, 256, 1,

'input\_mask', DataType.FLOAT32\_T,

inputWidth \* inputHeight \* 1

)

nntensorList = hiai.NNTensorList([inputImageTensor, inputMaskTensor])

将封装好的数据输入进行网络推断，把网络输出的数据转换成正确的格式并保存，即完成了网络推断步骤。注意，网络直接输出的数据为一个列表，其第0项是一个NWHC格式的4维数组，但是内存排布是NCHW格式，需要用numpy.reshape()接口将其形状转换成符合内存排布的格式，再保存结果。具体做法如清单5-9所示。

清单5-9网络推断与结果保存，详见demo.py

# 网络推断输出Tensor的列表

def GraphInference(graphHandle, inputTensorList):

if not isinstance(graphHandle, hiai.Graph):

print("[ERROR] graphHandle is not a Graph object.")

return None

resultList = graphHandle.proc(inputTensorList)

return resultList

# 将网络输出Tensor转换格式并保存结果

def SaveResult(resultList, fileName):

# 把网络输出的Tensor转换成NCHW的内存排布

resultArray = resultList[0].reshape([1, 3, inputHeight, inputWidth])

# 把NCHW的数据转换成NHWC格式

resultImage = resultArray[0].transpose([1, 2, 0]).copy()

# 把数据反归一化并转换成无符号8位整数保存

resultImage = ((resultImage + 1.0) \* 127.5).astype(np.uint8)

cv.imwrite(fileName, resultImage)

# 执行网络推断

resultList = GraphInference(myGraph, nntensorList)

# 将网络推断结果保存

SaveResult(resultList, 'demo.jpg')

至此，从输入无虚拟阴影AR图像到包含AR阴影图像的生成的完整网络推断过程就完成了，图像保存在Atlas 200DK开发者板上，可以通过scp等命令拷贝到主机查看生成结果。

5.4 案例系统部署

案例的网络推断过程运行在Atlas 200DK开发者板上。由于本案例是从本地文件读写图像数据的图像生成应用，模块之间的交互与案例部署相对简单。我们首先需要直接登录Atlas 200DK开发者板，为其配置相应的python运行环境并安装hiai与opencv的python接口[5]，然后执行案例的部署工作，即将相关的运行代码与OM模型文件拷贝到开发者板上的案例目录。

为了方便读者部署，本案例的云盘ARShadowDemo目录提供部署脚本deploy.sh。读者可将ARShadowDemo目录下载到Ubuntu主机，在成功连接了开发者板之后通过在终端执行脚本deploy.sh完成案例的部署工作。如果部署工作成功，Atlas 200DK的HwHiAiUser用户$HOME$目录下将会有ARShadowDemo案例目录被创建。完整详细的案例部署以及脚本输入参数请参考“部署指导说明书.doc”文档。本案例系统的数据集制作工具、完整代码、参考模型以及开发指导书等资料请从以下地址下载：

https://pan.baidu.com/s/120f8pHF2\_Tn1AqGcFo35zw (提取码: 2j7b)

* 1. 运行结果

本案例远程执行网络推断可以直接看到可视化结果，其运行流程如下：

1. 将主机上的需要处理的无阴影AR图像与虚拟物体掩码拷贝到Atlas 200DK上的ARShadowDemo目录下;
2. 远程启动案例程序执行网络推断并保存输出结果;
3. 从开发者板把保存的结果拷贝到主机。

为了方便读者直接运行案例，我们提供了远程执行网络推断的demo.sh脚本文件。在完成了部署工作并连接了Atlas 200DK开发者板之后，读者可在Ubuntu主机的ARShadowDemo目录下通过终端执行demo.sh脚本实现远程执行网络推断。如图5-14所示，执行脚本后首先会出现输入图像与掩码的显示窗口，此时可以点击InputImage或InputMask窗口并使其获得焦点，然后按任意键继续网络推断的执行。AR阴影图像生成后会被传回主机显示，其运行效果如图5-15所示，脚本会在终端输出每一步执行成功或失败的信息等日志。demo.sh脚本的详细输入参数请参考“部署指导说明书.doc”文档。

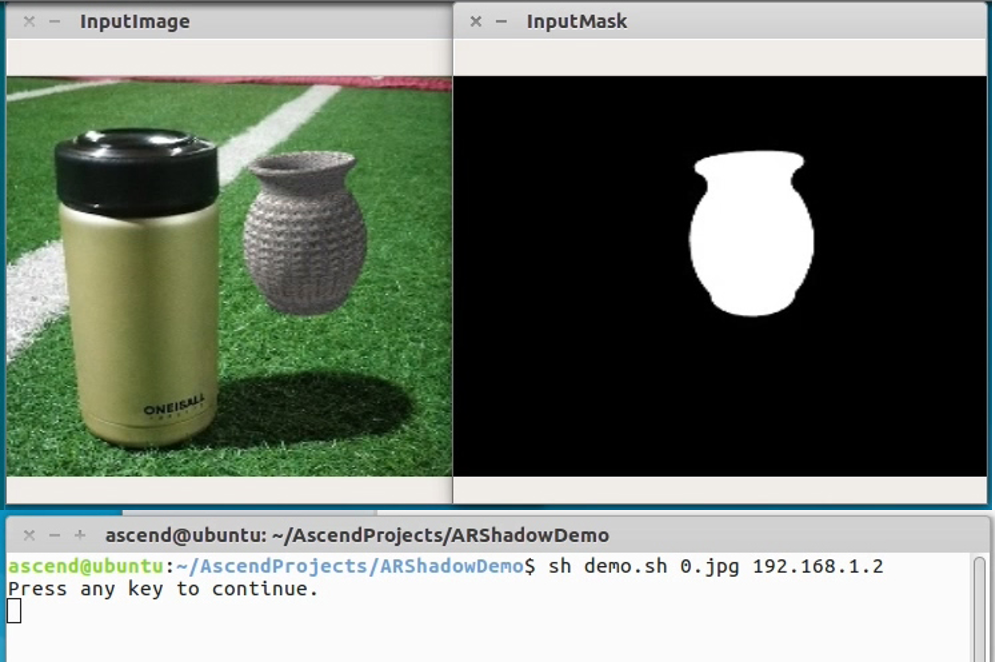


图 5-14 案例远程执行输入显示

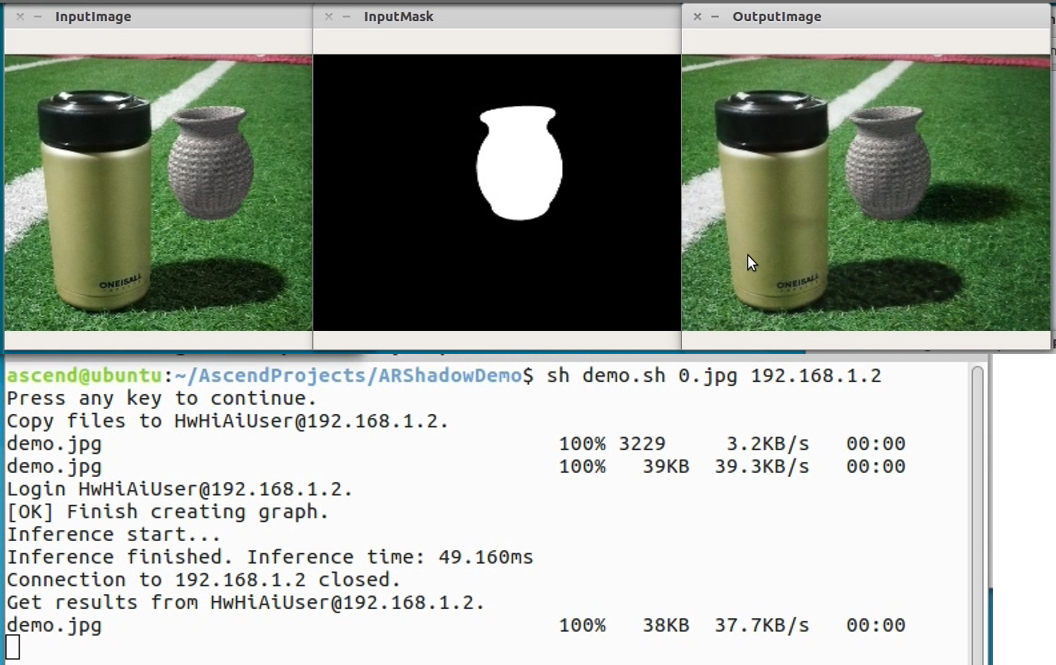


图 5-15 案例远程执行输出效果

最后，我们测算了AR阴影生成在Atlas 200DK上执行网络推断的时间消耗情况。单张图像的AR阴影生成平均时间消耗在45-47ms之间，其速度相比于基于逆渲染的方法要快很多。

5.6 本章小结

本章提供了一个基于华为Atlas 200DK的AR阴影生成案例。该案例利用GAN网络学习生成虚拟物体的阴影。本章详细地介绍了案例的设计思想、AR阴影数据集制作方法、GAN网络搭建与训练、模型转换以及如何移植到Atlas 200DK开发者板等内容。实验证实本案例能够不通过逆渲染，不借助图像以外的任何信息，以较快的速度生成外观较为合理的虚拟物体阴影。读者可以在该案例系统的基础上开发更有针对性的应用系统。

参考资料

[1] Isola P, Zhu J Y, Zhou T, et al. Image-to-image translation with conditional adversarial networks[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. .[S.l.]: [s.n.], 2017:1125–1134.

[2] Hartley, Richard, Zisserman, et al. Multiple view geometry in computer visiond[J]. Kybernetes, 2000, 30(9/10):1865 - 1872.

[3] LearnOpenGL. <https://learnopengl-cn.github.io/>

[4] python版分类示例. <https://gitee.com/Atlas200DK/sample-pythonClassifyDemo>

[5] 开发板hiai与opencv安装. https://bbs.huaweicloud.com/forum/thread-28882-1-1.html