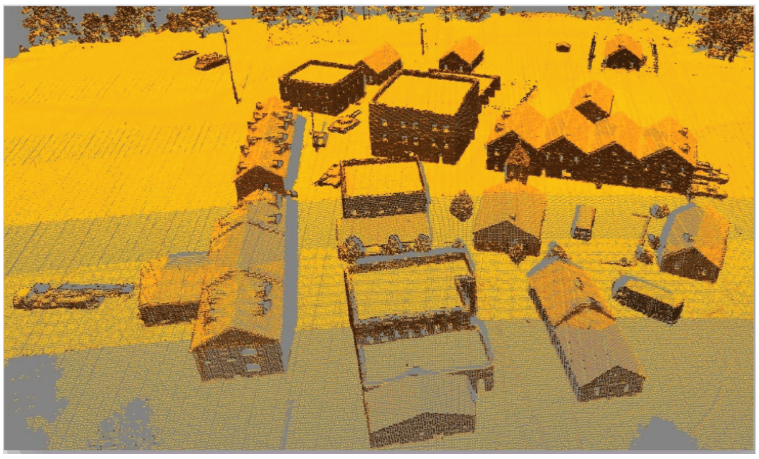
MARV216 - Programming project   Airborne LiDAR simulator



The idea is to build an airborne LiDAR (laser scanner) simulator that can be used to scan a 3D scene consisting of simplified man-made and natural targets. It would produce XYZ point clouds that are affected by the scanning parameters, positioning inaccuracy, and by the targets themselves. These can be visualized / analyzed further using other software.

Contents

Objectives

The project introduces the students with a real case. Simulators are handy tools for exploring complex systems. Climate forecasts are based on simulations with models, for example. Our project teaches how to break a complex system into manageable pieces. Functions and object-oriented programming will be used for abstraction.

LiDAR is an active remote sensing device, which 'actively images' the targets by transmitting and receiving optical wavelength energy pulses. It bears close resemblance to radar, and actually radar equation[1] applies to LiDAR as well. Pulsed LiDARs have gained foothold in topography, forestry and in the mapping of forested ecosystems. It is an important technology / tool.

Successful implementation of a LiDAR simulator calls for detailed information about the operating and functioning of the sensor, as well as the 3D imaging processes (What goes on at canopies or at target surfaces, when the pulse hits them?). This document clarifies these aspects thus freeing our minds to programming and implementation.

On instructor's and Students' roles

The project is reasonably demanding, not because of the math that is involved, but because there are so many factors - i.e. the simulated process is complex. The programming cannot commence until the programmers at some level understand the goals and the technical details (What is the simulator expected to do?). We will therefore first, also with the help of this document, try to see the 'big blurred picture' and later high-pass-filter out the many details. However, it is crucial not to have the simulator to do anything that is not meaningful to the end result.

[1] http://en.wikipedia.org/wiki/Radar#Radar_equation

IPO

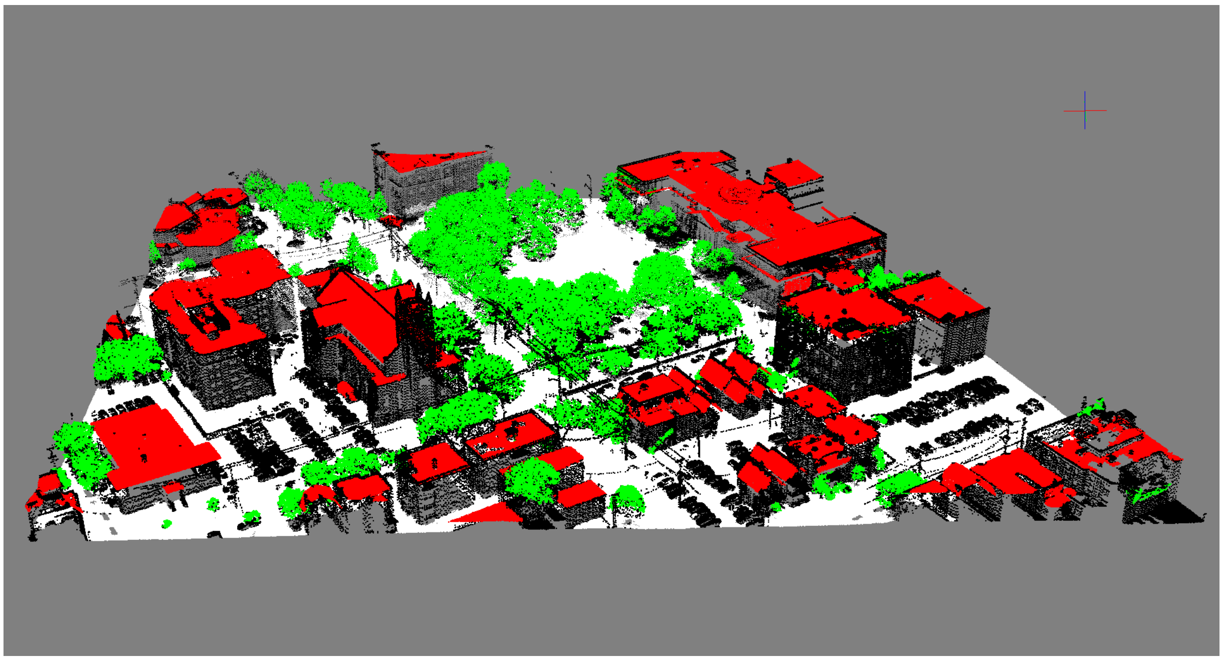
This program will follow the input-process-output -formula.

Input is multifaceted. We are using the LiDAR to map a scene consisting of man-made and natural geometric-optical objects: roof plates, walls, streets, grass, trees, etc. The scene is one form of input. The LiDAR campaign or flight over the scene is done at some flying speed and height. Similarly, the LiDAR sensor can be operated in several modes, and we need to input these selections.

The **processing** is basically simple. We let the LiDAR to function on top of the scene and collect data. Details include many processes: the LiDAR moves along a trajectory, the aircraft is made to sway, the mirror is oscillating while we fly, and every 10 milliseconds or so, we transmit a pulse towards the targets in the scene and figure out which ones they hit. The hit position in XYZ is important as is the reflectivity and orientation of the encountered target surface. These determine the amount of photons that we see reflecting back to the receiver in the LiDAR sensor.

The processing results in two kinds of observations: 1) We will have echoes (~returns) that are xyz-points and for each echo the maximal intensity of the reflection is stored. 2) We will also store the received photons on a time-axis as they arrive. This results in what is called LiDAR waveform.

The **output** is just a simple process of outputting the echo-data to be displayed in some other software. Ultimately the output is visual.



What is a LiDAR?

LiDAR is an active sensor that is described thru a few central parameters. It transmits very short (a few meters long) laser light pulses to a known direction **dir** = (i,j,k) from position **P** = (X,Y,Z), and measures the distance to the reflecting target(s) by measuring the time (t_{echo}) that elapsed until the reflected pulse returned (light makes a 2-way direct path trip):

$$\mathbf{[X,Y,Z]_{target} = P} + t/2 \times c \times \mathbf{(i,j,k)},\ \text{where} \qquad (1)$$

t is the travel time, and c is the speed of light. If, in (1), **P** = (0,0,0) and the direction (**dir**) is up (0,0,1), and the time is 0.121 milliseconds, the coordinates in meters are (0,0,18150.00). This example shows that it is important to know precisely the position (**P**) of the LiDAR at the time of transmission as well as the sensor attitude, that define the direction (i,j,k). (Fig. 1)

Fig. 1. The “attitude” means three things in 3D. There are rotations about X and Y axes (knobs in the pictured game), and the rotation about Z (rotate the whole game on the table).

If the platform is an airplane, the typical ground speed is 60-100 m/s. So when we are scanning from an altitude of 18 km, the platform moves 1.2 cm when the pulse is in the air. From a height of 2 km, the pulse is airborne 13 us, and the platform movement is just 1.3 mm. (Fig. 2).

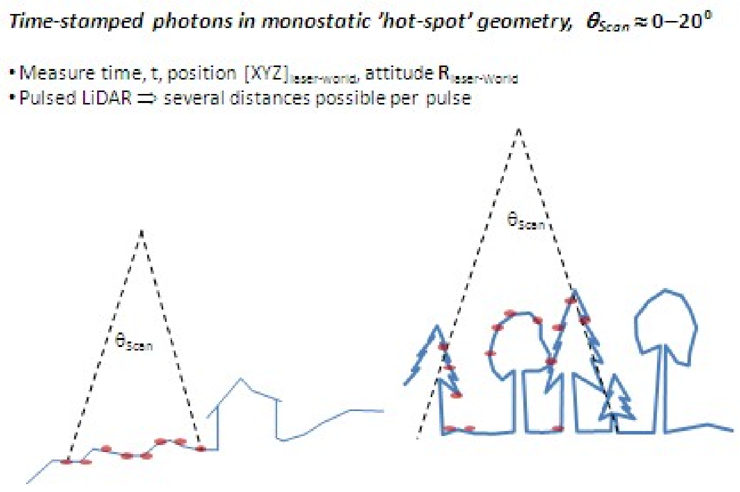

Time-stamped photons in monostatic 'hot-spot' geometry, $\theta_{Scan} \approx 0-20^0$
• Measure time, t, position $[XYZ]_{laser\text{-}world}$, attitude $R_{laser\text{-}World}$
• Pulsed LiDAR ⇒ several distances possible per pulse
θ_{Scan}
θ_{Scan}

Fig. 2. LiDAR pulses are transmitted at high frequency, e.g. 200,000 times per second (Hz). The pulse direction is purposely deviated some ± 20° to cover a wide swath (strip), while flying over a scene. The reflections come from targets that are illuminated and reflective enough. One pulse can yield several echoes from varying depth, if the first encountered object is semi-transparent (vegetation), or the pulse only partially hits an opaque target.

The energy of the laser light is not spatially or temporally constant. The 1-5-m-long laser light pulse transmitted has its peak radiance [W/m2] usually in the centre of the cross-section of the (conical) pulse. Time wise, the radiance rises to maximum and drops down to zero in a few nanoseconds. The energy over time distribution resembles a Gaussian. (Fig. 3).

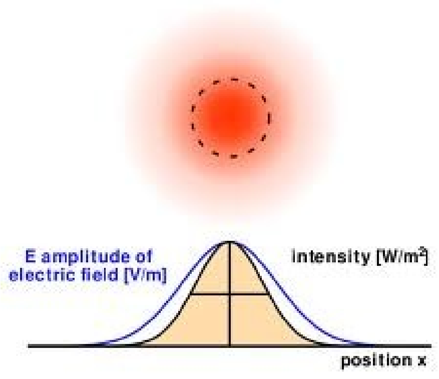


Fig. 3. This figure is not for LiDAR, but the upper sub-fig shows the energy distribution in a pulse, 'as seen by an eye'. Power per unit area is highest in the centre. The lower sub- figure: put time on the x-axis (some nanoseconds only in total), and the y-axis is the power (W).

The energy is concentrated into a cone that is defined by divergence (opening angle). Typical values are 0.1...2 milliradians. A pulse with 0.3 mrad divergence has a 30 cm footprint from 1 km.

When the energy cone hits an object, it is illuminated ('footprint'). The illumination is strongest in the center of the cone and diminishes towards the edges (Fig. 3). The illumination gives rise to a reflection. Since the receiver in the LiDAR is right next to the transmitter, they see the target in the same geometry. Thus, only the photons reflecting (back) in the direction of the LiDAR, contribute to received photon count (observations). There are several factors that influence the strength of the backscatter:

- The illumination and target size
  (a) Is the target larger or smaller than the illuminated footprint,
  (b) and in which part of the footprint does the target reside,
  (c) has the pulse energy/power reduced due to atmosphere, previous targets, long distance ?

- Target reflectivity [2]
  What is the reflectivity of the illuminated surfaces at the wavelength of the LiDAR?

- Geometry [3]
  If the illuminated surface is inclined, how is the reflection affected?
  Consider for example sloped roof, or leaves with inclination angles.

[2] http://en.wikipedia.org/wiki/Reflectivity
[3] http://en.wikipedia.org/wiki/Lambert's_cosine_law

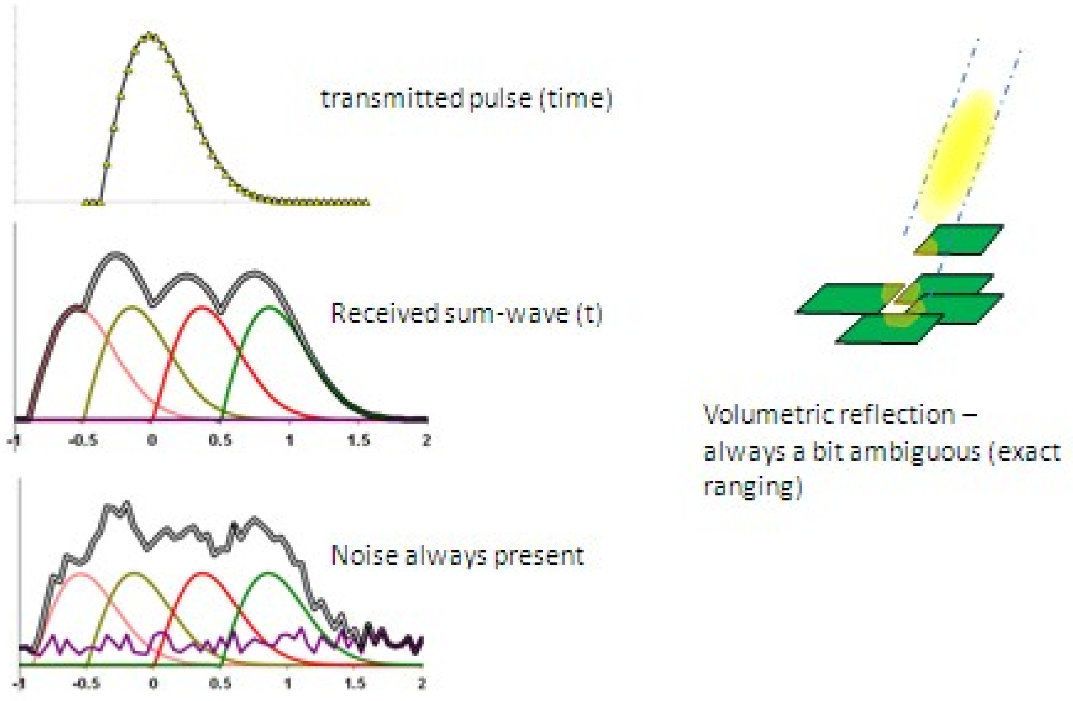


Fig. 4. Up-left: The transmitted pulse usually rises fast to a maximum and has a tail-end. Middle-left: If the pulse meets vegetation, the reflection is always volumetric (as oppose to well-defined surfaces). The thick line is the received wave, and it is a sum of several (leaf-level?) reflections. Low-left: The atmosphere scatters back photons and the electronics in the LiDAR receiver create noise (purple line). Detecting anything but the first echo becomes difficult as the blur makes other maxima to smear out.

Now consider that we have the LiDAR installed in an airplane. Unless we shoot the LiDAR towards a vacuum space or towards a black object, we usually get some photons back. Atmosphere reflects them, more from lower, thicker layers of the atmosphere, the medium, which is thicker with aerosols, particles, and water vapour[4]. This scattering is weak, but forms a background noise level for anyone interested in the terrestrial targets (N.B. atmospheric LiDARs existed before any topographic sensors). The electronics in the receiver are also producing noise, “fake photons”. This adds to the noise level, above which the signal must reach before we believe that it is an observation from the ground (Fig. 4).

[4] http://en.wikipedia.org/wiki/Optical_depth

Shortcuts and assumptions

A simulator has to contain the essential parts, but not everything. Shortcuts in the simulator have to be made, because we cannot simulate all the LiDAR functionality at very detailed level.

Pulse

A single ray cannot model the energy cone (recall spatial variation, Fig. 3), but a pencil (bundle) of rays is a good approximation. If the pulse path is **p** + d * **dir**, we can let **dir** to vary around the correct direction.

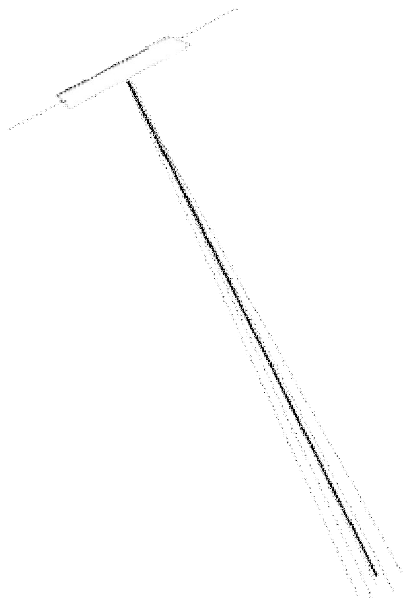


Fig. 5. Simple LiDAR: rotating mirror and a transmitted (here long) pulse. The thick line is **p** + d * **dir,** the 'central ray' of the pulse. The thinner rays depict sub-rays, which fill the beam divergence (opening angle), such that there are more sub-rays near the centre of the footprint.

But, we can fill the cone (opening angle of the pulse) with e.g. 100 rays with more rays the center of the cone (cross-section), to model the higher power per unit (W/m2) area there. Using 100 sub-rays, we can produce partial hits, where for example a corner of a roof is hit first by a subset of rays, while some of the rays continue their path down to another target. If a near-vertical LiDAR pulse hits a vertical wall, the sub- rays can be used to model a widened reflection (the LiDAR-target range varies in the footprint).

Receiver

Each sub-ray in a pulse hits something in the scene. We have to test the pulse-target intersections for all targets. The intersection that has shortest range, i.e. LiDAR-target distance, is the one that the sensor 'sees' in reality[5]. At the sub-ray level, target reflectance (refl = 0−1) and surface orientation (angle(**dir**, **surface normal**)) determine the signal, roughly speaking, the photon count.

[5] http://en.wikipedia.org/wiki/Z-buffering

Thus each sub-ray in a pulse yields (distance, signal strength) -observations. If there are 100 sub-rays in a pulse, the received signals vary in strength, and they span across a range of distances (Fig. 6).

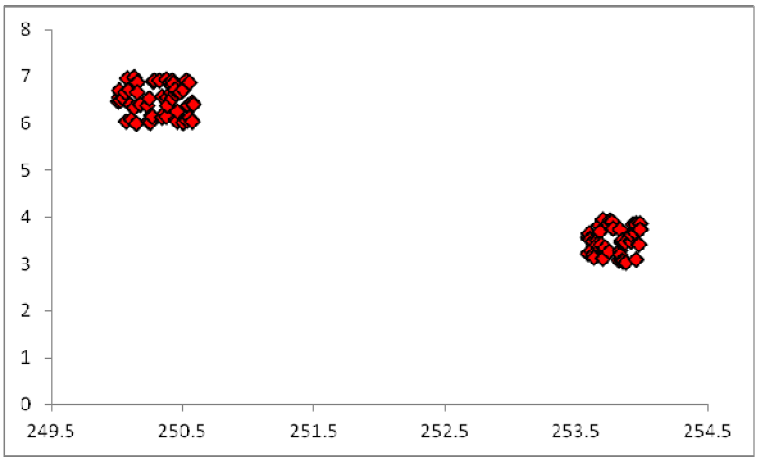


Fig. 6. An example where the pulse (100 sub-rays) has hit two surfaces at apprx. distances of 250.25 and 253.75 m, respectively. Each ray has produced a slightly different photon count (because here was assumed that laser speckle[6] affects observations). The distances vary, owing to oblique scanning - different parts of the illuminated target surface are at different distances. The distances do not vary if the surfaces are orthogonal to the pulse.

Receiver noise we can skip or just add some white noise[7] in the received signal. The data in Figure 6 assumes now that each pulse in a sub-ray is extremely short in duration (we see dots). In reality, they all have some length (Fig. 4 upper-left, Fig. 7), and the observations span across time (distance).

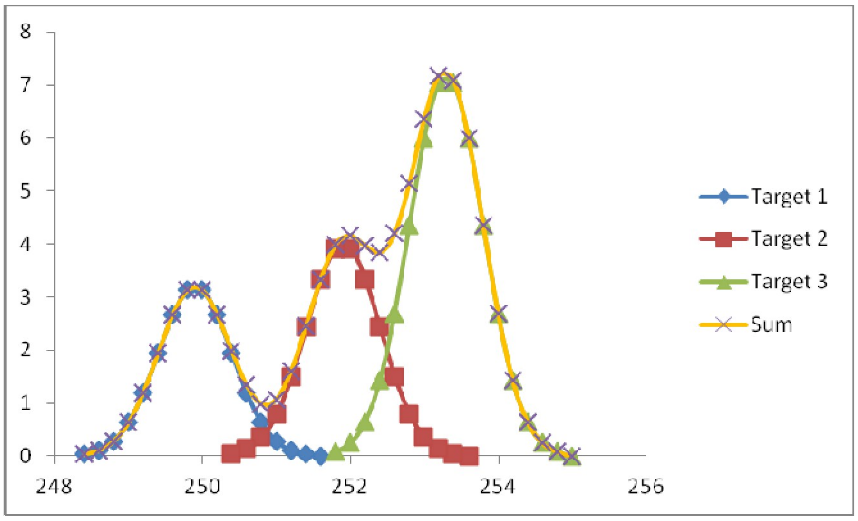


Fig. 7. In this example there are three "clumps" similar to those in Fig. 6. The convolution with the transmitted pulse shows how the photons actually arrive in the receiver from three sub-targets, being hit by the pulse. Sum-wave is what is stored, and it is the sum of the three pulse-shaped reflections from the targets.

[6] http://en.wikipedia.org/wiki/Speckle_pattern
[7] http://en.wikipedia.org/wiki/White_noise

Medium

We will assume that rays travel a straight path, although in reality that may not be exactly true[8].

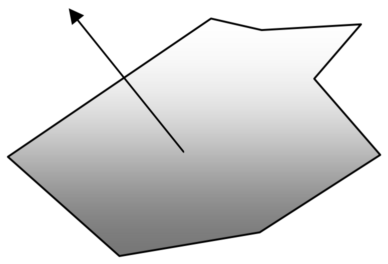
Atmospheric noise we can skip or, again, just add some white noise in the received signal. In near-infrared, the scattering (~path radiance) is roughly 1-2% of the reflection from ground; if the weather is dry (clear air between LiDAR and target). Clouds, smoke or fog are unwanted.

The power per unit area (w/m2) reduces with the second power of the scanning distance (the radar equation). We deal with this by using some nominal distance as the basis ($R_{nominal}$). The signals from distances R are affected by multiplication with term 1 / (R/ $R_{nominal}$)**2.

So if $R_{nominal}$ is 200 m, and we observe a photon count from 250 m, we can normalize this by 1 / ((250/200)**2).

Target reflectance

The targets must be specified some nominal reflectance, *refl* = 0−1. refl = 0 for a black target and refl = 1 for a perfect reflector. We assume that we have a near-infrared LiDAR (λ = 1064 nanometers). This keeps the atmospheric effects to a minimum, and the reflectance of vegetation is 0.3−0.5. We have to use guesswork for other types of surfaces (if we don't find literature values).



The inclination of the target surface is given by the surface normal vector (**n**). We assume that all our surfaces are planar. A 3D plane can be represented in the form: Ax + By + Cz + d = 0.

A, B, and C make the surface normal vector’s x, y, and z components, respectively. If the pulse hits the surface at 90 degree angle, the normal vector and the pulse direction vector are collinear (their angle == 0). This is the geometry, where we assume maximal reflectance. We can test different models, where the reflectance is proportional to cos(*angle*)***a*.

Trajectory

LiDAR will be flown over the scene between time points t_1 and t_2, along a 3D trajectory **P**(t). Thus time (t) is crucial for the simulator. Here trajectory not only includes the position **P(t)** = [x,y,z] (t), but also the angles in 3D [omega, phi, kappa](t) (Fig. 1). These angles give the orientation (~rotation, ~attitude) of the LiDAR sensor in the World's xyz-system (which is where we map the targets with LiDAR). An aircraft

[8] http://en.wikipedia.org/wiki/Atmospheric_refraction

carrying the sensor normally tries to follow a straight path (which of course is hard given the shape of Earth). Despite the fact that world is not flat, we will use a Cartesian 3D xy-system. In a small area, Earth curvature is negligible.

We will simulate the trajectory of the aircraft to include small deviations in 3D attitude (the 3 angles).

The 3D path of a LiDAR pulse is **p** + d***dir**. Here, **p** is the position, and **dir** is the direction of the pulse. These data (**p**(t), **dir**(t)) are a part of trajectory data . The **dir** vector is a function of the plane's attitude, i.e rotation (omega, phi, kappa) and the direction of the mirror (see below).

It is crucial to know **P** and **dir**, for each pulse transmission. Otherwise the resulting LiDAR (geometric) data is garbage. In practice, instrumentation called GNSS/IMU[9] is used to get the geometry over time. This instrumentation has limited accuracy, however. We can simulate it with our simulator.

Mirror

The LiDAR itself is mounted (bolted on the frame, or sits on a gyro-stabilized mount) on the aircraft. LiDAR is a device with its own 3D coordinate system. We will assume that it is perfectly aligned with the XYZ-system of the aircraft (Fig. 8).

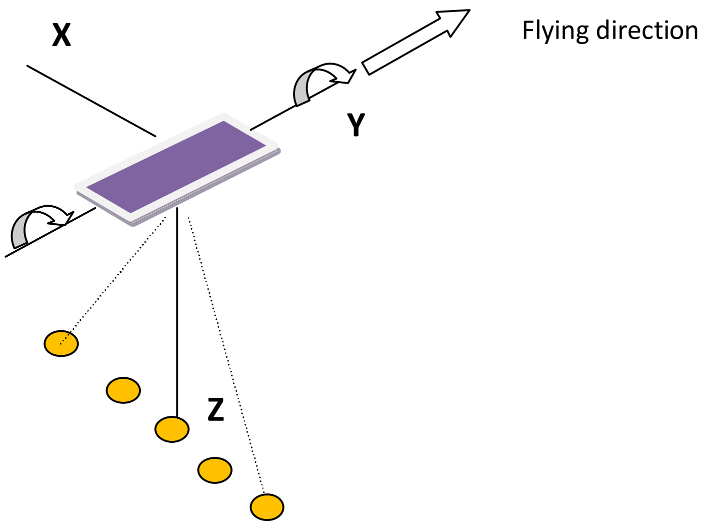


Fig. 8. LiDAR coordinate system. The oscillating mirror produces footprints that hit the x-axis on the ground. The Z of the echoes is positive.

Now, a LIDAR has a laser pump, which has a screen (fast electronic shutter) that opens up and lets a pulse thru for a few nanoseconds. Among first things that this pulse sees is an oscillating mirror. It reflects the pulse downwards, towards ground. The mirror is on an axis that is parallel to the flying

[9] http://www.ifp.uni-stuttgart.de/publications/phowo01/Reid.pdf

direction, so that the pulse, as the mirror moves, oscillates between 'left and right'. The maximal angular movement of the mirror is called scan angle. Typical values are ± 15° (Fig. 8).

The axis of the mirror is aligned with the y-axis of the LiDAR. We can say that the z-axis of the LiDAR points down. The pulse moves along the x-axis of the LiDAR, from negative to positive values. So, in the xyz system of the LiDAR, reflections from the ground all have coordinates (x,0,z).

This is not the right **dir** vector in the world coordinates, because LiDAR is rotated in 3D as the plane moves. So we will need rotate the (x,0,z)-vectors furthers in 3D, to match the attitude of the world coordinates, i.e. to get the right **dir** - the direction of the pulse (**p**+t***dir**)

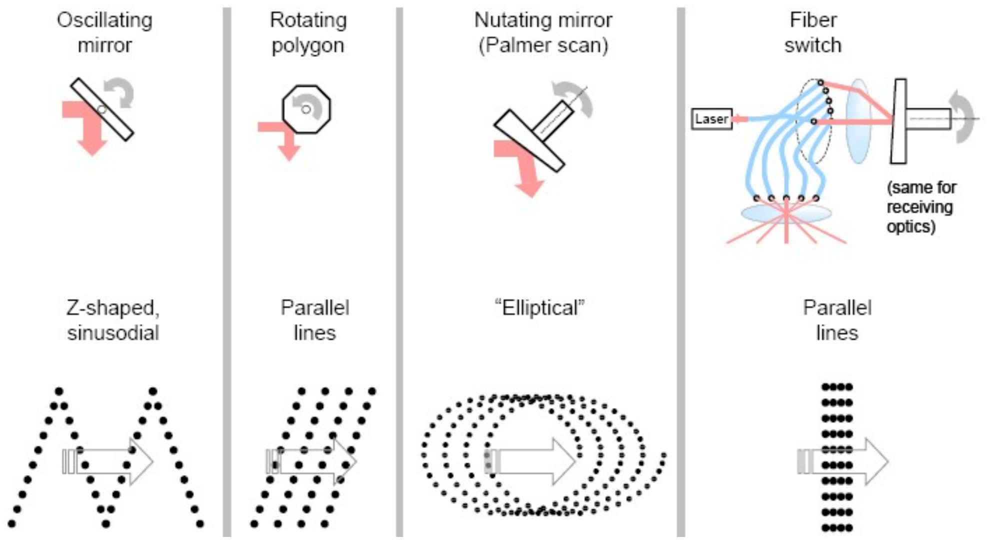


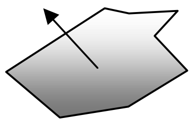
Fig. 9. We assume that we have an oscillating mirror (left case).

Time

LiDAR transmits a pulse at some PRF, pulse repetition frequency, e.g. at 150 kHz. This is the fastest repeating event in our simulator. For each pulse, we need the trajectory (**p** + d***dir**).

1/PRF sets the shortest time lapse in the simulator. We need to know what are **p**, **dir** and the mirror position at these time points. The mirror oscillates at some scan frequency, usually at 30-100Hz. One cycle is a middle-left-right-middle movement (or left-right-left).

Pulse – Target interaction --- of planar polygons and trees



The cone of energy loses irradiance (W^2/m^2) as the pulse spreads (diverges) with distance (~range, *R*). The effect is R^2 as stated before.

The receiver in LiDAR is just a very fast photon counter. It can calculate the incoming photons at the wavelength of interest (usually near-infrared light) at fast pace, e.g. at every 500 pikoseconds. Now, there are background photons and noise in the electronics, so there has to be certain minimum power received, before it can be considered 'true' or worthy. But that we can decide later, in the post-processing of the observations.

The pulse meets the surface at some angle of incidence. For a surface, whose normal points to LiDAR, the angle is zero. As stated earlier, we will assume that each surface has its nominal reflectance (*Refl*, 0−1), and a specific way of scattering light as a function of the incidence angle. We will assume that scattering has its peak when the surface normal points towards the sensor.

$$\text{Power\_received(sub-ray)} = (\text{Power\_ transmitted}) / (R/Rn)^2 \times (\text{Refl}) \times f(\text{Incidence\_angle}) \qquad (2)$$

Eq. 2 says that what we see returning (backscatter) is a function of what we transmit, which is kept constant, the distance (R**2), target's reflectance and the incidence angle. When that power arrives at the receiver, depends on the distance R.

Now, in our simulator we assume that man-made structures are all planar surfaces (short grass, gravel, asphalt, walls of buildings, roof materials etc.), while taller vegetation, which is trees for us, usually comprise ambiguous small targets (Fig. 4) that are many and occur in depth across a pulse's path, and smaller than the area illuminated by the LiDAR. Distribution of the direction of leaf normals varies - e.g. aspen leaves have their normal pointing to the horizon, while goat willow has them towards the LiDAR, flying above.

Vegetation does not form a surface. So Eq. 2 does not apply to vegetation, or we omit the last term, incidence angle. There is foliage (etc.) across a depth, and if their density (~radar cross-section[10]) is large enough, the reflection suffices to trigger a measurable signal in the LiDAR. We will describe vegetation in probabilistic terms – some 3D envelope will hold 'foliage' that has certain density, and we will let the pulse penetrate this foliage and the reflection is evaluated in discrete manner (there will be a kind of a lottery).

How?

Targets – how to describe and find the pulse-target intersection?

We will describe man-made targets by planar 3D polygon surfaces: ordered point-list {[xyz_1],[xyz_2],

[10] http://en.wikipedia.org/wiki/Radar_cross-section

…,[xyz$_n$]}. A house is made of roof plate(s), wall plates, and some ground plate that intersects the walls. Each surface is defined by the ‘corner points’, vertices that delineate the 3D plane.

We will thus need a way of saying if and where the 3D laser pulse (ray, line) intersects a plane polygon and at what incidence angle.

In terms of programming, a surface is clearly an **object**. The attributes are related to the xyz delineation (list of points making the surface patch), the surface material and its reflectance.

To say, which target was hit first by a sub-ray in a pulse, we need to keep track of the distances – closest hit target is seen by LiDAR and recorded as the observation (as in Figs. 6 & 7).

Trees will have a different geometry. We will define a crown envelope for the living part of the crown, and a tree will also have a trunk. Trees have several attributes: height, position, reflectance etc. The crown envelope–pulse intersection needs to be solved by some iterative means, if we choose non-trivial shape for the crown. If the crown were a cone or a sphere, it would be simpler to solve ray-object intersection.

We will use a concept called bounding box[11] to speed up the processes of finding the 3D planar polygons being hit by a pulse. Namely, it is fast to compute the intersection of a 3D plane and line. It is a 3D point. Then it is faster to check if that point is inside a 3D box that encloses the 3D polygon than it is to check if the point sits inside the polygon. If the point is not inside the polygons bounding box, it cannot be inside the polygon. We save some time.

Computing the trajectory and current attitude of the LiDAR (sensor).

Time is described (and incremented) in “ticks”, an integer that is incremented each time the LiDAR transmits a pulse. The pulse repetition frequency (PRF) tells this. We set it. If the PRF is 200,000 Hz, a tick is 5 microseconds. Thus we need to simulate the trajectory/attitude at this pace.

The trajectory starts at time t == 0. It starts from point $\mathbf{P}_{\mathbf{start}}$. We let the trajectory to be a straight **horizontal** line in 3D, stretching North. Horizontal means that the direction vector **dir_trj** (i,jk) will have k == 0 (we don't change the Z). Flying North is having j == 1.

$$\mathbf{P}(t) = \mathbf{P}_{start} + (0,1,0) \times t.$$

For example, if the flying speed is 65 m/s (0.000065 m/μs), we are 65 cm away from the start after 0.01 secs of flying. For simplicity, we always fly towards North, so i == 0, and j == 1. If we need to fly in other directions, we will just **rotate the scene around in XY** to have the same effect.

Because we fly straight to North (parallel to the positive Y-axis), the plane and the LiDAR, perfectly aligned with one another, are also aligned with world X, Y and Z. How do you turn an airplane? The roll, pitch and yaw are changed (Fig. 17). Now, we will not build a flight simulator, but for simplicity, we

[11] http://en.wikipedia.org/wiki/Minimum_bounding_box

assume that the trajectory is a straight line, although the angles change a little, as they do in reality, when the turbulence shakes the plane. So, in simulating the planes position and attitude, we will let the rotations (we call them omega, phi and kappa) vary slightly. These rotations make the plane and the LiDAR attached to it, to sway. We will use simple trigonometric functions with randomized coefficients to alter the omega, phi and kappa from their nominal values (0,0,0) as we fly over (see Fig. 10)

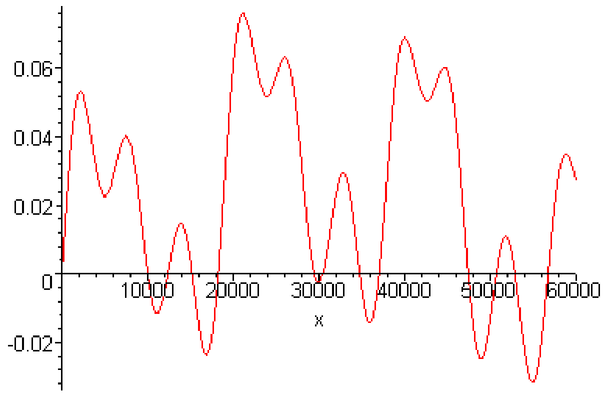


Fig. 10. An example for 60000 time ticks (transmitted pulses), where one of the three rotations is allowed to vary around 0, which is the default for the three rotations.

We simulate the "true" position and attitude of the LiDAR for all time points. Then it is possible to add GPS/IMU errors on top of these (not implemented in the first version).

How to say if the pulse hits a certain 3D polygon plane (a roof plate)?

If $\mathbf{p_1}$ is any point that lies on the plane, and **n** is the normal of the plane, we can check where the LiDAR ray and plane intersect:

$$t = ((\mathbf{p_1}-\mathbf{p}) \bullet \mathbf{n})/(\mathbf{dir} \bullet \mathbf{n}). \qquad (3)$$

p is the position of the LiDAR, and **dir** is the direction of the pulse. (•) is the dot product. The solution of (3) is the distance t, which is applied to the pulse equation (**p** + t × **dir**) to get the intersection point **px**. (Fig. 11).

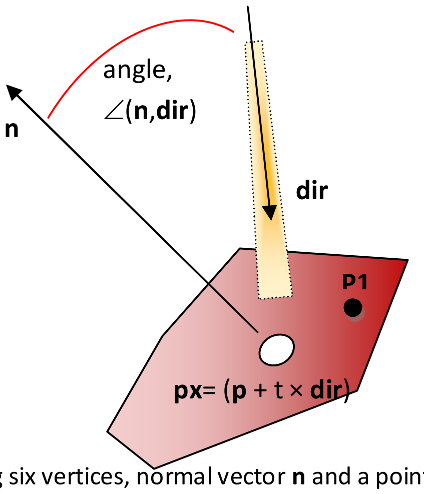


Fig. 11. 3D polygon plane consisting six vertices, normal vector **n** and a point **p1**. The intersection of the pulse with the plane is **px**, point in the centre of the "white footprint". Only a few pulses actually hit the polygon. This calls for inclusion testing.

The solution for t gives some [x,y,z] point **px**. Now, to check if **px** actually hit´s the planar surface polygon requires further testing. Recall that the planar polygon is 3D, consisting of xyz points. If we would have a 2D case, witha 2D **px** and 2D polygons, a general solution to **point_in_polygon()** is to check if a line drawn from the point intersects the bounding polygon an odd or an even number of times, the first indicates being inside. This works in 2D (Fig. 12)[12]. (Computational geometry[13])

If the planar surface describes vertical wall of a building, the xy-coordinates will show no variation, because the wall's projection in the xy-plane is a straight line, and you can't be inside a line! Testing inclusion with xy-coordinates will fail. We will be alternatively using yz-, xz-, or xy-planes (projections) to use the 2D point_in_polygon() to determine 3D point_in_polygon(). For vertical or near-vertical surfaces, we choose yz-, and xz-projections. You simply omit the x, y or z coordinate to project to these planes.

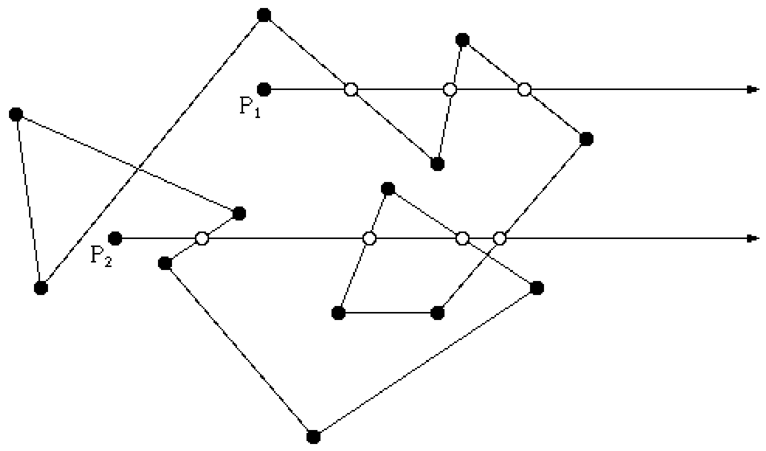


Fig. 12. Here two **px** points are denoted **p1** and **p2**. The black dots are the vertices of the polygon. Inclusion testing counts the intersections of arbitrary half-lines drawn from px. If the count is even, the point is out. Odd count means that px is in.

If the point [**p** + t × **dir**] is inside a planar polygon, compute the angle between (**n**, **dir**), store the distance (t), the reflectance of the surface and the angle (Fig. 11).

One pulse can intersect several overlapping (occluded from the viewing direction of the LiDAR) planar polygons (e.g. roof, wall, ground plane on the path). Thus we need to check them all and select the one with the closest distance to be the one resulting in an observation. We assume that the planar surfaces are opaque, i.e. non-transparent (reflectance > 0 , absorption > 0, transmittance = 0) .

[12] http://en.wikipedia.org/wiki/Point_in_polygon
[13] http://en.wikipedia.org/wiki/Computational_geometry

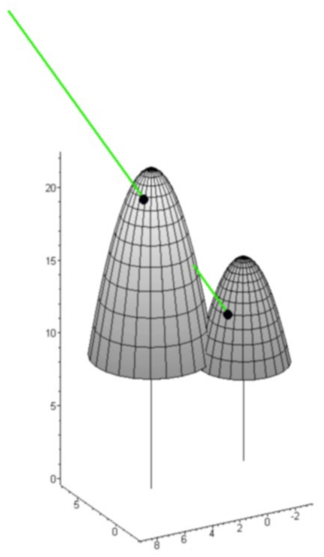


Fig. 13. Simplified trees. Crown envelope and a trunk.

LiDAR pulse in vegetation

We will place a few trees in the scene. Real trees are complex 3D objects. The trunk carries dead and living branches with foliage.

We assume that we know the position, species (sp), dbh, height (h) and the height of the base of the living crown (hc), for each tree. Also, the reflectance of foliage at the wavelength of the LiDAR is known.

We will simplify trees greatly. They consist of a conical trunk and a living crown, which is described by rotation symmetric curve of revolution. The curve gives the radius (r) of the crown at certain height, χ.

We will specify χ (height) so that $\chi = 0$ at the tree top and $\chi = 1$, at the crown base (χ is thus the relative height inside the living crown).

The curve that gives the crown radius is $r(\chi) = a + b*\chi**c$, (4)

where *a* is a small constant (flat top), *b* is the maximum crown radius (minus *a*), and *c* is a shape factor. If *c* == 1, we get a conical shape. We will further assume that *b* = f(*sp*, *dbh*), i.e. the maximum crown radius depends on the *dbh* and species (following the allometry of trees). *c* will depend on species as well. The stem is a cone (*c*==1), and it is likewise defined for $\chi = 0..1$, such that χ=1 at the trunk base. The observed dbh is measured 1.3 m above ground, so the base diameter (radius) is slightly (linearly) larger.

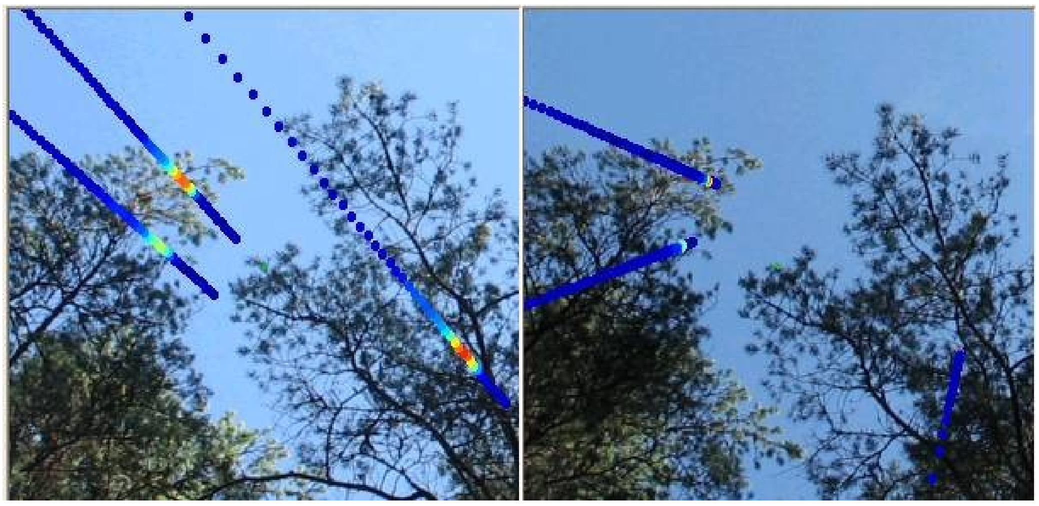


Fig. 14. These two views show parts of crowns of 25-m-tall pines as seen from the ground, from two nearby viewpoints (camera positions). It is clear that the concept of "crown envelope" is vague and a crude approximation of the real (fractal, random) crown shape. The colored lines show three REAL LiDAR pulses. Non-blue color means that from that distance (t, in **p** + t***dir**)) there where photons reflecting and captured by the receiver.

In reality there is no (surface-like) crown envelope such as those in Fig. 13, created by a rotating crown radius-function. The foliage is clumped (branches, annual whorls) inside the ‘envelope’ (Fig. 14). A pulse may intersect the envelope, but still pass thru without intersecting with the foliage.

The question now is how to simulate the real 3D traits of tree crowns to add realism to the simulator? We try this by assuming that the foliage density (kg/m3) in a crown depends on the 3D position. In doing so, we assume that our trees belong to the *Pinaceae* family. It is reasonable to assume that there is a vertical trend in foliage - near the crown base the density is lower than at the top. Similarly the density is lower near the branch tips (there are gaps between branches, both in vertical and horizontal direction, think of a pine, whorls of branches). The foliage is concentrated on the outer hull, near the stem we don't see foliage, except for at the top of the crown.

A point inside the envelope will be defined by coordinates (χ, rc, φ), where χ is the relative height inside the crown (0−1), rc is the relative horizontal distance from the trunk (0−1), and φ is the (azimuth) direction from the trunk (-180 − +180°).

1. Assume that the foliage density inside the crown envelope depends on φ, such that for every 60 degrees, there is a 'hot-spot', a branch (Fig. 15). We call this *thetafactor* (0-1).

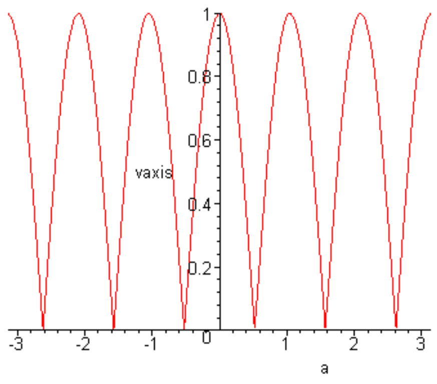


Fig. 15. *thetafactor* = |cos(6*a*+f)|, a = -π − +π

a is the angle that goes from -180 to +180°. The f-term may be used to shift the pattern (rotate it) of branches. We will let it rotate when we inspect the pulse path at approximately 30 cm height intervals (whorls of branches are at those intervals). *Thetafactor* will be responsible for branch-level clumping, although not perhaps perfectly realistic.

2. The dependency of foliage density across height χ, is such that we assume that the density is high and stays high (flat) from the top (χ=0) to the middle of the crown (χ=0.5). Then it drops linearly to zero. *heightfactor* = piecewise (**1**, 0 < χ < 0.5; **1-2χ** 0.5 > χ < 1).

3. The dependency across relative radius *rc*, i.e. horizontally from trunk to the tips of branches, is assumed to be linear: *radiusfactor* = 1 – rc. This assumption puts foliage near the trunk as well. Can be improved.

When we combine these for a point in the canopy (pulse's position), the result becomes:
foliagedensity = *thetafactor* × *heightfactor* × *radiusfactor*

foliagedensity is scaled between (0−1), as we multiply (0,1)-scaled values.

We further assume that the foliage has some nominal reflectance on the wavelength of the LiDAR, refl = 0−1. The amount of photons reflecting back from a sub-ray (to a receiver near by)

photons = *foliagedensity* × *reflectance*

Now, we can assume that *photons* has to be above a limit to be meaningful. Otherwise the pulse just continues down, and we check a next point a bit further down.

If the reflection is strong enough we bring it to the LiDAR receiver

photons = *foliagedensity* × *reflectance x rangebias*

Rangebias dampens the signal from distance R, with respect to some refrence distance Rn: 1/(R/Rn)**2

In tree canopy, we assume that the foliage is randomly oriented - the many **n**-vectors (of leaves, needles) look in different directions.

But, How to define the path of the pulse inside the crown (to get to point(χ, rc, φ))?

Recall that a LiDAR pulse is **p** + t × **dir**, where **p** is the position of the sensor, **dir** is the direction (i,j,k) of the pulse, and t is the length.

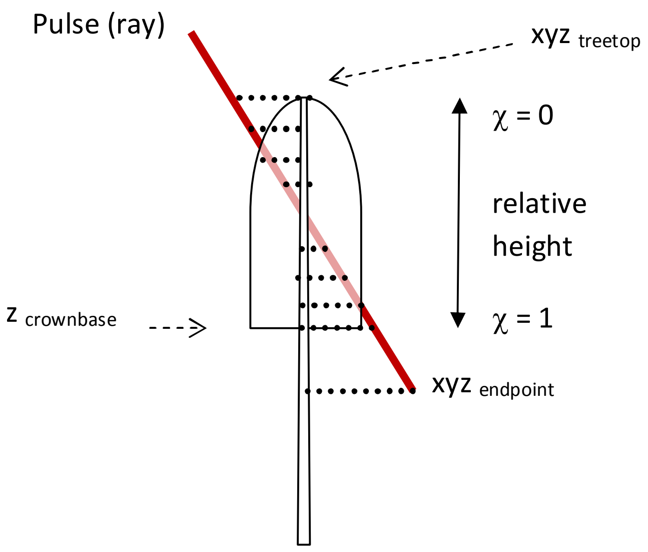


Fig. 16. Testing if the endpoint of the pulse (ray) is inside the crown envelope. The crown radius is a function of height (χ). We test if the endpoint's horizontal distance (dotted lines) is smaller than the crown radius (solid line). The endpoint is first brought to the z of the treetop. Then we take steps in z and compute a new endpoint. For each point inside the crown, we compute the (χ, rc, φ) coordinates, to calculate possible reflection.

1. **Ray's endpoint** for some non-zero t is given by: **p** + t × **dir** = $[x,y,z]_{endpoint}$
2. **Treetop is** known to be at: $[x,y,z]_{treetop}$
3. Solve t by forcing $z_{endpoint} = z_{treetop}$ $\Rightarrow$ we get an endpoint at the treetop's z: $[x,y,z]_{endpoint}$
4. Increase t, e.g. 30 cm, then the new endpoint is: [**p** + (t+0.3) × **dir**] **-** Is it inside the crown?
4.1 Solve the new z, and χ (height) inside crown. Is sqrt(x**2+y**2) > radius(χ)?
(Solve endpoint's distance to trunk and the crown radius at that z, if the endpoint fits the radius, it is inside the crown envelope, and we can test if that point produces a reflection)
5. If the endpoint of pulse is inside, compute (χ, rc, φ), in order to compute the foliage density at that point. If it is above a certain minimum, get the foliage reflectance (0-1, stored for all trees), multiply and trigger an echo (and stop here). Otherwise continue further down with the pulse (store the obtained density).
6. Increase t, another 30 cm, then is the new endpoint **p** + (t+0.3) × **dir** inside the crown?
7. If the endpoint of pulse is inside, compute (χ, rc, φ), to compute the foliage density at that point. Reduce the previous low density – as it has caused losses. if density is above a certain minimum, get the reflectance and trigger an echo and stop. Otherwise continue further down (store the density). (i.e. Loop between 6..7)

If a pulse does not trigger an echo in the crown, it continues its path towards ground. It is attenuated by possible losses. Therefore we must test the intersection for trees before other opaque surfaces.

On the required Math

(figures from Wikipedia)

Vectors

A 2/3D vector that stretches from point **a** to **b**, is defined by the direction (displacement) vector (**b**−**a**) and some scalar t, that defines the length of the vector (to get from **a** to **b**). The length of the direction vector is usually defined to be 1. Then t has the meaning "length" or distance.

A 3D vector has three components (a_1, a_2, a_3), or (a_x, a_y, a_z).

Standard basis vectors are (1,0,0), (0,1,0), and (0,0,1).
i.e. (a_1,a_2,a_3) = a_1(0,0,1)+a_2(0,1,0)+a_3(0,0,1)

They are often denoted i,j,k.

The length of a vector is sqrt(a_1^2+a_2^2+a_3^2) or $\|\mathbf{a}\| = \sqrt{{a_1}^2 + {a_2}^2 + {a_3}^2}$

A unit vector has length one. They are used to indicate direction. Any vector can be divided by its length to create a unit vector. Divide all three components with the length.

$$\hat{\mathbf{a}} = \frac{\mathbf{a}}{\|\mathbf{a}\|} = \frac{a_1}{\|\mathbf{a}\|}\mathbf{e}_1 + \frac{a_2}{\|\mathbf{a}\|}\mathbf{e_2} + \frac{a_3}{\|\mathbf{a}\|}\mathbf{e_3}$$

Dot product

The *dot product* of two vectors **a** and **b** (sometimes called the *inner product*, or, since its result is a scalar, the *scalar product*) is denoted by **a** · **b** and is defined as:

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \, \|\mathbf{b}\| \cos\theta$$

where θ is the measure of the (acute) angle between **a** and **b**.

The dot product can also be defined as the sum of the products of the components of each vector as

$$\mathbf{a} \cdot \mathbf{b} = a_1 b_1 + a_2 b_2 + a_3 b_3.$$

Position, velocity

The position of a point **x** = (x_1, x_2, x_3) in three dimensional space can be represented as a position vector whose base point is the origin

$$\mathbf{x} = x_1\mathbf{e}_1 + x_2\mathbf{e}_2 + x_3\mathbf{e}_3.$$

Given two points **x**=(x_1, x_2, x_3), **y**=(y_1, y_2, y_3) their displacement is a vector

$$\mathbf{y} - \mathbf{x} = (y_1 - x_1)\mathbf{e}_1 + (y_2 - x_2)\mathbf{e}_2 + (y_3 - x_3)\mathbf{e}_3.$$

which specifies the position of *y* relative to *x*. The length of this vector gives the straight line distance from *x* to *y*. Displacement has the dimensions of length.

Rotation Matrices

Rotation matrices are defined in 2D or 3D. They hold the needed coefficients (2 x 2, 3 x 3)to multiply a vector (2D or 3D) such that it is rotated about the origin. The coefficients are directional cosines (-1..+1) between axes of rotation[14]. **R** matrices are needed to describe angular displacements between objects (e.g. in geometry, geodesy, physics, computer graphics).

In LiDAR, the instrument, which is bolted to the bottom frame of the aircraft, gets to rotated about 3 axes, as the aircraft turns in roll, pitch, and heading (also called yaw). In addition, the mirror of the LiDAR moves, but this rotation is considered in the coordinate system of the LiDAR sensor. We simplify things and assume that the plane is flying from South to North, and roll, pitch, heading are considered rotations over x, y and z axis, respectively (Fig. 17).

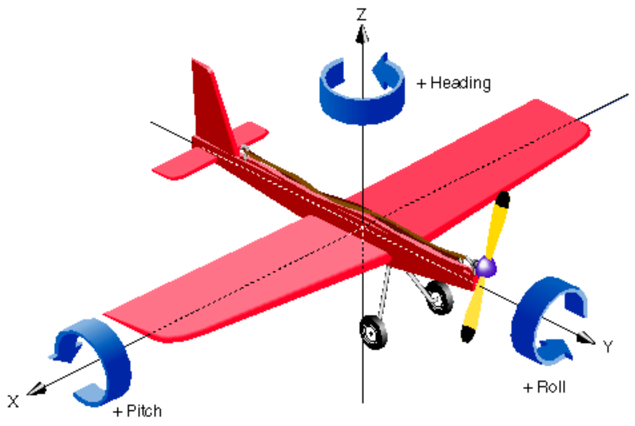
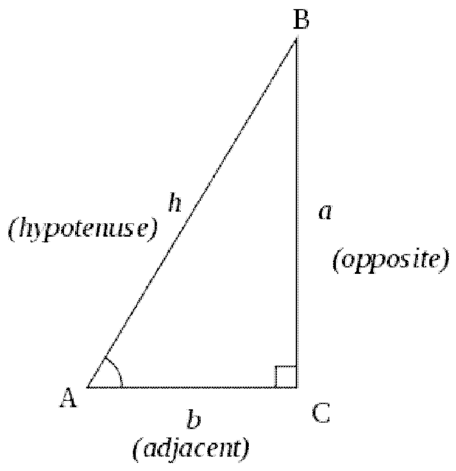


Fig. 17. In terrestrial laser scanning the LiDAR (laser scanner) stays immobile in one position, and on a tripod. Only the mirror moves. In airborne scanning, the platform moves and rotates, constantly.

[14] http://en.wikipedia.org/wiki/Rotation_matrix

Trigonometric functions

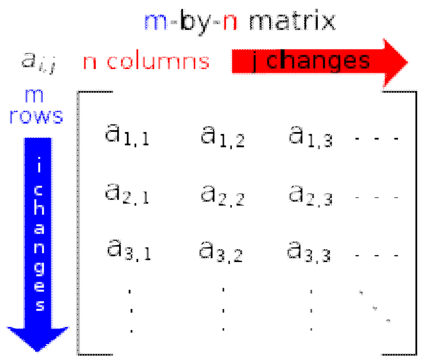


$$\sin A = \frac{\text{opposite}}{\text{hypotenuse}} = \frac{a}{h}. \qquad \cos A = \frac{\text{adjacent}}{\text{hypotenuse}} = \frac{b}{h}.$$

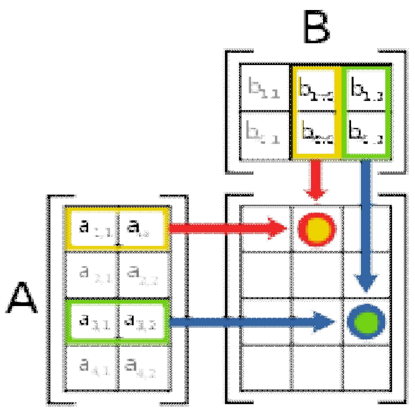
$$\tan A = \frac{\text{opposite}}{\text{adjacent}} = \frac{a}{b}.$$

Matrices & vectors

(http://en.wikipedia.org/wiki/Matrix_theory)


m-by-n matrix
$a_{i,j}$
n columns
j changes
m rows
i changes
$a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,1}$ $a_{3,2}$ $a_{3,3}$

Addition
$$\begin{bmatrix}1 & 3 & 1\\ 1 & 0 & 0\end{bmatrix} + \begin{bmatrix}0 & 0 & 5\\ 7 & 5 & 0\end{bmatrix} = \begin{bmatrix}1+0 & 3+0 & 1+5\\ 1+7 & 0+5 & 0+0\end{bmatrix} = \begin{bmatrix}1 & 3 & 6\\ 8 & 5 & 0\end{bmatrix}$$



$$\begin{bmatrix}1 & 0 & 2\\ -1 & 3 & 1\end{bmatrix} \begin{bmatrix}3 & 1\\ 2 & 1\\ 1 & 0\end{bmatrix} = \begin{bmatrix}5 & 1\\ 4 & 2\end{bmatrix}.$$

Multiply

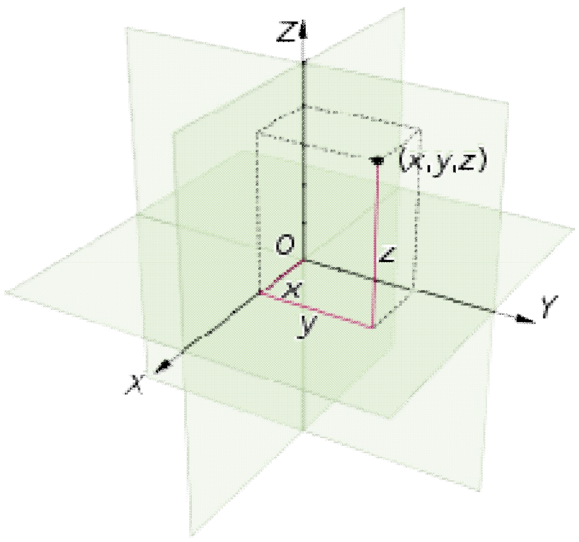
$$\begin{bmatrix}1 & 2 & 3\\ 0 & -6 & 7\end{bmatrix}^T = \begin{bmatrix}1 & 0\\ 2 & -6\\ 3 & 7\end{bmatrix}$$

Transpose

Determinant (http://en.wikipedia.org/wiki/Determinant)

Vectors are 1 x N (row) or N x 1 (column) matrices

Cartesian coordinate system



We can shift (move) points around in x,y, and (z), by simply adding the shift to the coordinates to get the new' position (for z, add c, z' = z + c):

$$(x', y') = (x + a, y + b).$$

We can rotate a point p = (x,y) to a new point (counterclockwise) using these equations.

$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta.$$

or $(x', y') = ((x\cos 2\theta + y\sin 2\theta), (x\sin 2\theta - y\cos 2\theta)).$

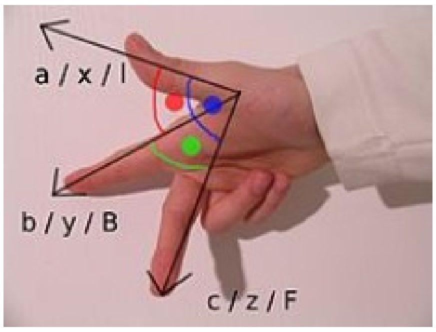
The 2x2 matrix

$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

is called 2D rotation matrix. In 3D, there are three rotations, about the x, y, and z axis. And the matrix has 3 x 3 elements. We use a **R**-matrix version that calls the three rotations omega (x), phi (y), and kappa (z-axis) (in Fig. 17 they are called roll, pitch, heading):

$$
\begin{aligned}
R_{o\_to\_1} &= \begin{pmatrix} \cos\varkappa & \sin\varkappa & 0 \\ -\sin\varkappa & \cos\varkappa & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos\varphi & 0 & -\sin\varphi \\ 0 & 1 & 0 \\ \sin\varphi & 0 & \cos\varphi \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\omega & \sin\omega \\ 0 & -\sin\omega & \cos\omega \end{pmatrix} \\
&= \begin{pmatrix} \cos\varkappa & \sin\varkappa & 0 \\ -\sin\varkappa & \cos\varkappa & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos\varphi & \sin\varphi\sin\omega & -\sin\varphi\cos\omega \\ 0 & \cos\omega & \sin\omega \\ \sin\varphi & -\cos\varphi\sin\omega & \cos\varphi\cos\omega \end{pmatrix} \\
&= \begin{pmatrix} \cos\varkappa\cos\varphi & \cos\varkappa\sin\varphi\sin\omega + \sin\varkappa\cos\omega & -\cos\varkappa\sin\varphi\cos\omega + \sin\varkappa\sin\omega \\ -\sin\varkappa\cos\varphi & -\sin\varkappa\sin\varphi\sin\omega + \cos\varkappa\cos\omega & \sin\varkappa\sin\varphi\cos\omega + \cos\varkappa\sin\omega \\ \sin\varphi & -\cos\varphi\sin\omega & \cos\varphi\cos\omega \end{pmatrix}
\end{aligned}
$$

If we define a 3D coordinate system xyz_{LiDAR} for the LiDAR instrument, we can say that the sensor is installed in the bottom of the airraft with y_{LiDAR}-axis pointing towards flying direction, z_{LiDAR}-axis points down, and x_{LiDAR}-axis points to the left of flying direction (see also Fig. 8). The mirror that shoots the pulse into various directions is attached to the y_{LiDAR}-axis, so that when it is rotated over y_{LiDAR}, the pulse takes only different x_{LiDAR}-values.



Right-hand rule, thumb = x, forefinger = y, middle f = z

But as we fly, xyz_{LiDAR} is unstable in the sense that it is not aligned with the world XYZ, but there are rotations in 3D, that are described by a **R**-matrix, that changes all the time (as the aircraft changes attitude, i.e. omega, phi and kappa fluctuate).

When we fly, the position [X,Y,Z] of the LiDAR (xyz_{LiDAR}) changes all the time (otherwise the plane would fall, or we'd scan the same place constantly).

So where is the pulse pointing to at an arbitrary point in time?

When the LiDAR mirror changes the pulse is pointing in direction $[x, 0, a]_{\text{LiDAR}}$
(*a* is the thickness of the mirror, x takes different values as mirror turns)

The position **p** is $[X,Y,Z]_{\text{world}}$

The rotation of LiDAR is described by the **R**.

The pulse is transmitted, and the 2-way time lapse is t_{pulse}

The distance is then: $(t_{\text{pulse}}/2) \times$ (speed of light)

1) Normalize LiDAR direction vector $[x, 0, \alpha]_{LiDAR}$ (in the system of the LiDAR) length 1: $[x_i, 0_j, z_k]_{LiDAR}$

2) Rotate it to World xyz-system:

> **R** × $[x_i, 0_j, z_k]_{LiDAR}$ = [ri,rj,rk] (It also is a unit vector),
> This is the **dir** vector

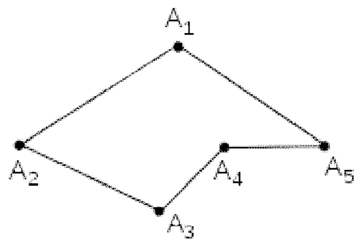
3) Compute the object coordinates keeping LiDAR as the origin, now rotated to match xyz-world.

> **0** + t * **dir** ⇔
> $(t_{pulse}/2)$ × (speed of light) x [ri,rj,rk]

4) Shift everything to the right place by adding the LiDAR position, **p**:

> $[X,Y,Z]_{world}$ + $(t_{pulse}/2)$ × (speed of light) x [ri,rj,rk]
> This is the position of the reflector.

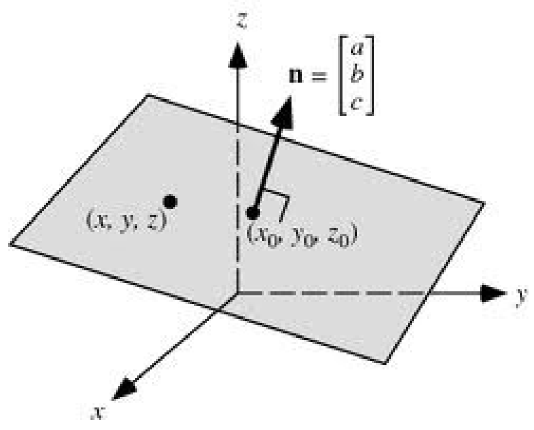
2/3D polyline (closed polygonal chain) – planar polygons.



We will describe real world objects using planar 3D polygons, that are made of xyz points $A_1..A_n$. The points all lie on the same 3D plane, thus it is a planar object, a surface patch.
Simplest possible such surface is a triangle.

A liDAR pulse is **p** + t × **dir**, where **p** is the position of the sensor, **dir** is the direction (i,j,k) of the pulse, and t is the length.

A plane can be defined as Ax+By+Cz+D=0, where [A,B,C] is the normal (n) vector of the 3D plane. Normal sits perpendicular to the plane. [-A,-B,-C] points to the other side of the plane, and there is no saying which is a better candidate (orientation of the plane, up/down sides?).



We can solve a, b and c, if we have three points A_i that belong to the surface. We solve the eqs:

$$ax_1 + by_1 + cz_1 + d = 0$$

$$ax_2 + by_2 + cz_2 + d = 0$$

$$ax_3 + by_3 + cz_3 + d = 0.$$

First we compile a matrix from the 3 points (A_1, A_2, A_3), and compute its determinant D.

$$D = \begin{vmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{vmatrix}$$

Then we set d to some non-zero value and solve three other determinants:

$$a = \frac{-d}{D}\begin{vmatrix} 1 & y_1 & z_1 \\ 1 & y_2 & z_2 \\ 1 & y_3 & z_3 \end{vmatrix} b = \frac{-d}{D}\begin{vmatrix} x_1 & 1 & z_1 \\ x_2 & 1 & z_2 \\ x_3 & 1 & z_3 \end{vmatrix} c = \frac{-d}{D}\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}.$$

a, b, and c make the surface's normal vector (**n**). The normal is important, because the LiDAR reflection is strongest if the surface is perpendicular to the pulse, i.e. the angle between the normal (**n**) and the pulse (**dir**) is zero. Note that the angle can be -90 to 90°. If it is larger, the **n** is pointing in the wrong direction (and we need to choose [–a,-b,-c]).