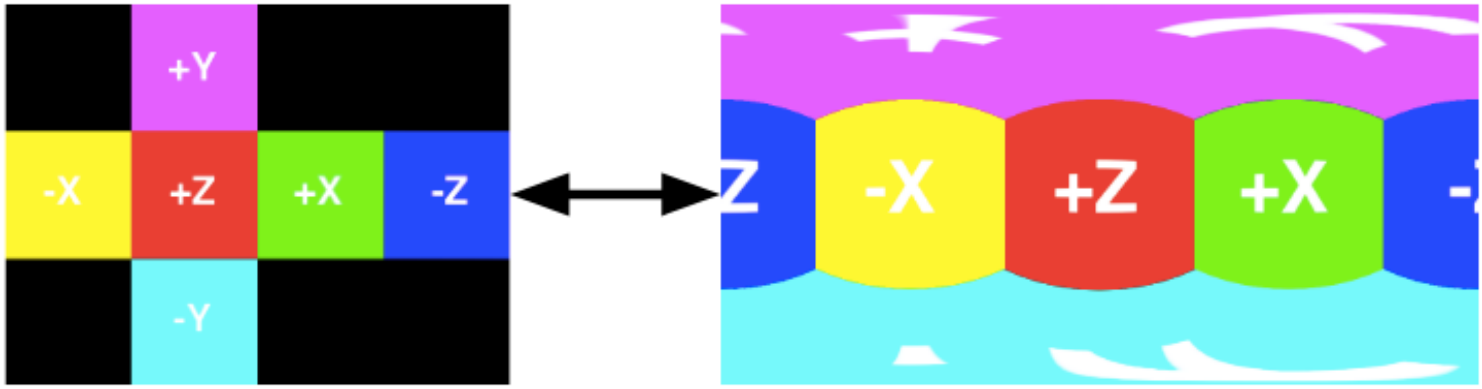
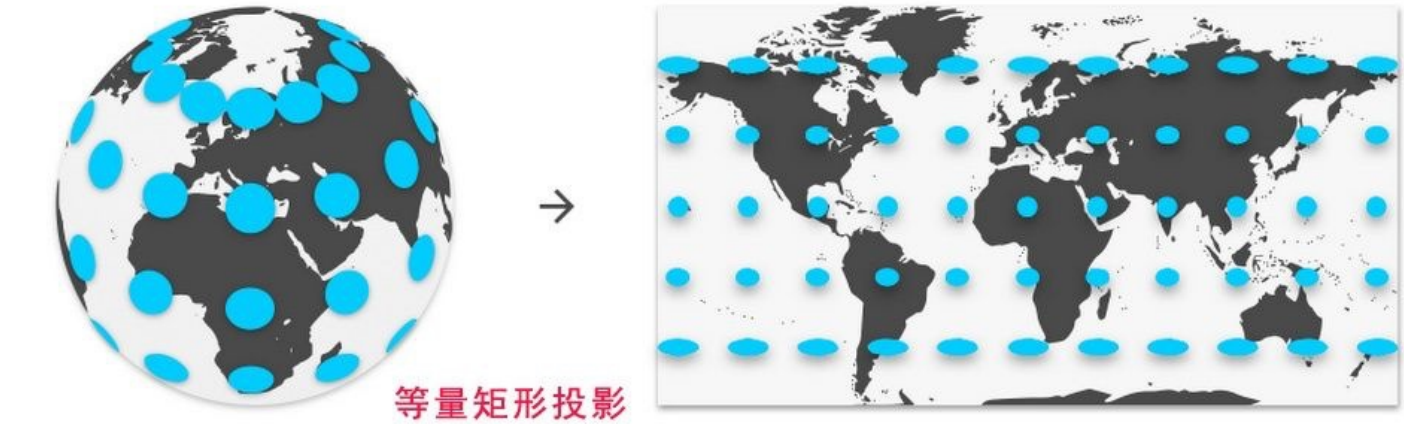
4-16 报告

基于GPU加速的VR视频分割

传统的VR图像分割为立方体贴图。对于等量矩形投影的VR视频，可以通过投影算法将其分割为前、后、左、右、上、下共六块立方体贴图（如图）





Python 实现

Python 提供了名为 `py360convert` 的包来支持图像转换，该包存在一些问题：

1. `py360convert` 完全基于 `NumPy` ，运行在CPU上，速度有限
2. 为了节约时间， `py360convert` 默认限制了输出贴图的尺寸

解决方法

既然没有合适的工具就自己造工具。Python提供了科学计算库 `Numba` 。在 `Numba` 的加持下Pyhon可以利用GPU进行加速计算。这里我参考立方体贴图的公式编写了一套基于双线性插值的投影程序，在GPU上运行。
一些代码（Github上也有）：

- 坐标映射：

```
@cuda.jit ('void(uint8[:,:,:],uint8[:,:,:])')
def project_front(dest, src):
# index pixels
i = cuda.blockIdx.x * cuda.blockDim.x + cuda.threadIdx.x
j = cuda.blockIdx.y * cuda.blockDim.y + cuda.threadIdx.y
k = cuda.threadIdx.z
edge: int = cuda.blockDim.x * cuda.gridDim.x
edge2 = edge * 2
edge4 = edge * 4
# convert pixel position to geological position
x: float = 1.0
y: float = (2 * i / edge) - 1
z: float = 1 - (2 * j / edge)
# convert (x,y,z) to (r, phi, theta)
theta = math.atan2 (y, x)
phi = math.atan2 (z, math.hypot (x, y))
# map to (u,v) coordinates
uf = edge2 * (theta + math.pi) / math.pi
vf = edge2 * (math.pi / 2 - phi) / math.pi
u1: int = int (math.floor (uf))
v1: int = int (math.floor (vf))
mu: float = uf - u1
mv: float = vf - v1
u2: int = u1 + 1
v2: int = v1 + 1
u1 = u1 % edge4
u2 = u2 % edge4
if v1 > edge2 - 1:
    v1 = edge2 - 1
if v2 > edge2 - 1:
    v2 = edge2 - 1
# bi-linear combination
p0: float = src[v1, u1, k] * (1 - mu) * (1 - mv)
p1: float = src[v1, u2, k] * mu * (1 - mv)
p2: float = src[v2, u1, k] * (1 - mu) * mv
p3: float = src[v2, u2, k] * mu * mv
dest[j, i, k] = int (math.floor (p0 + p1 + p2 + p3))
```

- 主进程：

```
import numba.cuda as cuda
import numpy as np
import math
import cv2
from utils.ImageProjectorCubicBilinear import project_front, project_right, proje
ct_back, project_left, project_up, project_down
from utils.ImageProjectorCubicBilinear import ProjectConfig
import time
# filename
video_file_name = 'test_video.mp4'
# generate config with the shape of frame
config = ProjectConfig ((3840, 1920))
# define project functions, CUDA streams, input and output placeholders
function_list = [project_front, project_right, project_back, project_left, projec
t_up, project_down]
stream_list = [cuda.stream () for i in range (6)]
stream0 = cuda.stream ()
imgOut_gpu_list = [cuda.device_array (shape=config.view_shape, dtype=np.uint8, st
ream=stream_list[i]) for i in
                range (6)]
imgOut_cpu_list = [np.zeros (shape=config.view_shape, dtype=np.uint8) for i in ra
nge (6)]
title_list: list = ['front', 'right', 'back', 'left', 'up', 'down']
# Capture the image
cap = cv2.VideoCapture (video_file_name)
start_time = time.time ()
counter = 0
while (cap.isOpened ()):
ret, frame = cap.read ()
frame = cv2.resize (frame, (3840, 1920))
imgIn_gpu = cuda.to_device (frame, stream=stream0)
# stream0.synchronize()
for i in range (6):
    function_list[i][config.grid_dim, config.block_dim, stream_list[i]] (imgOut_g
pu_list[i], imgIn_gpu)
    imgOut_gpu_list[i].copy_to_host (imgOut_cpu_list[i], stream=stream_list[i])
cuda.synchronize ()
# for i in range(6):
#   stream_list[i].synchronize()
# cv2.imshow('front', imgOut_cpu_list[0])
# for i in range (6):
#   cv2.imshow (title_list[i], imgOut_gpu_list[i].copy_to_host (stream=stream_lis
t[i]))
if cv2.waitKey (1) & 0xFF == ord ('q'):
    break
```

```
counter += 1
if (time.time () - start_time) > 1:
    print ("FPS: ", counter / (time.time () - start_time))
    counter = 0
    start_time = time.time ()
cap.release ()
cv2.destroyAllWindows ()
```

经过实验，在输入为4K分辨率（3712×1920）的视频下，算法在同时处理六个面的投影（不写入磁盘，不显示）时可以达到**25FPS**，此时显卡（RTX 2060Super）占用率为**50%**，显存占用率为**300MB**。

如果在主程序中进行保存操作，势必会对效率产生影响，因此需要进行多进程编程。这是具体程序设计的问题，可以往后放。

另一个影响效率的原因是，H.264/H.265视频事宜YUV格式进行编码的，将其转换为RGB色彩空间徒增了许多开销。

头朝向预测

阅读了学长的论文 *A Multi-User 360-Video Streaming System for Wireless Network* ，也看了一些别的论文（包括学长论文的参考文献）

文章提出了两种方法：一种是基于线性回归（Linear regression）的预测，这种办法的延迟在300ms以下。另一种是基于LSTM神经网络的预测。

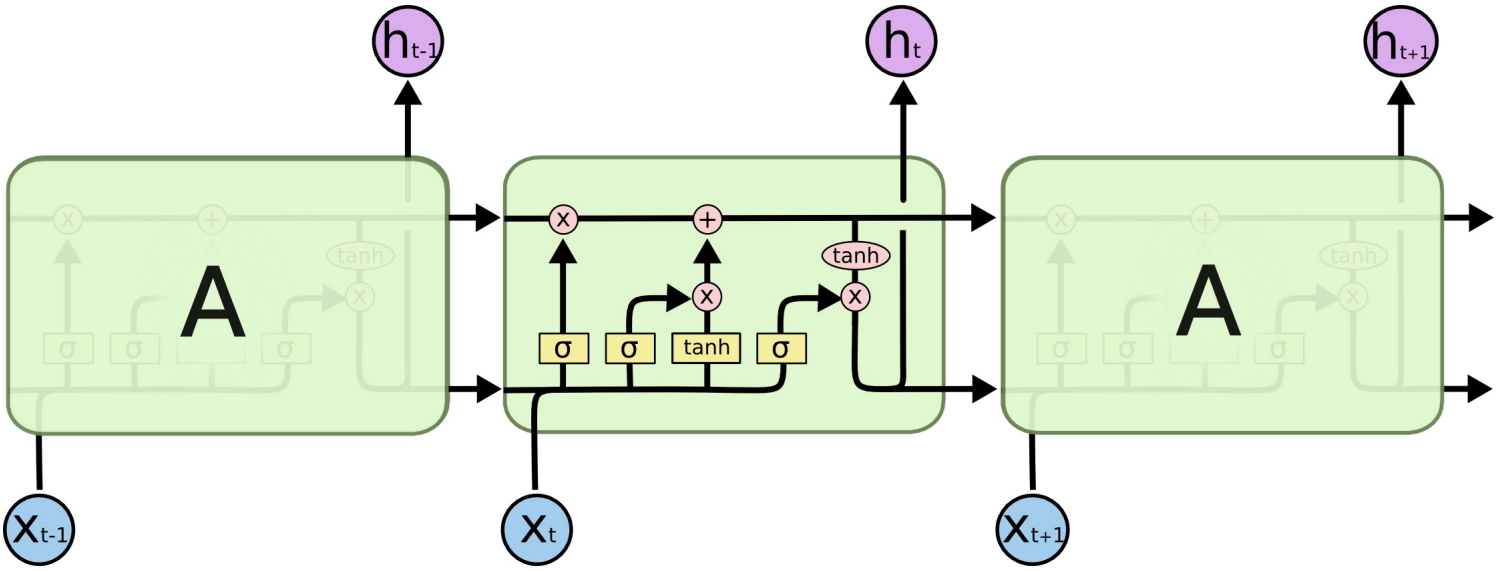
线性回归分析 - 头部运动相关性分析

在学长的论文里没有找到详细的关于线性回归分析的介绍，一下是我阅读有关资料的一些感悟

一般情况下，当用户的期望向某个方向转动时，用户的颈部肌肉会进行复杂的动作。头部不同方向的运动之间应该存在耦合。因此可以进行相关性分析。

LSTM - 模型介绍

这里有一张LSTM模型的示意图



详细介绍在 `LSTM.pdf` 中

模型的输入数据为用户过去2秒中产生的的60条 (r_t, y_t, p_t) 记录。模型预测用户100ms后的头部朝向。判断标准为 **root mean square error** 和 **failure rate**。

读后感

- 想要学习一下LSTM / 线性回归两种方法，尤其是他们的时间成本。
- ~~想要看一下代码的具体实现~~
- ~~用户的头部数据是从哪里来的？是真实的角度数据还是角速度数据~~ 训练是真实值，实际应用可能通过角速度计算。（滤波）
- conjoint view 部分我们貌似用不到，因为是局域网多播multicast实现
- UDP单播是一个可以考虑单选项：UDP意味着可以允许在视频播放的过程中丢帧
- 不同的观测设备有不同的视角
- 相关的算法应该部署在设备端还是服务端呢？

另一种离线预测方法

在研究过程中我发现了这篇论文：*Predicting Head Movement in Panoramic Video: A Deep Reinforcement Learning Approach*

该论文建立了一个收集全景视频注意力的数据库，采集了被试者在全景视频上的注意点。论文的结论是不同受试者的头部移动数据（HM）有高度一致性

据此，论文首次提出应用**深度强化学习（DRL），最大化智能体**与**人类行为的一致性**，进而来预测 HM 位置是一个合理的预测全景视频注意力模型的方法。即一个基于统计规律的、离线的预测系统。这套系统可以在大量用户的注意力分布数据基础上，学习出一套**“人类观众”**的观看规律。

接下来，论文提出了在线-DHP 算法。在线-DHP 算法旨在**根据某个特定观看者的历史头动轨迹**预测其**下一帧的头动位置**。

BASNet