6.801/6.866: Machine Vision, Lecture 22

Professor Berthold Horn, Ryan Sander, Tadayuki Yoshitake
MIT Department of Electrical Engineering and Computer Science
Fall 2020

These lecture summaries are designed to be a review of the lecture. Though I do my best to include all main topics from the lecture, the lectures will have more elaborated explanations than these notes. Therefore, if you're looking for the most rigorous review and treatment of these topics, we encourage you to rewatch the lecture videos. With that said, we hope these summaries are beneficial for your learning. If you have any feedback for these lecture summaries, please submit it **here**.

1 Lecture 22: Exterior Orientation, Recovering Position and Orientation, Bundle Adjustment, Object Shape

Today, we will conclude our discussion with photogrammetry with more discussion on **exterior orientation**. We will focus on how exterior orientation can be combined with other photogrammetry techniques we've covered for recovering position and orientation, solving the bundle adjustment problem, and determining optimal representations for object shape for solving alignment and recognition problems.

1.1 Exterior Orientation: Recovering Position and Orientation

Consider the problem of a drone flying over terrain:

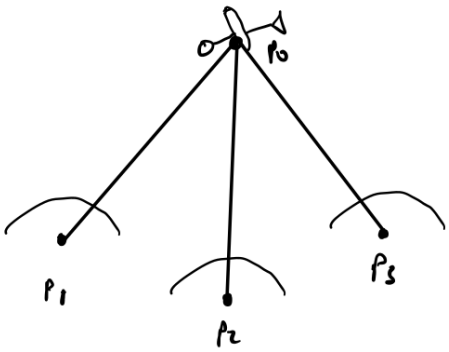

Figure 1: Example problem of recovering position and orientation: a drone flying over terrain that observes points for which it has correspondences of in the image plane. The ground points are given by p_1, p_2, and p_3, and the point of interest is p_0. This photogrammetry problem is sometimes referred to as Church's tripod.

Problem Setup:

- Assume that the ground points p_1, p_2, and p_3 are known, and we want to solve for p_0.
- We also want to find the attitude of the drone in the world: therefore, we solve for **rotation** + **translation** (6 DOF).
- We have a mix of 2D-3D correspondences: 2D points in the image plane that correspond to points in the image, and 3D points in the world.
- Assume that the **interior orientation** of the camera is known (x_0, y_0, f).
- Connect image to the center of projection.

How many correspondences do we Need? As we have seen with other frameworks, this is a highly pertinent question that is necessary to consider for photogrammetric systems.

Since we have 6 DOF with solving for **translation** + **rotation**, we need to have at least 3 correspondences.

1.1.1 Calculating Angles and Lengths

Next, we need to figure out how to solve for the angles between ground points, and the lengths to these ground points from the plane:

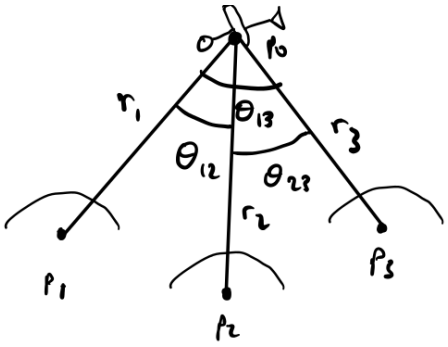

Figure 2: As a next step of solving this exterior orientation problem, we calculate the angles between the ground points relative to the plane, as well as the lengths from the plane to the ground points. The angles of interest are given by: θ_{12}, θ_{13}, and $\theta_{2,3}$, and the lengths of interest are r_1, r_2, and r_3.

- Given the problem setup above, if we have rays from the points on the ground to the points in the plane, we can calculate angles between the ground points using dot products (cosines), cross products (sines), and arc tangents (to take ratios of sines and cosines).
- We also need to know the lengths of the tripod - i.e. we need to find r_1, r_2, and r_3.
- From here, we can find the 3D point p_0 by finding the intersection of the 3 spheres corresponding to the ground points (center of spheres) and the lengths from the ground points to the plane (radii):

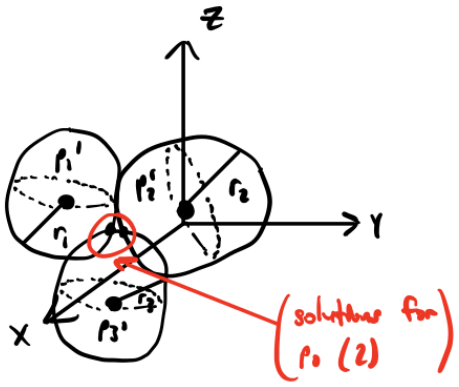

Figure 3: We can find the location of the plane p_0 by finding the intersection of 3 spheres using the point information p_i and the length information r_i.

Note that with this intersection of spheres approach, there is ambiguity with just three points/correspondences (this gives 2 solutions, since the intersection of 3 spheres gives 2 solutions). Adding more points/correspondences to the system reduces this ambiguity and leaves us with 1 solution.

Another solution issue that can actually help us to reduce ambiguity with these approaches is that mirror images have a mirror cyclical ordering of the points that is problematic. This allows us to find and remove "problematic solutions".

What if I only care about attitude? That is, can I solve for only rotation parameters? Unfortunately not, since the variables/parameters we solve for are coupled to one another.

Some laws of trigonometry are also helpful here, namely the law of sines and cosines:

Figure 4: For any triangle, we can use the law of sines and law of cosines to perform useful algebraic and geometric manipulations on trigonometric expressions.

Law of Sines: $\frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C}$

Law of Cosines: $a^2 = b^2 + c^2 - 2bc\cos A$

We do not use these approaches here, but it turns out we can solve for the lengths between the different ground points (r_{12}, r_{23}, and r_{13}) using these trigonometric laws.

1.1.2 Finding Attitude

From here, we need to find **attitude** (rotation of the plane relative to the ground). We can do this by constructing three vectors with the translation subtracted out (as we have seen with previous approaches):

$$\mathbf{a}_1 \Longleftrightarrow \mathbf{b}_1 = \mathbf{p}_1 - \mathbf{p}_0 \tag{1}$$
$$\mathbf{a}_2 \Longleftrightarrow \mathbf{b}_2 = \mathbf{p}_2 - \mathbf{p}_0 \tag{2}$$
$$\mathbf{a}_3 \Longleftrightarrow \mathbf{b}_3 = \mathbf{p}_3 - \mathbf{p}_0 \tag{3}$$

Note that the vectors $\mathbf{a}_1, \mathbf{a}_2$, and $\mathbf{a}_3$ are our correspondences in the **image plane**. This means that we can now relate these two coordinate systems. We can relate these via a rotation transformation:

$$R(\hat{\mathbf{a}}_i) = \hat{\mathbf{b}}_i \; \forall \; i \in \{1, 2, 3\}$$

We can first represent R as an orthonormal rotation matrix. It turns out we can actually solve this by considering all three correspondences at once:

$$R(\hat{\mathbf{a}}_1, \hat{\mathbf{a}}_2, \hat{\mathbf{a}}_3) = (\hat{\mathbf{b}}_1, \hat{\mathbf{b}}_2, \hat{\mathbf{b}}_3)$$

We can solve for R using matrix inversion, as below. Question to reflect on: Will the matrix inverse result be orthonormal?

$$A \triangleq (\hat{\mathbf{a}}_1, \hat{\mathbf{a}}_2, \hat{\mathbf{a}}_3)$$
$$B \triangleq (\hat{\mathbf{b}}_1, \hat{\mathbf{b}}_2, \hat{\mathbf{b}}_3)$$
$$RA = B \implies R = BA^{-1}$$

In practice, as we saw with other photogrammetry frameworks, we would like to have more than 3 correspondences in tandem with least squares approach. Can use estimates with first 3 correspondence to obtain initial guess, and then iteration to solve for refined estimate. May reduce probability of converging to local minima.

Note that errors are in image, not in 3D, and this means we need to optimize in the image plane, and not in 3D.

1.2 Bundle Adjustment

Bundle Adjustment (sometimes abbreviated as BA) is a related photogrammetry problem in which, in the most general case, consists of K landmarks (points in the image observed by cameras) and N cameras.

Goal of BA: Determine the 3D locations of landmark objects and cameras in the scene relative to some global coordinate system, as well as determine the orientation of cameras in a global frame of reference.

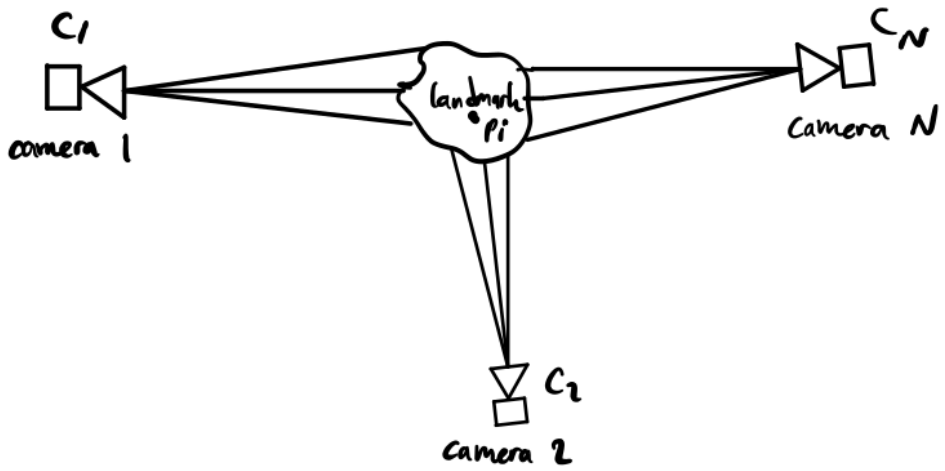

Figure 5: Bundle Adjustment (BA). In the general case, we can have any number of K landmark points ("interesting" points in the image) and N cameras that observe the landmarks.

What are our unknowns for Bundle Adjustment

- **Landmarks** ($\{p_i\}$): A set of points that are observed by multiple views, of which we want to find the coordinates of in a global coordinate system.
- **Cameras** ($\{c_i\}$): This is the set of camera locations in space with respect to some global coordinate system.
- **Camera Attitudes** ($\{R_i\}$): These are the orientations of the cameras in space with respect to some global coordinate system.
- **Intrinsic Orientations** ($\{x_{0_i}, y_{0_i}, f_i, K_i\}$: These refer to the camera intrinsics for each camera in our problem.

This approach is typically solved using **Levenberg-Marquadt (LM)** nonlinear optimization. Although there are many parameters to solve for, we can make an initial guess (as we did with our previous camera calibration approaches to avoid local minima) to ensure that we converge to **global**, rather than **local** minima.

How do we find "interesting points" in the image? One way to do this is to use descriptors (high-dimensional vectors corresponding to image gradients), as is done with SIFT (Scale-Invariant Feature Transforms). Some other, more recent approaches include:

- SURF (Speeded Up Robust Features)
- ORB (Oriented FAST and rotated BRIEF)
- BRIEF (Binary Robust Independent Elementary Features)
- VLAD (Vector of Locally Aggregated Descriptors)

1.3 Recognition in 3D: Extended Gaussian 3D

Recall our previous discussions with recognition in 2D - let's now aim to extend this to 3D!

Goal of 3D Recognition: Describe a 3D object.

Let's consider several representations for effective **alignment estimation** and **recognition**:

- **Polyhedra**: When this is our object representation, we often consider this to be a class of "block world problems". We can describe these polyhedra objects semantically with edges, faces, and linked data structures. Because this is not a realistic representation of the world, for systems in practice, we look for more complicated representations.
- **Graphics/curves**: These can be done with a mesh. Here, we consider any curved surface. Meshes can be thought of as polyhedral objects with many facets (polygon faces). This representation is well-suited for outputting pictures.

Figure 6: Example of a mesh. Note that this can be constructed as many facets, where each facet is a polygon.

What kind of tasks are we interested in for meshes, and more generally, the rest of our object representations?

- Find **alignment/pose** (position/orientation): For alignment, we can accomplish this task by assigning correspondences between vertices, but this is not a very effective approach because there is no meaning behind the vertices (i.e. they are not necessarily deemed "interesting" points), and the vertex assignments can change each time the shape is generated. Can do **approximate alignment** by reducing the distance to the nearest vertex iteratively and solving.
- Object **recognition**: We cannot simply compare numbers/types of facets to models in a library, because the generated mesh can change each time the mesh is generated. It turns out that this is a poor representation for recognition.

Since neither of these representations lend themselves well for these two tasks, let us consider alternative representations. First, let us consider what characteristics we look for in a representation.

1.3.1 What Kind of Representation Are We Looking For?

We are looking for an object representation that exhibits the following properties:

- **Translation Invariance**: Moving an object in a coordinate system does not change (at least not substantially) the object's representation.
- **Rotation Consistency**: Rotating the object changes the representation of the object in an understandable and efficient way that can be applied for alignment problems.

Representations to consider given these properties:

- **Generalized Cylinders**:
  - These can be thought of as extruding a circle along a line.
  - In the generalized case, we can extrude an arbitrary cross-section along a line, as well as allow the line to be a general curve and to allow the cross-section (generator) to vary in size.

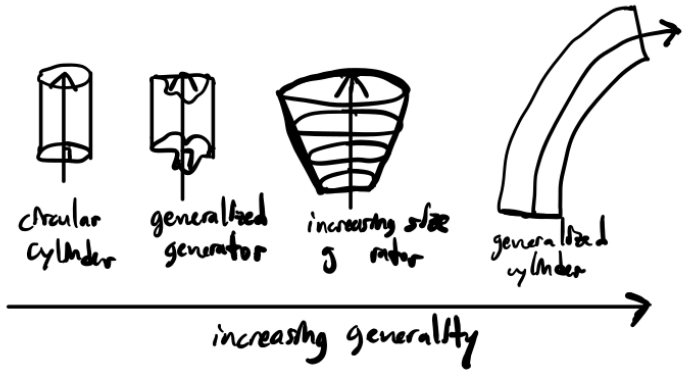

Figure 7: Representations of increasing generality for generalized cylinders.

Example: Representing a sphere as a generalized cylinder:

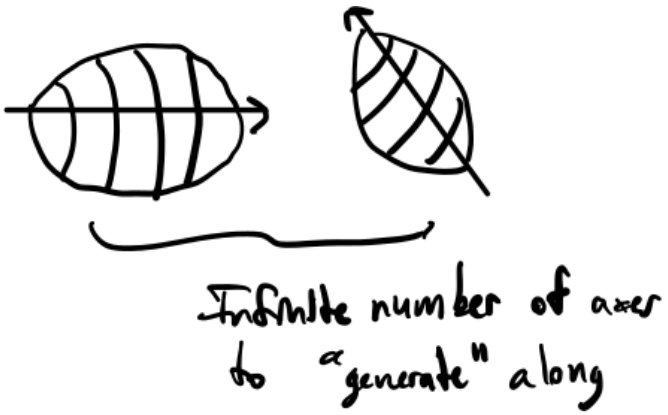

Figure 8: Representing a sphere as a generalized cylinder - unfortunately, there are an infinite number of axes we can use to "generate" the cylinder along.

  - This same problem shows up elsewhere, especially when we allow for inaccuracies in data.
  - This approach has not been overly successful in solving problems such as alignment, in practice.
  - This leads us to pursue an alternative representation.

- **Polyhedra Representation**:
  - Let's briefly revisit this approach to get a good "starting point" for our object representation that we can solve **alignment** and **recognition** problems.
  - One way, as we have seen before, was to take edges and faces, and to build a graph showing connectedness of polyhedra.
  - Instead, we take the unit normal vector of faces and multiply them by the area of the face/polygon.

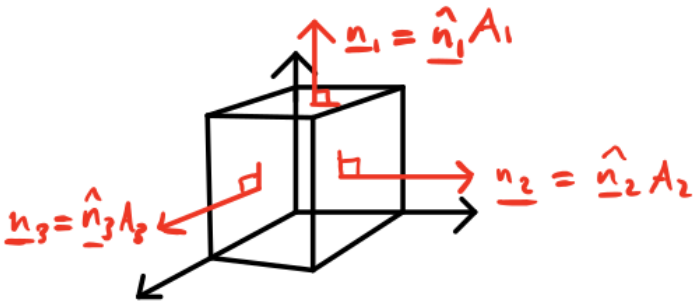

Figure 9: Polyhedra approach in which we represent faces by the unit normal vector multiplied by the area of the face of the polyhedra representation.

  - Despite throwing away information about the connectedness of faces, Minkowski's proof shows that this generates a unique representation for **convex polyhedra**.
  - Note that this proof is non-constructive, i.e. there was no algorithm given with the proof that actually solves the construction of this representation. There is a construction algorithm that solves this, but it is quite slow, and, as it turns out, are **not needed for recognition tasks**.
  - Further, note that the sum of vectors form a closed loop:

$$\sum_{i=1}^{N} \mathbf{n}_i = 0$$

    Therefore, any object that does not satisfy this is **non-convex polyhedra**.

Let's consider what this looks like for the following cylindrical/conic section. Consider the surface normals on the cylindrical component ($A_i \hat{\mathbf{n}}_i$) and the surface normals on the conic component ($B_i \hat{\mathbf{n}}_i$). Note that we still need to be careful with this representation, because each mesh generated from this shape can be different.

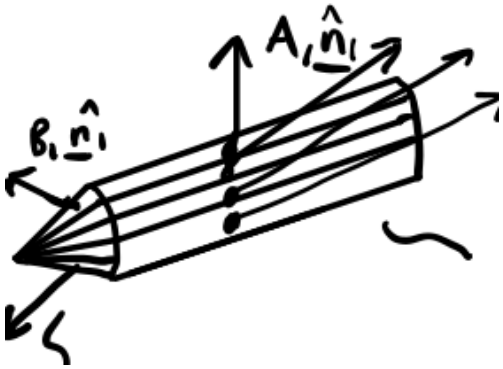

Figure 10: A shape represented by a mesh that we will convert to an extended Gaussian sphere, as we will see below. Note that the surface normals $A_i\hat{\mathbf{n}}_i$ and $B_i\hat{\mathbf{n}}_i$ each map to different great circles on the sphere, as can be seen in the figure below.

Idea: Combine facets that have similar surface normals and represent them on the same great circle when we convert to a sphere.

What does this look like when we map these scaled surface normals to a unit sphere?

Idea: Place masses in great circle corresponding to the plane spanned by the unit normals of the **cylindrical** and **conic** sections above. Note that the mass of the back plate area of our object occupies a single point on the mapped sphere, since every surface normal in this planar shape is the same.

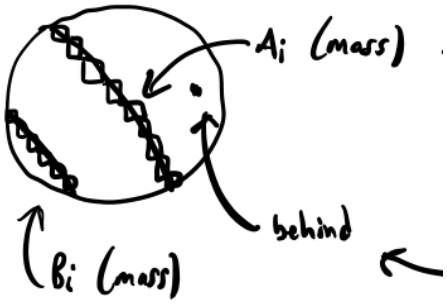

Figure 11: Extended Gaussian sphere mapping, which maps the surface normals of arbitrary (convex) surfaces to the sphere such that the surface vector corresponds to a scaled unit normal in both the original surface and the sphere.

Therefore, this sphere becoems our representation! This works much better for **alignment** and **recognition**. Note that we work in a sphere space, and not a high-dimensional planar/Euclidean space. Therefore:

- Comparisons (e.g. for our recognition task) can be made by comparing masses using distance/similarity functions.
- Alignment can be adjusted by rotating these spheres.

How well does this representation adhere to our desired properties above?

- **Translation Invariance**: Since surface normals and areas do not depend on location, translation invariance is preserved.
- **Rotation**: Since rotating the object corresponds to just rotating the unit sphere normals without changing the areas of the facets, rotation of the object simply corresponds to an equal rotation of the sphere.

Loosely, this representation corresponds to **density**. This density is dependent on the degree of curvature of the surface we are converting to a sphere representation:

- Low density $\leftrightarrow$ High curvature
- Higqh density $\leftrightarrow$ Low curvature

To better understand this representation in 3D, let us first understand it in 2D.

1.3.2 2D Extended Circular Image

The idea of the extended Gaussian Image: what do points on an object and points on a sphere have in common? They have the same surface normals.

Figure 12: Representation of our extended Gaussian image in 2D. The mapping between the surface and the sphere is determined by the points with the same surface normals.

Gauss: This extended Gaussian image representation is a way to map arbitrary convex objects to a sphere by mapping points with the same surface normals.

We can make this (**invertible**) mapping in a **point-point** fashion, and subsequently generalize this to mapping entire **convex shapes** to a sphere. There are issues with non-convex objects and invertibility because the forward mapping becomes a **surjective mapping**, since multiple ponts can have the same surface normal, and therefore are mapped to the same location on the circle.

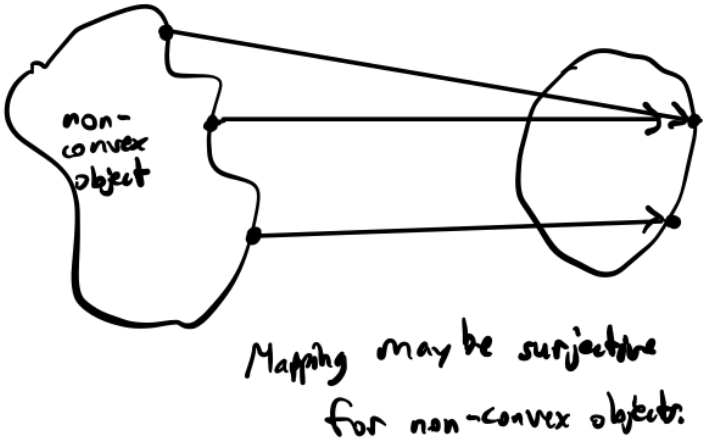

Figure 13: When surfaces are non-convex, we run into issues with inverting our mapping, because it may be surjective, i.e. some points on the surface may map to the same point on the sphere due to having the same surface normal.

Idea: Recall that our idea is to map from some convex 2D shape to a circle, as in the figure below.

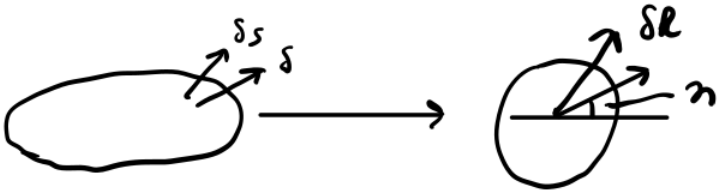

Figure 14: Our forward mapping from our surface to a circle.

Note that in this case, density depends on which region (and consequently the degree of curvature) of the object above. We are interested in this as a function of angle η. Analytically, our **curvature** and **density** quantities are given by:

- **Curvature**: $\kappa = \frac{d\eta}{dS} = \frac{1}{R_{\text{curvature}}}$ (The "turning rate")

- **Density**: $G = \frac{1}{\kappa} = \frac{dS}{d\eta}$ (inverse of curvature)

These are the only quantities we need for our representation mapping! We just map the density $G = \frac{dS}{d\eta}$ from the object to the unit sphere. This is invertible in 2D for convex objects, though this invertibility is not necessarily the case in 3D.

1.3.3 Analyzing Gaussian Curvature in 2D Plane

Let us analyze this mapping in the 2D plane, using the figure below as reference:

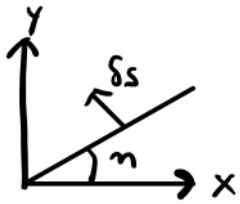

Figure 15: Analyzing Gaussian curvature in the 2D plane.

Mathematically, then, we can say:

$$\delta_x = -\sin(\eta)$$
$$\delta_y = \cos(\eta)$$

Then we can integrate to find x and y:

$$x = x_0 + \int -\sin(\eta) dS$$
$$x = x_0 + \int -\sin(\eta) \frac{dS}{d\eta} d\eta$$
$$x = x_0 + \int -\sin(\eta) G(\eta) d\eta$$

And similarly for y :

$$y = y_0 + \int \cos(\eta) G(\eta) d\eta$$

What happens if we integrate over the entire unit circle in 2D? Then we expect these integrals to evaluate to 0:

$$\int_0^{2\pi} -\sin(\eta) G(\eta) d\eta = 0$$
$$\int_0^{2\pi} \cos(\eta) G(\eta) d\eta = 0$$

These conditions require that the centroid of **density** $G(\eta)$ is at the origin, i.e. that the weighted sum of $G(\eta) = 0$.

1.3.4 Example: Circle of Radius R

In the case of a circle with constant radius, it turns out our curvature and density will be constant:

$$K = \frac{d\eta}{dS} = \frac{1}{R}$$
$$G(\eta) = \frac{dS}{d\eta} = R$$

Note that while this holds for spheres, we can actually generalize it beyond spheres too (making it of more practical value). This is because we can fit curves of arbitrary shape by fitting a circle locally and finding the radius of the best-fitting circle.

Note that because density is equal to R, this density increases with increasing radius. Because density is the same everywhere, we cannot recover orientation as there is not uniqueness of each point. Let's try using an ellipse in 2D.

1.3.5 Example: Ellipse in 2D

Recall that there are several ways to parameterize an ellipse:

- **Implicitly**: $\left(\frac{x}{a}\right)^2 + \left(\frac{y}{b}\right)^2 = 1$

- **"Squashed circle"**: $x = a\cos\theta,\ y = b\sin\theta$ (note that θ is the variable we parameterize these parametric equations over. Because this takes a parametric form, this form is better for generating ellipses than the first implicit form because we can simply step through our parameter θ.

Next, let's compute the normal vector to the curve to derive a surface normal. We can start by computing the tangent vector. First, note that the vector $\mathbf{r}$ describing this parametric trajectory is given by:

$$\mathbf{r} = (a\cos\theta, b\sin\theta)^T$$

Then the tangent vector is simply the derivative of this curve with respect to our parameter θ:

$$\mathbf{t} = \frac{d\mathbf{r}}{d\theta} = (-a\sin\theta, b\cos\theta)^T$$

Finally, in 2D we can compute the normal vector to the surface by switching x and y of the tangent vector and inverting the sign of y (which is now in x's place):

$$\mathbf{n} = (b\cos\theta, a\sin\theta)T$$

We want this vector to correspond to our normal in the sphere:

$$\mathbf{n} = (b\cos\theta, a\sin\theta)T \leftrightarrow \hat{\mathbf{n}} = (\cos\eta, \sin\eta)^T$$

Then we have:

$$\begin{aligned}
&b\cos\theta = n\cos\eta \\
&a\sin\theta = n\sin\eta \\
&n^2 = b^2\cos^2\theta + a^2\sin^2\theta \\
&\mathbf{s} = (a\cos\eta, b\sin\eta)^T \\
&K = \left(\frac{\mathbf{s}\cdot\mathbf{s}}{ab}\right)^{\frac{3}{2}} \\
&n^2((a\cos\eta)^2 + (b\sin\eta)^2) = (ab)^2
\end{aligned}$$

These derivations allow us to find the extrema of curvature: ellipses will have 4 of them, spaced $\frac{\pi}{2}$ apart from one another (alternating minimums and maximums): $\eta = 0, \eta = \frac{\pi}{2}, \eta = \pi, \eta = \frac{3\pi}{2}$.

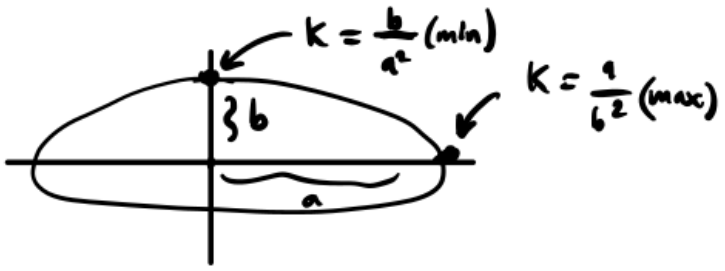

Figure 16: Curvature extrema of an ellipse: An ellipse has 4 extrema, spaced $\frac{\pi}{2}$ apart from one another. Extrema depend on the major and minor axes of the ellipse.

If a is the major axis and b is the minor axis, then extrema of curvature are given by:

- **Maximum**: $\kappa = \frac{a}{b^2}$
- **Minimum**: $\kappa = \frac{b}{a^2}$

Without loss of generality, we can flip a and b if the major and minor axes switch.

For alignment, we can rotate these ellipses until their orientations match. If the alignment is not constructed well, then the object is not actually an ellipse.

Geodetic Identity: Related, this identity relates geocentric angles to angles used for computing latitudes:

$$b\cos\theta = n\cos\eta$$
$$a\sin\theta = n\sin\eta$$
$$a(\tan\theta) = b(\tan\eta)$$

What are some other applications of this in 2D? We can do (i) **circular filtering** and **circular convolution**!

1.3.6 3D Extended Gaussian Images

Fortunately, many of the results in 2D Gaussian images generalize well to 3D. As with previous cases, we can begin with the Gauss mapping (starting with points, but generalizing to different convex shapes in the surface). From here, we can compute **curvature** and **density**:

- **Curvature**: $\kappa = \frac{\delta S}{\delta O} = \lim_{\delta \to 0} \frac{\delta S}{\delta O} = \frac{dS}{dO}$
- **Density**: $G(\eta) = \frac{\delta O}{\delta S} = \lim_{\delta \to 0} \frac{\delta O}{\delta S} = \frac{dO}{dS}$

Although curvature in 3D is more complicated, we can just use our notion of Gaussian curvature, which is a single scalar.

As long as our object is convex, going around the shape in one direction in the object corresponds to going around the sphere in the same direction (e.g. counterclockwise).

Example of non-convex object: Saddle Point: As we mentioned before, we can run into mapping issues when we have non-convex objects that we wish to map, such as the figure below:

Figure 17: An example of a non-convex object: a saddle point. Notice that some of the surface normals are the same, and therefore would be mapped to the same point on the sphere.

Note further that traveling/stepping an object in the reverse order for a non-convex object inverts the curvature κ, and therefore allows for a negative curvature $\kappa < 0$.

Example of 3D Gaussian Curvature: Sphere of Radius R: This results in the following (constant) curvature and density, as we saw in the 2D case:

- **Curvature**: $\kappa = \frac{1}{R^2}$
- **Density**: $G(\eta) = R^2$

Extensions: Integral Curvature: Next time, we will discuss how Gaussian curvature allows for us to find the integral of Gaussian curvature for recognition tasks.