# Summer Undergraduate Research Opportunities

Emile Okada
University of Cambridge

July, 2016

# 1 Week 1

## 1.1 Reading

I started the week reading Chapter 1 section 2 of Charles L. Epstein's "Introduction to the Mathematics of Medical Imaging". It covered how to reconstuct a 2d convex object from the shadows of an object. The idea is fairly straighforward. If $h(\theta)$ is the shadow function as described in the book (essentially the distance of the support line in direction $(-\sin(\theta), \cos(\theta))$ from the origin), then the convex hull can be parameterized by

$$(x(\theta), y(\theta)) = h(\theta) \cdot (\cos(\theta), \sin(\theta)) + h'(\theta) \cdot (-\sin(\theta), \cos(\theta)). \quad (1)$$

This idea can be extended to 3d by considering slices of the object. Fix some vector $\mathbf{v}$ and then consider the collection of planes perpendicular to $\mathbf{v}$. In each of the planes one can use the 2d method to construct a 2d convex hull of the intersection of the object with the plane. Stringing all these 2d slices together then gives a rough reconstruction of the 3d object from its shadows.

I also spent some time reading up on the TV transform and scale spaces, to get a rough idea of which project I'd like to do. I ended up going with the tomography project, but spend roughly 1.5 days doing reading for the other project.

### 1.1.1 Remaining tasks

The above method only allows for 'slicewise convex' reconstructions. While better than convex, this doesn't ustilize all the data available from the shadows e.g. see the rabit ears below. We know the two ears are two separate 'blobs' and we can tell this from the shadow, but this information is nonetheless lost in the reconstruction. I need to try to find ways on how to improve on this.

## 1.2 Coding

On Thursday I started coding. To have some examples to work with I used the Wolfram Mathematica ExampleData function to obtain a 3D model of a rabit.

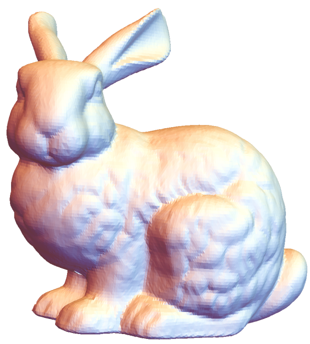

Figure 1: 3D model of a rabit from Mathematica ExampleData.

I then took its orthogonal projection and binarized the image to obtain shadows of the rabit from 10 different angles.

Figure 2: Shadow of rabit with $\theta = 0$

Figure 3: Shadow of rabit with $\theta = \pi/2$

I then finished writing the code for reconstructing the object from its shadows in python (see page 5) on Friday.

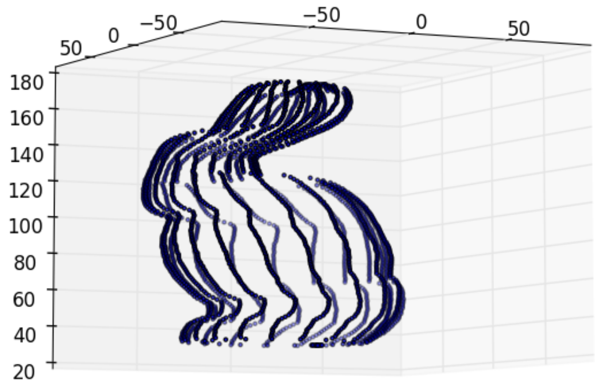


Figure 4: Reconstruction of the rabit using the script on page 5.

### 1.2.1 Challenges

I had some problems originally with taking the derivative of the shadow function since the input was a bit noisy. To fix this problem I convolved the data with a discrete gaussian (binomial distribution) to smooth out the noise. This worked well with the example data. However, it remains to see whether it will work well enough for the actual real life pictures which will probably be a lot more noisy.

### 1.2.2 Remaining tasks

There are currently a lot more data points than I need. At the moment I'm calculating the shadow function fow each row of the image matrix. This results in a lot of points which causes the rendering to take some time. I can probably get away with a lot fewer points in the z direction. I also plan on adding a polygonal mesh so it looks a bit nicer. Lastly, I should probably also do some smoothing in the z direction to fix the sudden cutoffs that tend to occur when the variations in the 3D object are smaller than the 'resolution' in the z-direction.

## 1.3 Building

I haven't begun building yet, but I now have a rough idea of the teh set should look like. At the moment I'm thinking of creating a simple turn table (rotating platform) on which a person stands. I'll then place some bright lights in front for them and a big white sheet of paper behind on which to shine their shadow. A camera is then placed behind the paper to capture the shadows.

### 1.3.1 Remaining tasks

I'll create a proper design this week and make a basic prototype on a smaller scale hopefully by next week.

# 2 Code

## 2.1 3D reconstruction from shadows

```
from __future__ import division
from PIL import Image

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

import math
import re
import os
import misc

class image3d:
    def __init__(self,files):
        (self.width, self.height) = Image.open(files[0]).size
        self.resolution = len(files)
        #self.aspect = self.width/self.height
        self.aspect = 1
        self.shadow_list = [[] for i in range(self.height)]
        self.x, self.y, self.z = [], [], []

        #Determine the shadow function
        for f in files:
            img = Image.open(f)
            img = self.preprocess(img)
            img_data = self.img_to_array(img)
            for i, row in enumerate(img_data):
                (first, last) = self.read_value(row)
                self.shadow_list[i].insert(len(self.shadow_list[i])//2,first)
                self.shadow_list[i].append(last)

        for i in range(self.height):
            #Smooth the data by convolving with discrete Gaussian (i.e. binomia
            self.shadow_list[i] = misc.list_convolve(self.shadow_list[i],misc.b

            #Add points
            coords = misc.transpose(self.convex_hull(i))
            self.x.append(coords[0])
            self.y.append(coords[1])
            self.z.append(coords[2])

        #Create figure
        fig = plt.figure()
        ax = fig.add_subplot(111,projection='3d')
        ax.scatter(self.x,self.y,self.z,marker='.')
        plt.xlim([-90,90])
```

```
        plt.ylim([-90,90])
        plt.show()

    @property
    def coordinates(self):
        return [self.x, self.y, self.z]

    def preprocess(self,img):
        """Convert image to greyscale and binarize image"""

        img = img.convert('L')
        out = img.point(lambda i: 255 if i>255/2 else 0)
        return out

    def img_to_array(self,img):
        """Convert Image object to matrix"""

        l = list(img.getdata())
        (w,h) = img.size
        return [l[w*i:w*(i+1)] for i in range(h)]

    def read_value(self,img_row):
        """Find shadow function value for a particular row"""

        mid_point = len(img_row)//2
        try:
            first = mid_point - img_row.index(0)
        except ValueError:
            first = 0
        try:
            last = len(img_row) - 1 - img_row[::-1].index(0) - mid_point
        except ValueError:
            last = 0
        return (first, last)

    def convex_hull(self,z_coord):
        """Calculate the points belonging to the convex hull of the particular

        theta = np.linspace(0,2*math.pi,2*self.resolution)
        shadow_derivative = misc.differentiate(self.shadow_list[z_coord], math.
        return [(self.aspect*(h*math.cos(t)-s*math.sin(t)),self.aspect*(h*math.

files = ['./Pictures/'+f for f in os.listdir('./Pictures/')]
files = sorted(files, key=lambda x: int(re.search(r'(\d+)\.jpg',x).group(1)))

space_shuttle = image3d(files)
```

### 2.2 Miscellaneous functions

```
from __future__ import division
```

```
import math

def transpose(l):
    return zip(*l)

def binomial(n,k):
    return math.factorial(n)/(math.factorial(k)*math.factorial(n-k))

def binomial_density(n):
    return [binomial(n,k)/(2**n) for k in range(n+1)]

def dot_product(lista,listb):
    return sum(a*b for a,b in zip(lista,listb))

def differentiate(y, step_size):
    y_temp = y[-1:] + y + y[:1]
    return [(t - s)/(2*step_size) for s, t in zip(y_temp, y_temp[2:])]

def list_convolve(l,kernel):
    left = (len(kernel)-1)//2
    right = len(kernel)//2
    l_temp = l
    if left > 0:
        l_temp = l[-left:] + l + l[:right]
    elif right > 0:
        l_temp = l + l[:right]
    if kernel != []:
        return [dot_product(l_temp[i:i+len(kernel)],kernel) for i in range(len(
    return l
```