CVIT Workshop Day 4 Session 1

Formulate the problem of fingerprint recognition *perfectly.*

You need to learn to formulate the problem perfectly. It takes a lot of effort to perfectly state a problem. But when we solve a new problem, then we should first focus on perfectly formulating the problem.

Different formulations of a problem can seem to be very similar, but they might be very different, one might be very difficult, other might be very easy.

Sometimes it is the smartness in reframing the problem, and you would be able to solve it.

Thus, we need to be very specific in what we are doing.

fuck it was completely wrong.

When you are in the lab, you should interact with others, and you would learn a lot in the process.

disciplined, curious and work on your shortcoming

Sometimes the idea for a new thing may come from a completely different thing, and these lead to some very happy accidents. So always listen, always listen, wherever you are, whenever you are listening someone speak. You never know when something might come useful to you.

3D Computer Vision

Motivations

Usually we represent 2D images with a grid. But how do we represent a 3D image? For differnet use cases, different representations.

The first one is the point cloud representation. If you just have points in a space (their positions in the space). The dimension is nx3.

Voxel is is like a 3D grid. For a particular cell in the grid, whether that value is 0 or 1. It is very discontinuous and jagged (because of the grid). If you increase the size of the grid, then it will be much fine.

The third is the mesh. It is used a lot, because it is efficient, and it can represent almost all the properties of a 3D surface. Mesh is like a graph where the vertices are connected using edges, and there is something called a face. The triange formed by these edges and vertices are called a face.

Implicit representation is gaining very popularity. Usually we represent 2D images with a grid. But how do we represent a 3D image? For differnet use cases, different representations. The first one is the point cloud representation. If you just have points in a space (their positions in the space).

Another representation called the implicit representation is gaining popularity. There are two equatios for a circle. We have in terms of coordinates, and in terms of the parametric coordinates. In the parametric representation, we get the exact points where the circle boundary exists. This is not the case for the coordinate equation, because for that we must **sample a large number of points, and we can find out the points closes to the boundary.** Therefore the coordinate representation is like a distance function.

Now we can use this idea of distance function to define areas where SDF < 0 and where SDF > 0. This would give us very smooth boundaries.

How do we save the object representation? There are a bunch of file formats for saving the 3D mesh. The popular ones are .obj and the .ply .

The obj format is vt x y z. The vt stands for the texture coordinate. .obj is like a text file.

.ply is a binary encoded. It is very efficient for storing the data, but it does not support textured meshses.

Major research areas

3D reconstruction from single image

If we are given a single image of the person, then we can generate a complete 3D mesh of the person. This requires a lot of training data.

NERF extensions are also a very hot topic.

DALL-E uses diffusion. Given a text prompt, you can generate images. Given a text prompt, generate a mesh (a coloured one).

Often the methods above are based on static images. But what if we want to perform the 3D reconstruction from a video of a moving person. There is still some noise in the edges. This is similar to the 3D scan of the body part, where we took the scan from different angles.

Garment extraction from a single image. For example we would like to try on a single garment. For this we can extract the garment worn by a person.

Cloth simulation is another research area. When the pose changes, then wrinkles are formed in the clothes, just like the real life.

UV parameterization and texture mapping; UV parameterization comes from a mapping literature. In mathematics mapping has it's own branch, and it is very vast. This was evolved in the Computer Graphics and Computer Vision community. We want to draw something on a mesh, and it was very difficult, because meshes are curved, and it is difficult to draw on them. So if you want to drawm then you must map it into an image. We can then manipulate it, color some parts of it, etc. We can map the image of the globe to an atlas. This process is UV parameterization. The X axis is U, the Y axis is V. We already have very efficient algorithms for images, and thus we do not need to f we have two point something for processing meshes.

For a closed object, for examlpe the 3DD model of a rabbit, then the UV mapping would require you to cut it from somewhere and then flatten it. The red line in the figure is called a **seem**, and it is the line along which we cut.

Another problem is hand-object interaction. Given a 3D object, and the initial location of my hand, determine how can the hand grasp the object. Note that the hand is not in perfect contact with the object, there is only partial contact, and estimating that contact field is difficult.

Meshlab is the open source system for processing and editing the 3D triangular meshes. It provides a set of tools for editing, cleaning, healing, etc.

Meshlab object

First we initialize a `MeshSet()` object.

To reduce the definition of a mesh, then we can collapse several triangles.

Note that the first loaded meshahas an ID of 0. I am loading the same mesh.

Sampling and surface reconstruction. Suppose that we have a mesh that is not clean, for example, the kinec camera mesh of the moving person, then we need to do some pre processing. So given a mesh, we would like to sample uniformly on that mesh. We construct that mesh with new sample points that are more uniform than the original one.

So like this is based on some observations by author that what we need to find is a funtion, a smooth function that can represent this surface. Note that the normals are exactly perpendicular to the face. We can draw the face normals. A gradient of the function representing the surface should be in the direction of the normal. This was the fundamental observation made by the authors of this paper. A Poisson equation is used. It is a partial differential equation.

$$\nabla X = n$$

Taking gradient on both sides we have:

$$\nabla^2 X = \nabla n$$

which is

$$\Delta X = \nabla n$$

where the gradient of gradient becomes the Laplacian.

Based upon the point cloud that we have, we would calculate a gradient operator. We can find out the derivate of a grid using the first principles. You know just the scalar values of the surfaces because of X. There is an algorithm called **marching cubes** algorithm, we can do a sliding window thing in the grid, and based on the value, we could make a mesh out of it. An implicit function is a function of the form:

$$f(x) = 0$$

We can sample a random point usign this implicit function, and then based on the value of $f(x)$ we can derive which side of the surface the point lies. SDF is an example.

Morphing. We have two meshes, and we can morph the meshes. We can play around with the percentage of morphing. Linear morphing between two meshes.

Meshlab has a GUI, and therefore is very intuitive. Geodesic distance — distance of the path between two points along the surface of a mesh. You cannot use the Euclidean distance for this. Meshlab is very good at this. Meshlab has got almost all the filters (Except a few ones which were introduced in the last 5 - 6 years).

Open3D is good for interactive simulations. You could get a particular vertex, or a particular edge using Open3D. They store the vertices and the edges as an array, which is visible to the user.

Open3D object

We first import all the required libraries.

Then we have a visualizer, which can take a list, the vertices, the edges, etc. and can represent it. This is Trimesh based.

For example if we want to read a point cloud;

same for the triangle meshes;

The .ply file; the `pcd.points` gives the points.The `pcd.colors` gives the color values. Similarly `pcd.normals` gives the normals.

```
print(point.shape, colors.shape, normals.shape)

# three dimensional matrices
```

And we can read a mesh object also:

```
mesh = o3d.io.read_traingle_mesh(os.path.join(src, filename))

vertices = np.asarray(mesh.vertices)
faces = np.asarray(mesh.faces)
colors = np.asarray(mesh.vertex_colors)
normals = np.asarray(mesh.vertex_normals)
print(vertices.shape, faces.shape, colors.shape, normals.shape)
# (15258, 3) (30338, 3) (0, 3) (15258, 3)
```

There is something called the baricentric interpolation. We define the baricenter. r. Let α, β and γ represent the distances of the vertices from the baricenter. All the points inside the triangle can we expressed in terms of the baricenter and these parameters. The color can be defined in terms of a point, or for the whole face.

When we are mapping a 3D object to 2d space. For example we are mapping a cube to 2D image. Think of the UV mapping as if we have a cube, and we unwrap it, and we would have shape like this. When we have this 2D space, then we can draw something on this, and whatever change we make to the UV map, then it would reflect on the original object (while rendering). This is a very usual way of representing the color.

A camera has its intrinsic matrix, which is denoted by k, and it also has its extrinsic matrix, denoted by RT (rotation translation). RT matrix gives the coordinates and the orientation of the camera in space. The k value gives the details about the camera itself, about how the camera is. When we perform the multiplication of these matrices, we can find the exact coordinates where the images would be projected in the world coordinates.

We can get the cloud points from depth by backprojecting the depth values. It is very noisy though, unlike the mesh.

ICP algorithm (Iterative Closest Point algorithm)

If there are two different point clouds (red and blue rabbits in the screen). Across multiple views we get partial point clouds from the various camera, and we need to align them. ICP algorithm does exactlt this. There are bunch of variations of iterative closest point. Optimizing for an R and T. How to perform these operations for matching of one point cloud with another. This is called rigid ICP. It is iterative. You run for some number of iterations, and slowly it converges to the right answer. If you want to learn more about it, there is Open3D documentation which explanis this. What will be the loss function here that we would want to minimize?

We havet he correspondence step and the alignment step. We find the closes point q_i in Q for each point p_j in P. Then in alignment step we update the transformation by minimizing the L2 distance between the corresponding points.

There is a library in Open3D (a function) and we can just use that. We can set multiple variables here, like what kind of ICP we would want.

```
p2p_icp = o3d.pipelines.registration.registration_icp(
# parameters)
```

this is not supposed to happen, probably something that I commented out…

Avinash Sharma lecture

good luck!