Raw Point Cloud to Depth Map Conversion

December 11, 2021

[ ]:

```
import numpy as np
from scipy.interpolate import griddata
from tqdm import tqdm
import cv2
import os
import scipy
import skimage
import numpy as np
from scipy.sparse.linalg import spsolve
from PIL import Image
import matplotlib.pyplot as plt
from skimage.color import rgb2gray
from numba import jit
from glob import glob
import argparse


parser = argparse.ArgumentParser(description='Generating Depth Maps')
parser.add_argument('--path', metavar='path', default=None,
                    help='Path of the directory containing the Images and the␣
 ↪Point Clouds')


# Code adapted from various sources.

def fill_depth_colorization(imgRgb=None, imgDepthInput=None, alpha=1):
    imgIsNoise = imgDepthInput == 0
    maxImgAbsDepth = np.max(imgDepthInput)
    imgDepth = imgDepthInput / maxImgAbsDepth
    imgDepth[imgDepth > 1] = 1
    (H, W) = imgDepth.shape
    numPix = H * W
    indsM = np.arange(numPix).reshape((W, H)).transpose()
    knownValMask = (imgIsNoise == False).astype(int)
    grayImg = rgb2gray(imgRgb)

    vals, rows, cols, numPix = process(imgDepth, indsM, grayImg)
```

```python
    A = scipy.sparse.csr_matrix((vals, (rows, cols)), (numPix, numPix))

    rows = np.arange(0, numPix)
    cols = np.arange(0, numPix)
    vals = (knownValMask * alpha).transpose().reshape(numPix)
    G = scipy.sparse.csr_matrix((vals, (rows, cols)), (numPix, numPix))

    A = A + G
    b = np.multiply(vals.reshape(numPix), imgDepth.flatten('F'))

    new_vals = spsolve(A, b)
    new_vals = np.reshape(new_vals, (H, W), 'F')

    denoisedDepthImg = new_vals * maxImgAbsDepth

    output = denoisedDepthImg.reshape((H, W)).astype('float32')

    output = np.multiply(output, (1-knownValMask)) + imgDepthInput

    return output

@jit (nopython = True)
def process(imgDepth, indsM, grayImg):
    (H, W) = imgDepth.shape
    numPix = H * W
    winRad = 1
    len_ = 0
    absImgNdx = 0
    len_window = (2 * winRad + 1) ** 2
    len_zeros = numPix * len_window

    cols = np.zeros(len_zeros) - 1
    rows = np.zeros(len_zeros) - 1
    vals = np.zeros(len_zeros) - 1
    gvals = np.zeros(len_window) - 1

    for j in range(W):
        for i in range(H):
            nWin = 0
            for ii in range(max(0, i - winRad), min(i + winRad + 1, H)):
                for jj in range(max(0, j - winRad), min(j + winRad + 1, W)):
                    if ii == i and jj == j:
                        continue

                    rows[len_] = absImgNdx
                    cols[len_] = indsM[ii, jj]
                    gvals[nWin] = grayImg[ii, jj]
                    len_ = len_ + 1
                    nWin = nWin + 1

            curVal = grayImg[i, j]
            gvals[nWin] = curVal
            c_var = np.mean((gvals[:nWin + 1] - np.mean(gvals[:nWin+ 1])) ** 2)

            csig = c_var * 0.6
            mgv = np.min((gvals[:nWin] - curVal) ** 2)
            if csig < -mgv / np.log(0.01):
                csig = -mgv / np.log(0.01)

            if csig < 2e-06:
                csig = 2e-06

            gvals[:nWin] = np.exp(-(gvals[:nWin] - curVal) ** 2 / csig)
            gvals[:nWin] = gvals[:nWin] / np.sum(gvals[:nWin])
            vals[len_ - nWin:len_] = -gvals[:nWin]
            rows[len_] = absImgNdx
            cols[len_] = absImgNdx
            vals[len_] = 1

            len_ = len_ + 1
            absImgNdx = absImgNdx + 1

    vals = vals[:len_]
    cols = cols[:len_]
    rows = rows[:len_]
    return vals, rows, cols, numPix

def resize(img, dmap, new_shape):
    img = cv2.resize(img, new_shape, cv2.INTER_NEAREST)
    dmap = cv2.resize(dmap, new_shape, cv2.INTER_NEAREST)

    return img, dmap
def sortbyName(path):
    if '.npy' in path:
        num = int(path.split(os.path.sep)[-1].split('.npy')[0])
    elif '.jpg' in path and not 'dm.jpg' in path:
        num = int(path.split(os.path.sep)[-1].split('.jpg')[0])
    return num
class generateDepthMaps:
    def __init__(self, path):
        self.currDirectory = path
        self.getImgsDmaps()
        self.generateDmaps()
    def getImgsDmaps(self):
        self.allImgs = glob(self.currDirectory + os.path.sep + '*.jpg')
        self.excludeDmaps = glob(self.currDirectory + os.path.sep + '*dm.jpg')
        if len(self.excludeDmaps) > 0:
            for element in self.excludeDmaps:
                if element in self.allImgs:
                    self.allImgs.remove(element)
        self.allImgs = sorted(self.allImgs, key = sortbyName)
        self.allDmaps = sorted(glob(self.currDirectory + os.path.sep + '*.
 ↪npy'), key = sortbyName)

    def generateDmaps(self):
        for i in tqdm(range(len(self.allImgs))):
            out = fill_depth_colorization(cv2.imread(self.allImgs[i])/255., np.
 ↪load(self.allDmaps[i]))
            outPath = self.allDmaps[i].split('.npy')[0] + 'dm.jpg'
            plt.imsave(outPath, out)
if __name__ == '__main__':
    args = parser.parse_args()
    if args.path == 'none':
        print('Generating Depth Maps for all DIrectories.')
        basePath = r'/media/athrva/New Volume/cis520/Final Project/kitti_256'
        allDirectories = glob(basePath + os.path.sep + '*')
        for k in range(17):
            if not '.txt' in allDirectories[k]:
                _ = generateDepthMaps(allDirectories[k])
    else:
        try:
            _ = generateDepthMaps(args.path)
        except:
            print('Error Occurred. Please check the specified path.')
```