ESE- RFC Template

1   Status of this Memo

This memo provides information to the NASA Earth Science (ESE) community.   This memo does not specify an ESE standard of any kind. Distribution of this memo is unlimited.

2   Change Explanation

This RFC does not update or change a previous RFC.

3   Copyright Notice

4   Abstract

This document provides information about the Backtrack Orbit Search Algorithm (BOSA), including appropriate use and details of a typical implementation.

5   Table of Contents

ESE-RFC-010v1
Category: Technical Note
Updates/Obsoletes: None

Ross Swick
October 2008
Backtrack Tech Note

6   Introduction

Orbit searching is by far the most accurate way to search for level 0-2 orbital swath data covering a specified area of interest. The idea behind backtrack orbit search is that while spatial search of swath data is difficult in general, Earth Science swath data has a number of characteristics that make the task a lot easier. Remotely sensed data is valuable to Earth scientists because it is frequent, regular, and global. For the purposes of doing Earth Science, scientists have an interest in keeping the data as consistent as possible. Among other things, that means they want the sensor to have a constant field of view. An easy way to accomplish that is to put the satellite in a circular orbit. For this reason (and others), all Earth Science satellites are in a circular orbit.

The Backtrack Orbit Search Algorithm exploits this fact to greatly simplify the orbit model by just modeling an orbit as a great circle under which the Earth rotates. The simplicity of the model allows backtrack to be more efficient than orbit propagator methods, which are designed to work with any satellite. The simplified orbit model relies on only three parameters: inclination, period, and swath width. The accuracy of the method depends on the stability of

those three parameters over the life of the sensor, but there is also a scientific interest in keeping those parameters stable, so they generally stay within reason or the data aren't useful.

As the name implies, backtrack works by tracing the orbit backwards. Backtrack starts with the area of interest and answers the question "In order for the sensor to have seen this area, where must the satellite have crossed the equator?" There is no time dependence, so the speed of the algorithm is independent of the time range searched. There is no cumulative error because backtrack backs up at most one orbit. There is no performance hit from using a lookup table because backtrack calculates the equatorial crossing range, and the subsequent search is a simple, fast, Boolean search on that crossing range.

This document contains detailed descriptions of the general algorithm, a number of special cases, and a number of variations. To augment the descriptions we have also included a worked example and the java code for a working implementation as appendices.

7   Definitions:

The heading of the satellite as it crosses the equator on the ascending pass is the **inclination**.

The angular distance from the pole to the satellite as it passes closest to the pole is the **declination**. This is (inclination - 90).

The amount of time the satellite takes to complete one orbit is the **period**.

The width of the field of view of the sensor is the **swath width**.

The northernmost point achieved by a given orbit is the northern **inflection point**. This is the point at which the orbit goes from ascending to descending. Similarly the southernmost point achieved by a given orbit is the southern inflection point, the point where the orbit goes from descending to ascending.
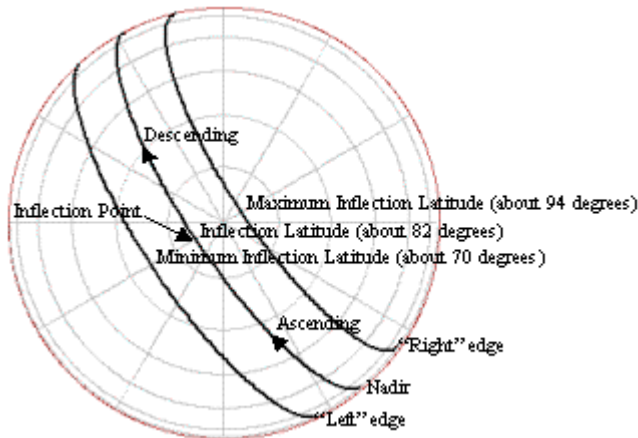
Figure 1: An example orbit illustrating the minimum and maximum latitudes covered by a swath at the inflection point. In this case the swath covers the pole so the right edge is on the other side of the pole. We use circular latitude for the maximum to indicate this.

The northernmost latitude achieved by an orbit, the latitude of the northern inflection point, is the **inflection latitude**. This is (90 - declination).

The southernmost latitude covered by a swath, when the satellite is at the northern inflection point, is the **minimum inflection latitude**. It is the latitude of the southernmost swath edge when the satellite is at the point of inflection.

The northernmost latitude covered by a swath, when the satellite is at the northern inflection point, is the **maximum inflection latitude**. It is the latitude of the northernmost swath edge when the satellite is at the point of inflection. We make this a circular latitude since it could be greater than 90 if the swath covers the pole.   So max_inflection_lat = (inflection_lat + (inflection_lat -min_inflection_lat)).
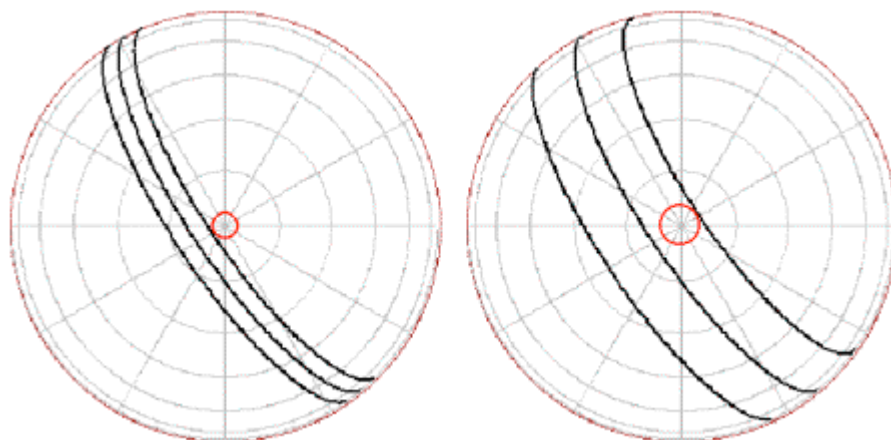

Figure 2: Some sensors never see an area around the pole, which is defined by the maximum coverage latitude (left). Other sensors may see a given area around the pole on every pass, which is defined by the total coverage latitude (right).

The latitude above which all orbits cover the area is the **total coverage latitude** (shown in the left hand image above). Not all sensors have this. Sensors on satellites with a low inclination will never cover the poles, and even some sensors on polar orbiting satellites may have too narrow a swath width to cover the pole. But sensors with a wide enough swath, on polar orbiting satellites may cover the pole, and some region around the pole, on every pass. That region is defined by the total coverage latitude; total_coverage_lat = min (90, (180 - max_inflection_lat)

The latitude above which no orbits cover the area is the **maximum coverage latitude** (shown in the right hand image above). Not all sensors have this. Some sensors with a wide enough swath, on a polar orbiting satellite, may always cover the pole. But sensors with too narrow a swath, or on satellites with a low inclination, may never see the pole and never see some region around the pole. That region is defined by the maximum coverage latitude; max_coverage_lat = min (90, max_inflection_lat)

Circular latitude is a convenient way to encapsulate the latitude and direction of the satellite in a single number. Ideally the circular latitude would simply be the angular distance from the ascending equatorial crossing, but for our purposes it is more convenient to correlate the circular latitude to the actual latitude of the satellite along-track. This creates a (pseudo-) circular latitude that is a discontinuous function. For example: for the swath pictured in figure 1 above the circular latitudes along track would be [0, 82] ascending, [98, 262] descending, and [278, 360] ascending.   Our use of circular latitudes in describing partial orbits is discussed further in section 12 below.

8   Algorithm Description:

This section attempts to describe the backtrack algorithm in sufficient detail to enable a reasonably skilled programmer to implement it. While the Backtrack algorithm uses a *simplified* orbit model it is still not simple. As the algorithm uses concepts and techniques from Spherical Trigonometry, 3-D Cartesian Geometry, ordinary Euclidean Geometry, and basic Algebra a strong math background is highly recommended.

Backtrack was originally developed for data from DMSP satellites. The DMSP satellites are in a sun-synchronous polar orbit.   The algorithm works equally well for satellites that are in shallower orbits and we have attempted to generalize the description, but you may see some of Backtrack's heritage persisting in the description below.

The description focuses on the most basic case of finding crossings for orbits during which the sensor would see the area of interest on the ascending pass.   The differences in the algorithm required for finding descending data are covered in section 9, and a number of special cases and variations are covered in sections 10-12. All of the cases assume a retrograde, circular orbit. We can think of no reason the algorithm wouldn't also work for satellites in a prograde circular orbit, but we have had no opportunity to work with data from such a satellite.

A short statement of the problem is: Given an area of interest and the period, inclination, and swath width of an instrument on a satellite in a retrograde circular orbit; find all ascending equatorial crossings of the satellite for all orbits of the satellite during which the instrument recorded data over the point of interest on the ascending pass.

8.1   Break the perimeter into sample points.

The algorithm works on points so given an area create a set of points along the perimeter that are less than swath width/2 apart.   The points need to be close enough so the resulting equator crossing ranges will overlap and hence be mergable into one searchable crossing range.   Only perimeter points are needed because that's where the extremes of the area are.   I.e. any swath that includes an interior point of the area must also include a perimeter point.   Steps 8.2-8.7 are repeated for each sample point then the crossing ranges are merged prior to the search step.

8.2   Find the orbit with nadir crossing the given point.

Let the point of interest be (lat_p, lon_p). We seek the ascending equator crossing point of nadir at $(0, lon_n)$

First consider a static (non-rotating) sphere. We can approximate the orbital path of the satellite by connecting $(0, lon_n)$ to (lat_p, lon_p) with a great circle. This great circle also contains the inflection point (lat_{inf}, lon_{inf}). Using arcs along the meridian of the point of interest, the great circle orbit, the equator, and the meridian from the north pole to the inflection point we can create two spherical triangles. We do not know the longitude of the inflection point but we do know the latitude because we know the inclination of the orbit. That latitude is (90-declination) where declination = (inclination-90) so we know the length of the great circle arc along lon_{inf} that connects the inflection point to the North Pole.
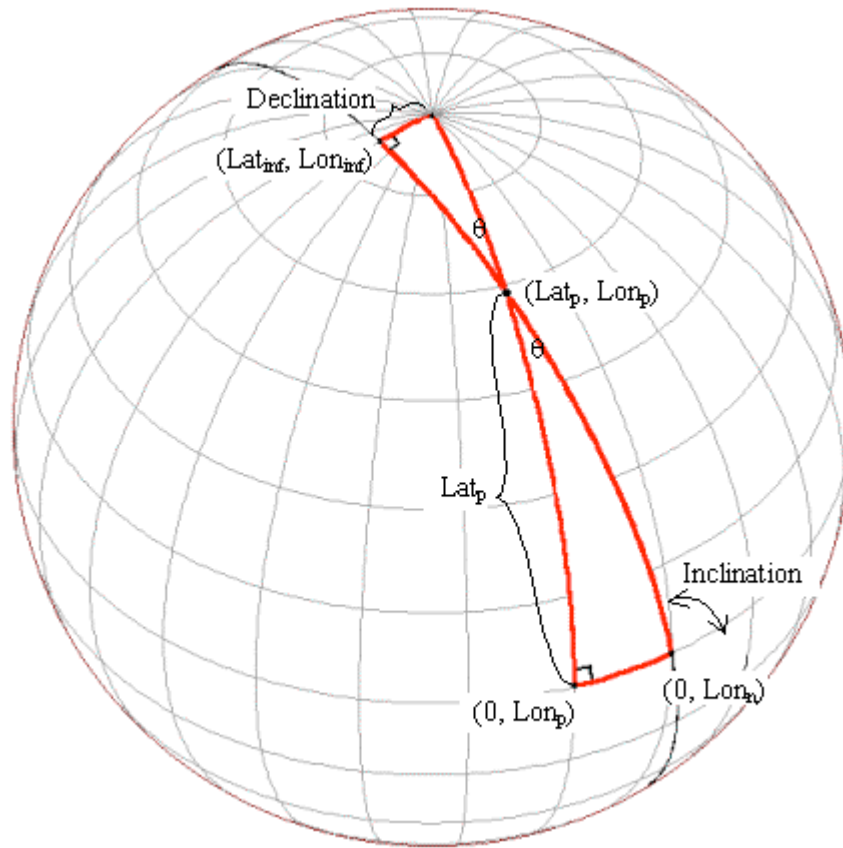
Figure 3: Spherical triangles used to calculate the ascending crossing of a satellite that sees the point of interest on the ascending pass.

Given a spherical triangle with sides alpha, beta, and gamma and opposite interior angles Alpha, Beta, and Gamma. the Law of sines for spherical triangles is:

sin(alpha)/sin(Alpha) = sin(beta)/sin(Beta) = sin(gamma)/sin(Gamma)

Let theta be the angle between the meridian of the point of interest and the orbital great circle. Then by the Law of sines

$sin(90 - lat_{inf})$ / sin(theta) = $sin(90 - lat_p)$ / sin(90)   so

sin(theta) = $sin(90 - lat_{inf})$ / $sin(90 - lat_p$)

Similarly

$sin(lon_n - lon_p)$/ sin(theta) = $sin(lat_p$) / $sin(lat_{inf})$ so

$sin(lon_n - lon_p)$ = sin(theta) $*sin(lat_p$) / $sin(lat_{inf})$ so

 lon_n = asin(sin(theta) $*sin(lat_p)$ / $sin(lat_{inf}$)) $+ lon_p$

ESE-RFC-010v1   **Ross Swick**
Category: Technical Note   **October 2008**
Updates/Obsoletes: None   **Backtrack Tech Note**

Similarly we can find the length of the arc between $(0, lon_n)$ and (lat_p, lon_p)

$sin(L_{arc}) / sin(90) = sin(lat_p) / sin(lat_{inf})$ so

$L_{arc} = asin(sin(lat_p) / sin(lat_{inf}))$

The true equatorial crossing can be found by adjusting for the rotation of the Earth during the time it takes the satellite to travel the distance of L_{arc}.

$RealLon_n = lon_n + [period * L_{arc} * ROTATION\_RATE / circumference]$

Eventually we'll want to adjust everything by that amount - but for now we still need to use the static Earth.

8.3   Convert to Cartesian 3-space and find the orbital plane.

These two points (lat_p, lon_p) and (lat_n, lon_n) lie on the Great Circle that is the orbit. Combined with the origin they define a plane.

Convert to Cartesian

$x = r * cos(lon) * cos(lat);$

$y = r * sin(lon) * cos(lat);$

$z = r * sin(lat);$

The plane of the Great Circle is defined by:

$(y_p z_n - y_n z_p)x - (x_p z_n - x_n z_p)y + (x_p y_n - x_n y_p)z = 0$

Let

$a = (y_p z_n - y_n z_p)$

$b = -(x_p z_n - x_n z_p)$

$c = (x_p y_n - x_n y_p)$

So the plane of the orbit on a static sphere is: $ax + by + cz = 0$

8.4   Find points on the swath edges:

Given the swath width, find points (lat_{edge}, lon_{edge}) on each edge of the swath by going half that distance perpendicular to the ground track. This only works for downward looking sensors. Forward looking and backward looking sensors can also use this method by creating a virtual satellite that is downward looking. For side viewing sensors simply dividing the swath width in half won't work. Instead the swath width can be broken apart into left and right distances from nadir. This would require one additional parameter in the database table that contains the sensor
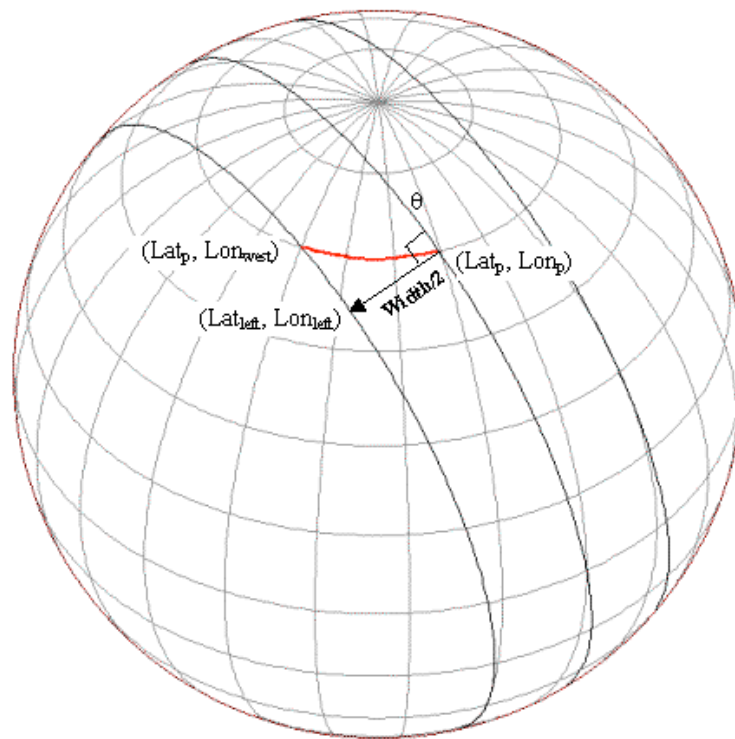
information.

Figure 4: The cross-track swath width is constant but the longitudinal swath width, along a constant line of latitude, varies. Additionally the longitudinal half swath widths, to the right and left of nadir, are different.

For the left edge of the swath we need to find the point (lat_{left}, lon_{left}) that is distance (width/2) from the point of interest (lat_p, lon_p) along the heading (180 + theta).

$lat_{left} = $ asin(($\sin(lat_p)*\cos(width/2)$) + ($\cos(lat_p)*\sin(width/2)*\sin(180+theta)$))

$lon_{left} = lon_p - $ acos(($\cos(width/2) - \sin(lat_p)*\sin(lat_{left})$) / ($\cos(lat_p)*\cos(lat_{left})$))

For the right edge of the swath we need to find the point (lat_{right}, lon_{right}) that is distance (width/2) from the point of interest (lat_p, lon_p) along the heading (theta).

$lat_{right} = $ asin(($\sin(lat_p)*\cos(width/2)$) + ($\cos(lat_p)*\sin(width/2)*\sin(theta)$))

$lon_{right} = lon_p + $ acos(($\cos(width/2) - \sin(lat_p)*\sin(lat_{right})$) / ($\cos(lat_p)*\cos(lat_{right})$))

8.5 Find the planes of the swath edges:

Convert the edge point to Cartesian (x_{edge}, y_{edge}, z_{edge}).

On a static Earth the swath edge is a small circle parallel to the Great Circle defined by nadir. In Cartesian space that defines a plane parallel to the plane

$ax+by+cz=0$

That plane is: $ax + by + cz = d$ where $d = (ax_{edge} + by_{edge} + cz_{edge})$

and the equation for the plane of the small circle which is the other edge of the swath is:

$ax + by + cz = -d$

8.6 Find the intersects with the given latitude:

We want the point where each small circle intersects the latitude of the given point, and the sphere. That is:

$ax + by + cz = d$: swath edge

$x^2 + y^2 + z^2 = r^2$: sphere

$z = z_p$: lat = latp

Three equations, three unknowns. We're keeping the latitude constant so we only care about the longitude which is atan(y/x). Substituting $z = z_p$ and rearranging we get:

$ax + by + cz_p - d = 0$ and

$x^2 + y^2 + z_p^2 - r^2 = 0$

10

Solving this set of equations gets fairly hairy fairly quickly. Fortunately we have computers to do the hard part for us. The following is from a session with Mathematica:

In[11]:= Solve[{$(x^2) + (y^2) - (r^2) + (z^2) == 0$,

   $(ax) + (by) + (cz) - d == 0$},

   {x, y}]

Out[11]= {

> {x -> $(d - (b^2 d/(a^2+b^2)) - cz + (b^2 cz/(a^2+b^2)) -$

> $(b * \text{Sqrt}[-(a^2 d^2) + a4r^2 + a^2 b^2 r^2 + {}^2a^2 cdz - a4z^2 -$

> $a^2 b^2 z^2 - a^2 c^2 z^2]) / (a^2 + b^2)) / a$,

> y -> $(bd - bcz + \text{Sqrt}[-(a^2 d^2) + a4r^2 + a^2 b^2 r^2 + {}^2a^2 cdz -$

> $a4z^2 - a^2 b^2 z^2 - a^2 c^2 z^2]) / (a^2 + b^2)$},

> {x -> $(d - (b^2 d/(a^2+b^2)) - cz + (b^2 cz/(a^2+b^2)) +$

> $(b * \text{Sqrt}[-(a^2 d^2) + a^4 r^2 + a^2 b^2 r^2 + {}^2a^2 cdz - a4z^2 -$

> $a^2 b^2 z^2 - a^2 c^2 z^2]) / (a^2 + b^2)) / a$,

> y -> $(bd - bcz - \text{Sqrt}[-(a^2 d^2) + a^4 r^2 + a^2 b^2 r^2 + {}^2a^2 cdz -$

> $a4z^2 - a^2 b^2 z^2 - a^2 c^2 z^2]) / (a^2 + b^2)$}}

Grouping the constants together makes the solution slightly less messy.

Let

$P = cz_p - d$ (Constants in the Plane equation)

$S = z_p{}^2 - r^2$ (Constants in the Sphere equation)

In[12]:= Solve[{$(x^2) + (y^2) + S == 0, ax + by + P == 0$},{x,y}]

Out[12]= {

$$
\{ x \to \cfrac{-P + \cfrac{b^2 P}{a^2 + b^2} - \cfrac{b * \text{Sqrt}[-a^4 S - a^2 b^2 S - a^2 P^2]}{a^2 + b^2}}{a},
$$

$$
> \quad y \to \cfrac{-bP + \text{Sqrt}[-(a^4 S) - a^2 b^2 S - a^2 P^2]}{a^2 + b^2} \},
$$

$$
> \quad \{ x \to \cfrac{-P + \cfrac{b^2 P}{a^2 + b^2} + \cfrac{b * \text{Sqrt}[-(a^4 S) - a^2 b^2 S - a^2 P^2]}{a^2 + b^2}}{a},
$$

$$
> \quad y \to \cfrac{-bP - \text{Sqrt}[-(a^4 S) - a^2 b^2 S - a^2 P^2]}{a^2 + b^2} \}\}
$$

So we solve for x and y and convert back to spherical:

$$
lon = atan(y/x)
$$

There are at most two solutions for each edge of the swath. Of the four possible solutions we need the closest two, one east, one west. The other two are where the planes of the swath edges cross the latitude on the other side of the sphere.

8.7   Find crossings for the extreme orbits:

Find the distance, in longitude, from the original point to the edge points (adjusted for the dateline). Let:

$D_E = lon_{east} - lon_p$

$D_W = lon_p - lon_{west}$

Find the point D_E west of (lat_p, lon_p), which is (lat_p, lon_p - D_E), and the point D_W east of (lat_p, lon_p), which is (lat_p, lon_p + D_W)

Note: An error is introduced into the algorithm here. The sensor will see the points (lat_p, lon_p - D_E) and (lat_p, lon_p + D_W) some time prior to, or after, it passes over (lat_p, lon_p). To be completely accurate we should adjust for the rotation of the Earth during the intervening time. But the error is always small, makes less difference as it gets bigger, and adjusting for it means taking the sensor geometry into account. So we tolerate it in an effort to keep the algorithm simple. Still, we could adjust at this point using some "generic" sensor geometry to increase the accuracy somewhat without sacrificing the generality of the algorithm.

Find the nodal crossing of the orbit (on a rotating sphere) that passes through (lat_p, lon_p - D_E) using step 1 above and call it $lon_{nadirWest}$.   Any orbit with a nodal crossing at $lon_{nadirWest}$ includes the given point (lat_p, lon_p) on its east edge.

Find the nodal crossing of the orbit (on a rotating sphere) that passes through (lat_p, lon_p + D_W) using step 1 above and call it $lon_{nadirEast}$. Any orbit with a nodal crossing at $lon_{nadirEast}$ includes the given point (lat_p, lon_p) on its west edge.

 Any orbit with a nodal crossing in the range [$lon_{nadirWest}$, $lon_{nadirEast}$] must include the given point (lat_p, lon_p)

8.8   Merge the ranges found in steps 8.2-8.7

Each sample point created in step 8.1 produces a crossing range and those ranges should overlap. Consequently they can be merged into a single crossing range that defines where the satellite must have crossed the equator on the previous ascending pass if the sensor were to see the area of interest.

8.9   Query the database or other information storage system.

The crossing range can be used to create a spatial clause that searches on crossings in a database query. For our database that clause looks like this:

 where nodal_crossing between ($lon_{nadirWest}$, $lon_{nadirEast}$)

Combined with other clauses (temporal, parameter, quality flags, etc.) a single, relatively small, query is all that is required to return record locators matching the users criteria.

9 Descending Passes:

To find the ascending crossing range for orbits during which the sensor would see the area of interest on the descending pass the algorithm is similar with some minor adjustments.
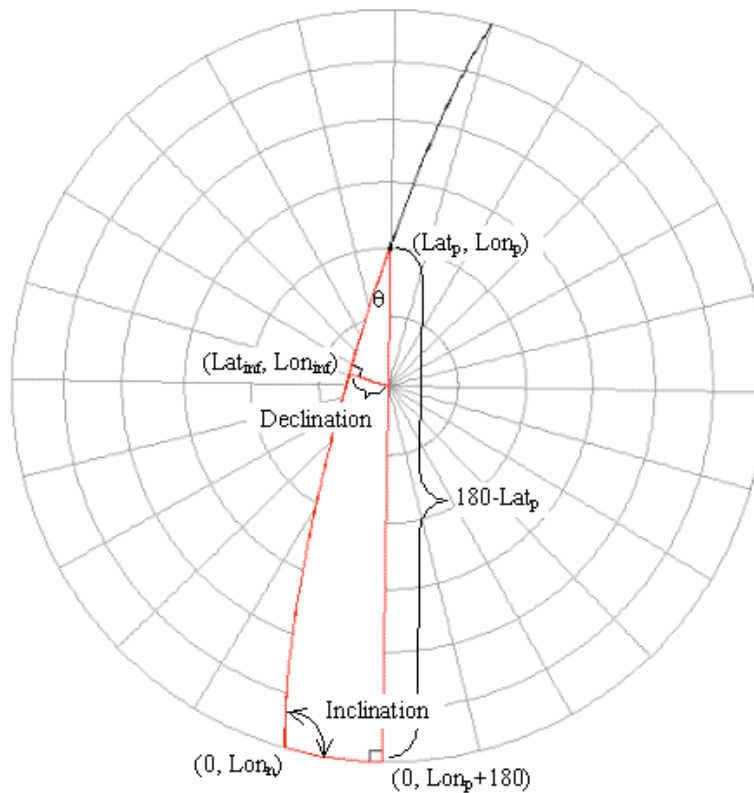

Figure 5: Spherical triangles used to calculate the ascending crossing of a satellite that sees the point of interest on the descending pass.

The angle theta can be found the same way - but the point of interest is now on the opposite side of the sphere as the ascending crossing so adjustments have to be made.

ESE-RFC-010v1
Category: Technical Note
Updates/Obsoletes: None

Ross Swick
October 2008
Backtrack Tech Note

$\sin(lon_n - (lon_p + 180)) / \sin(theta) = \sin(180 - lat_p) / \sin(inclination)$ so

$\sin(lon_n - lon_p - 180) = \sin(theta) * \sin(lat_p) / \sin(180 - lat_{inf})$ so

$lon_n = asin(\sin(theta) * \sin(lat_p) / \sin(lat_{inf})) + lon_p + 180$

Similarly we can find the length of the arc between $(0, lon_n)$ and (lat_p, lon_p)

$\sin(L_{arc}) / \sin(90) = \sin(180 - lat_p) / \sin(180 - lat_{inf})$ so

$L_{arc} = asin(\sin(lat_p) / \sin(lat_{inf}))$

10   Correcting the Longitudinal Half Swath Widths for Earth Rotation:

The radius of the small circle that defines the swath edge is proportional to the radius of the sphere by the cos of the angular distance (distance on the sphere) from the parallel great circle. That is: $radius_{sc} = radius * \cos(distance)$

The distance between two points in Cartesian space is $sqrt((x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2)$ so the distance between left edge point and the west edge point is:

$D = sqrt((x_{left} - x_{west})^2 + (y_{left} - y_{west})^2 + (z_{left} - z_{west})^2)$

Those two points, and the center of the small circle, create an isosceles triangle in the plane of the small circle. We can use ordinary Euclidean trigonometry to find the vertex angle of that triangle and hence the length of the arc on the small circle.

$\cos(v) = (2 * radius_{sc}^2 - D^2) / 2 * radius_{sc}^2 = 1 - (D^2 / 2 * radius_{sc}^2)$

$v = acos(1 - (D^2 / 2 * radius_{sc}^2))$

 The correction would be the distance the earth rotated in the time it took the satellite to go v degrees.

The correction for the right/east edge can be found in a similar fashion.

11   Special Cases: The absolute value of the latitude of …

11.1   …the point of interest is greater than the maximum coverage latitude.

If the point of interest is above (or below) the maximum coverage latitude the sensor never sees that point so the algorithm returns with no results for that point. If every sample point in the area of interest is above (or below) the maximum coverage latitude it's no use even running the search.

11.2   …the point of interest is greater than the total coverage latitude.

If the point of interest is above (or below) the total coverage latitude the sensor sees that point during every orbit and the algorithm returns a crossing range of [0, 360]. Once that happens there is no need to test further sample points and really no need for a crossing range clause in the search as every orbit is a match.

11.3   …the point of interest is greater than the minimum inflection latitude.

If the point of interest is above (or below) the minimum inflection latitude it means one of the swath edges is not present at that latitude making it impossible to find the longitudinal half swath width. Since the algorithm distinguishes between ascending and descending passes the cutoff in either direction is the longitude of the inflection point.

11.4   …the point of interest is greater than the inflection latitude.

If the point of interest is above (or below) the inflection latitude, and didn't qualify for cases 1 or 2 above, it is never the case that the satellite passes over that point, but there is still a limited range of crossings for which the sensor will see the point. The algorithm can still use the point itself to find the longitudinal half swath widths (with appropriate cutoffs as noted in case 3) but needs to substitute the inflection latitude for the latitude of the points in order to find the initial cross track edges points and the crossings.

12   Variations:

12.1   Forward and backward looking sensors:

The algorithm as described assumes a downward looking sensor. As mentioned above the algorithm could be adjusted to work with forward and backward looking sensors by creating a virtual satellite. Technically the algorithm doesn't care where the satellite is; it cares where the sensor's field of view is. If a sensor is pointed x degrees forward of the satellite location that is functionally equivalent to a downward looking sensor on a virtual satellite x degrees ahead. Because the Earth rotates during the time it takes the satellite to travel x degrees the inclination of the virtual satellite may be slightly different from the inclination of the real satellite so that adjustment to the input parameters needs to be made.

One unfortunate side effect is that orbits are often defined by where the satellite is so the orbital data from a sensor looking x degrees forward may start and end at x degrees north latitude instead of at the equator. One can adjust for this by recording the start/end circular latitude (discussed below) in the inventory and incorporating that into the search. Or one could avoid storing those extra fields in the inventory for each and every orbit by adjusting the algorithm to use x degrees north as the reference latitude.

12.2   Side viewing sensors:

 It was also mentioned above that the algorithm could easily be adjusted to work for side viewing sensors. As described the algorithm assumes the ground track of the satellite (nadir) is in the center of the swath, so the distance to the left and right edges of the sensors field of view is just half the swath width. To accommodate side viewing sensors we just have to be more explicit about those distances. Instead of just the swath width as input we would need the two distances from nadir to the swath edge.

For example a sensor with a 1400-kilometer wide swath centered 200 kilometers left of nadir would have left and right distances of 900 kilometers and 500 kilometers respectively. The same sensor at a more extreme angle, centered 800 kilometers left of nadir, would have left and right distances of 1500 kilometers and -100 kilometers respectively where the negative value of the right distance indicates the right edge of the swath is actually left of nadir. The sign is a matter of convention and one can adopt whatever convention is convenient.

12.3   Partial Orbits:

The algorithm as described assumes full orbits that start and end at the same reference latitude, but sometimes it is more practical to slice the data into partial orbits which means the start and end of the data become important. While the orbit the data is part of may intersect the area being searched the data itself may not. Unfortunately just the geographic location isn't sufficient; the direction of the satellite is also significant. To combine the start/end latitude of the data with the direction of the satellite into a single number we have adopted a convention of circular latitudes. Starting at the equator the circular latitudes are [0, 90] ascending, [90, 270] descending, and [270, 360] ascending.

Care must be taken to ensure the circular latitudes are in sync with the indexed crossing longitude of the data. For example, a descending half orbit that starts at its northernmost point and ends at its southernmost point—for a satellite with an inclination of 98.78 degrees indexed to the previous ascending crossing—would have start/end circular latitudes of {98.78, 261.22}. The same half orbit indexed to the next ascending crossing would have start/end circular latitudes of {-261.22, -98.78}. And, an ascending half orbit indexed to the included ascending crossing would have start/end circular latitudes of {-81.22, 81.22}.

12.4   Multiple Orbits:

It is also sometimes convenient to include more than one orbit in a data granule and circular latitudes can also be used to compensate. For example, a two orbit granule—that starts/ends on

ESE-RFC-010v1   **Ross Swick**
Category: Technical Note   **October 2008**
Updates/Obsoletes: None   **Backtrack Tech Note**

the equator during the ascending pass and indexed to the initial ascending crossing—would have start/end circular latitudes of {0, 720}. Indexed to the included ascending crossing the start/end circular latitudes would be {-360, 360}. And, indexed to the ending ascending crossing the start/end circular latitudes would be {-720, 0}

Obviously the algorithm has to compensate accordingly by computing the start/end circular latitudes of the search area for the different possibilities and creating multiple clauses for the search. To keep the algorithm general some indication of how many orbits are in each granule, and which crossing the granules are indexed to should be added as an input parameter. One possible convention is to always index to the first included ascending crossing, or the previous ascending crossing if none are included (as with descending half orbits), and indicate the number of orbits or fraction of an orbit in each granule with a single floating point number as an input parameter.

12.5 Lookup tables:

One advantage of the algorithm is it figures out everything mathematically rather than using lookup tables. This makes it faster, more flexible, more accurate, and less expensive. It's faster because every database query takes a certain amount of time just to initialize, which is often orders of magnitude larger than the amount of time it takes to compute the answer. It's more flexible because changes in the orbit of a satellite can be accommodated by adjusting a few input parameters rather than recreating an entire lookup table. This is especially true during the pre-launch timeframe when people are preparing the system. Efforts expended creating lookup tables prior to launch may be wasted if the satellite doesn't achieve the expected orbit.

The Backtrack algorithm is also demonstrably more flexible in adding another sensor to the system. Instead of requiring a new lookup table for each new sensor, which are costly to create and can be quite large, the algorithm requires only the addition of a few input parameters in a single row of an existing table. Moreover, the accuracy of lookup table methods is directly related to the size of the table. Depending on context, doubling the accuracy of a lookup table can mean doubling, or quadrupling, its size. This not only increases the required storage but also increases the time it takes to query the table. The backtrack algorithm, on the other hand, starts out as accurate as the input parameters and the computations allow.

Finally, it would appear that one has to do the math in order to create the lookup tables in the first place. So, actually going ahead with the creation and use of the lookup tables is extra effort. Often that effort is deemed necessary because the orbital mechanics and spherical trigonometry involved in doing the math are quite complex, prone to error during the coding process, and computationally expensive. But the backtrack algorithm simplifies the orbital mechanics by limiting itself to circular orbits and simplifies the math involved by using a hybrid of spherical trigonometry, Cartesian solid geometry, Euclidean planar geometry, and simple algebra.

Still, there are places in the algorithm where lookup tables could be used. The longitudinal half swath widths, for example, are constant for a given latitude and direction. The correction due to the rotation of the Earth of the longitudinal half swath widths is also constant for a given latitude

ESE-RFC-010v1   **Ross Swick**
Category: Technical Note   **October 2008**
Updates/Obsoletes: None   **Backtrack Tech Note**

and direction. If the desired accuracy is such that the lookup table would be small, a search for corrected widths based on latitude and direction might be faster than doing the math.

Alternatively, the entire algorithm, or pieces of it, could be used to generate lookup tables for legacy systems that require them. This has in fact been done for one sensor, which reduced the cost of generating the table by about 90%. And, with a table generator written, the costs associated with changing that table or generating another for a different sensor are virtually nothing.

13   Informative References

Bates, Roger R., Donald D. Mueller, and Jerry E. White, *Fundamentals of Astrodynamics*, Dover Publications, New York, 1971.

Selby, Samuel M. CRC Standard Mathematical Tables, 24th Edition, CRC Press, 1976.

Swick, R., and K. Knowles. "The Backtrack Orbit Search Algorithm", http://www.geospatialmethods.org/bosa, November, 2004.

14   Authors' Address

Ross Swick, National Snow and Ice Data Center, Cooperative Institute for Research in Environmental Sciences, University of Colorado, Boulder, CO., 80309, USA. Tel: 303-492-6069 fax 303-492-2468, email: swick@nsidc.org

Ken Knowles, Operational Systems Inc., 1521 Easy Rider Ln Suite 101, Boulder, CO, 80304, USA. Tel: 303-579-9015 fax 303-444-1250, email: ken.knowles@opsysinc.com

15   Appendix A: Glossary of acronyms

Acronym	Description
BOSA:	Backtrack Orbit Search Algorithm
CIRES:	Cooperative Institute for Research in Environmental Sciences
ECS:	EOSDIS Core System
EOSDIS:	Earth Observing System Data and Information System
NASA:	National Aeronautics and Space Administration
NOAA:	National Oceanographic and Atmospheric Administration

16   Appendix B: Worked Example: Pittsburgh

The picture below is a relatively ordinary orbit for a polar orbiter. The ascending equatorial crossing is at 77.75E. The satellite has an inclination of 98.78 degrees (declination of 8.78 degrees) and a period of 101 minutes. The swath width is 1400 km.
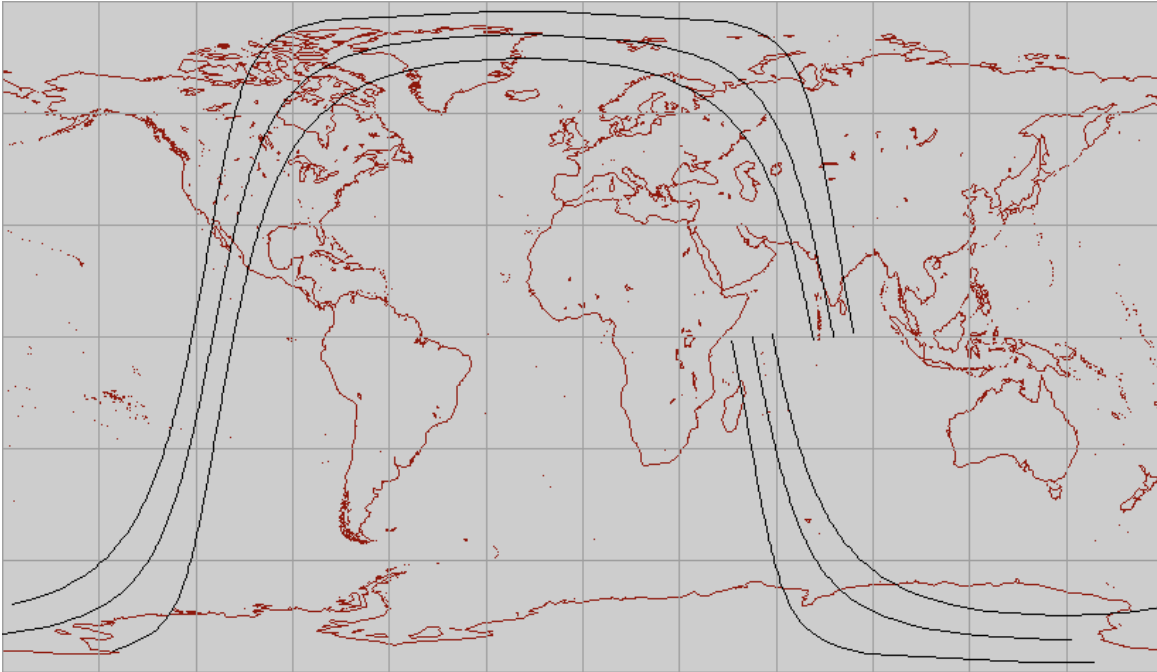
We want to find all the orbits for which this sensor sees Pittsburgh on the ascending pass. Given the location of Pittsburgh (40.50, -80.22) the algorithm looks at what the swath would look like if the ground track of the satellite went right through Pittsburgh on the ascending pass.

1) Find the orbit with nadir crossing the given point.

$\sin(\text{theta}) = \sin(90 - \text{lat}_{inf}) / \sin(90 - \text{lat}_p)$
$\sin(\text{theta}) = \sin(8.78) / \sin(49.5) = 0.15264 / 0.76041 = 0.2007$
$\text{theta} = 11.5779$

$\text{lon}_n = \text{asin}(\sin(\text{theta}) * \sin(\text{lat}_p) / \sin(\text{lat}_{inf})) + \text{lon}_p$
$\text{lon}_n = \text{asin}(0.2007 * \sin(40.5) / \sin(81.22)) - 80.22$
$\text{lon}_n = \text{asin}(0.2007 * 0.64945 / 0.98828) - 80.22$
$\text{lon}_n = \text{asin}(0.13189) - 80.22$
$\text{lon}_n = 7.57883 - 80.22 = -72.64117$

Then correct for rotation

$\sin(L_{arc}) / \sin(90) = \sin(lat_p) / \sin(lat_{inf})$ so
$L_{arc} = \mathrm{asin}(\sin(lat_p) / \sin(lat_{inf}))$
$L_{arc} = \mathrm{asin}(\sin(40.5) / \sin(81.22))$
$L_{arc} = \mathrm{asin}(0.64945 / 0.98828)$
$L_{arc} = \mathrm{asin}(0.65715)$
$L_{arc} = 41.0830$ degrees.

Which means the true equatorial crossing can be found by adjusting for the rotation of the Earth during the time it takes the satellite to travel 41.0830 degrees.

$RealLon_n = lon_n + [$ period $* L_{arc} *$ ROTATION_RATE / circumference$]$
$RealLon_n = -72.64117 + [101 * 41.0830 * (15/60) / 360]$
$RealLon_n = -72.64117 + [101 * 41.0830 * (15/60) / 360]$
$RealLon_n = -72.64117 + 2.8815$
$RealLon_n = -69.75965$

And if that's all we wanted we'd be done. But we need to find the crossings for all the orbits that cover Pittsburgh, which means we need to take the swath width into account, which means we need to go back to the static sphere.

2) Convert to Cartesian 3-space and find the orbital plane.

The two points (lat_p, lon_p) and (lat_n, lon_n) lie on the Great Circle that is the orbit on the static sphere. Combined with the origin they define a plane.

x = r * cos(lon) * cos(lat);
y = r * sin(lon) * cos(lat);
z = r * sin(lat);

The radius is just a constant multiplier so we'll use a radius of 1.

$x_p = \cos(lon_p) * \cos(lat_p) = \cos(-80.22) * \cos(40.5) = 0.16986 * 0.76041 = 0.12916$
$y_p = \sin(lon_p) * \cos(lat_p) = \sin(-80.22) * \cos(40.5) = -0.98547 * 0.76041 = -0.74936$
$z_p = \sin(lat_p) = \sin(40.5) = 0.64945$

$x_n = \cos(lon_n) * \cos(lat_n) = \cos(-72.64117) * \cos(0.0) = 0.29836$
$y_n = \sin(lon_n) * \cos(lat_n) = \sin(-72.64117) * \cos(0.0) = -0.95445$
$z_n = \sin(lat_n) = \sin(0.0) = 0$

The plane of the Great Circle is defined by:
$(y_p z_n - y_n z_p)x - (x_p z_n - x_n z_p)y + (x_p y_n - x_n y_p)z = 0$

Let

$a = (y_p z_n - y_n z_p) = (-0.74936 * 0) - (-0.95445 * 0.64945) = 0.61987$
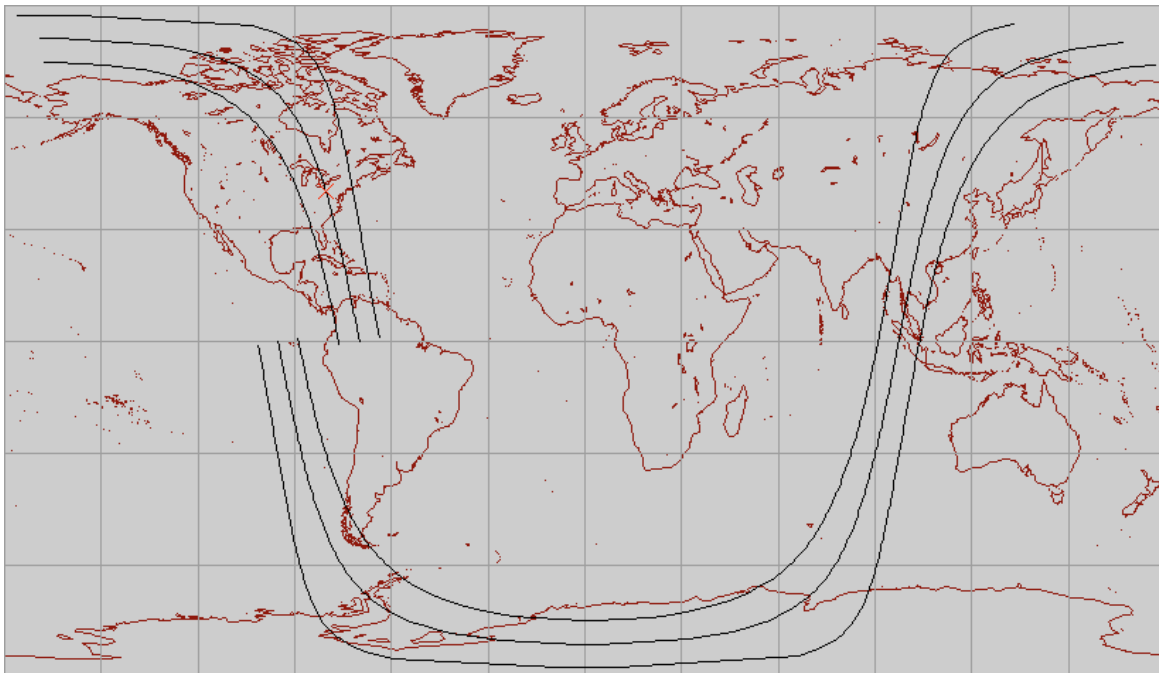$b = -(x_p z_n - x_n z_p) = -((0.12916 * 0) - (0.29836 * 0.64945)) = 0.19377$
$c = (x_p y_n - x_n y_p) = (0.12916 * -0.95445) - (0.29836 * -0.74936) = -0.123277 + 0.22358 = 0.10030$

And the plane of the orbit on a static Earth is: $ax + by + cz = 0$

Sanity check:

$(0.61987 * 0.12916) + (0.19377 * -0.74936) + (0.10030 * 0.64945) = 0.08006 - 0.14520 + 0.06514 = 0$

$(0.61987 * 0.29836 + (0.19377 * -0.95445) + (0.10030 * 0) = 0.18494 - 0.18494 + 0 = 0$

3) Find points on the swath edges:

We have the width in kilometers but we need to keep the units consistent. For this example we'll use 111 km/degree but you may want to be more precise.

width = 1400 / 111 = 12.6126 degrees.

For the left edge of the swath we need to find the point (lat_{left} , lon_{left}) that is distance (width/2) from the point of interest (lat_p , lon_p) along the heading (180 + theta).

lat_{left} = asin((sin(lat_p)*cos(width/2)) + (cos(lat_p)*sin(width/2)*sin(180 + theta)))
lat_{left} = asin((sin(40.5)*cos(6.3063)) + (cos(40.5)*sin(6.3063)*sin(191.5779)))
lat_{left} = asin((0.64945 * 0.99395) + (0.7604 * 0.10984 * -0.2007))
lat_{left} = asin(0.6455 + (-0.01676))
lat_{left} = asin(0.62874) = 38.957

lon_{left} = lon_p - acos((cos(width/2) - sin(lat_p) * sin(lat_{left})) / (cos(lat_p)*cos(lat_{left})))
lon_{left} = -80.22 - acos((cos(6.3063) - sin(40.5) * sin(38.957)) / (cos(40.5)*cos(38.957)))
lon_{left} = -80.22 - acos((0.99395 - 0.64945 * 0.62874) / (0.7604*0.77762))
lon_{left} = -80.22 - acos((0.99395- 0.40834) / (0.5913))
lon_{left} = -80.22 - acos((0.58561481) / (0.5913))
lon_{left} = -80.22 - acos(0.990384)
lon_{left} = -80.22 - 7.952
lon_{left} = -88.1721

For the right edge of the swath we need to find the point (lat_{right} , lon_{right}) that is distance (width/2) from the point of interest (lat_p , lon_p) along the heading (theta).

lat_{right} = asin((sin(lat_p)*cos(width/2)) + (cos(lat_p)*sin(width/2)*sin(theta))
lat_{right} = asin((sin(40.5)*cos(6.3063)) + (cos(40.5)*sin(6.3063)*sin(11.5779))
lat_{right} = asin(0.64945 * 0.99395) + (0.7604 * 0.10984 * 0.2007)
lat_{right} = asin(0.6455 + 0.01676)
lat_{right} = asin(0.66226) = 41.472

lon_{right} = lon_p + acos((cos(width/2) - sin(lat_p)*sin(lat_{right})) / (cos(lat_p)*cos(lat_{right})))
lon_{right} = -80.22 + acos((cos(6.3063) - sin(40.5)*sin(41.472)) / (cos(40.5)*cos(41.472)))
lon_{right} = -80.22 + acos((0.99395 - 0.64945 * 0.66226) / (0.7604 * 0.74927))
lon_{right} = -80.22 + acos((0.99395 - 0.4301) / (0.7604 * 0.74927))
lon_{right} = -80.22 + acos(0.56384524 / 0.5697)
lon_{right} = -80.22 + acos(0.98964)
lon_{right} = -80.22 + 8.253 = -71.967

4) Find the planes of the swath edges:

Convert the edge points to Cartesian and then the planes of the swath edges are given by ax + by + cz = d.

x_{left} = cos(lon_{left}) * cos(lat_{left}) = 0.031887 * 0.77762 = 0.02479
y_{left} = sin(lon_{left}) * cos(lat_{left}) = -0.99949 * 0.77762 = -0.77722
z_{left} = sin(lat_{left}) = 0.62874

x_{right} = cos(lon_{right}) * cos(lat_{right}) = 0.30956 * 0.74928 = 0.23195
y_{right} = sin(lon_{right}) * cos(lat_{right}) = -0.95088 * 0.74928 = -0.71247
z_{right} = sin(lat_{right}) = 0.66225

Perform a sanity check. The two points we found should be the same distance from the orbital plane.

ax_{left} +by_{left} + cz_{left} = d = -(ax $_{right}$ +by_{right} + cz_{right})
(0.61987 * 0.02479) + (0.19377 * -0.77722) + (0.10030 * 0.62874) = d = -((0.61987 * 0.23195) + (0.19377 * -0.71247) + (0.10030 * 0.66225))
(0.015367 - 0.15060 + 0.06306) = d = -(0.14378 - 0.138055 + 0.06642)
-0.07217 = d = -0.07214

There's some cumulative rounding error, but it's pretty good.

5) Find the intersects with the given latitude:

Now we have those planes we can find the points where the planes intersect the latitude of our point of interest. That is:
ax + by + cz = d   : swath edge
x^2 +y^2 + z^2 = r^2 : sphere
z = z_p : lat = lat_p

Three equations, three unknowns. We're keeping the latitude constant so we only care about the longitude which is atan2(y, x). Substituting z = z_p and rearranging we get:

ax + by + cz_p -d = 0 and
x^2 -y^2 + z_p^2 - r^2 = 0

Let

planar_const = cz_p -d = (0.10030 * 0.64945) - 0.07215 = 0.0651398 - 0.07215 = -0.007010

sphere_const = z_p^2 - r^2 = 0.64945^2 - 1 = 0.42179 - 1 = -0.57821

scale = $(a^2 + b^2)$ = $(0.61987^2 + 0.19377^2)$ = 0.42179

radical = SQRT(($-a^4$ * sphere_const) - (a^2 * b^2 * sphere_const) - (a^2 * planar_const^2))

radical = SQRT((-0.61987^4 * -0.57821) - (0.61987^2 * 0.19377^2 * -0.57821) – (0.61987^2 * 0.007010^2))

radical = SQRT((0.147639 * -0.57821) - (0.38424 * 0.0375468 * -0.57821) - (0.384239 * 0.00004914))

radical = SQRT(0.0853666 + 0.0083418 - 0.00001888)

radical = SQRT(0.093689) = 0.3060

firstXterm = (b^2 * planar_const)/scale = (0.19377^2 * -0.007010) / 0.42179 = -0.000624

secondXterm = ((b*radical)/scale) = (0.19377 * 0.30609) / 0.42179 = 0.14062

So

x = (-planar_const + firstXterm - secondXterm) / a = (0.007010 + -0.000624 - 0.14062) / 0.61987 = -0.22555

y = (-(b*planar_const) + rad) / scale = (-(0.19377 * -0.007010) + 0.30609) / 0.42179 = 0.7289

lon = atan2(y, x) = atan2(0.7289, -0.22555) = atan(-3.2316) + 180 = -72.806 +180 = 107.194

OR

x = (-planar_const + firstXterm + secondXterm) / a = (0.007010 + -0.000624 + 0.14062) / 0.61987 = 0.237156

y = (-(b*planar_const) - rad) / scale = (-(0.19377 * -0.007010) - 0.30609) / 0.42179 = -0.72247

lon = atan2(y, x) = atan2(-0.72247, 0.237156) = atan(-3.0464) = -71.827

On the other side of the orbit let:

planar_const = cz_p +d = (0.10030 * 0.64945) + 0.0727 = 0.06514 + 0.0727 = 0.13784

radical = SQRT(($-a^4$ * sphere_const) - (a^2 * b^2 * sphere_const) - (a^2 * planar_const^2))

radical = SQRT((-0.61987^4 * -0.57821) - (0.61987^2 * 0.19377^2 * -0.57821) - (0.61987^2 * 0.13784^2))

radical = SQRT((0.147639 * -0.57821) - (0.38424 * 0.0375468 * -0.57821) - (0.384239 * 0.0189998))

radical = SQRT(0.0853666 + 0.0083418 - 0.0073005)

radical = SQRT(0.086408) = 0.29395

ESE-RFC-010v1   **Ross Swick**
Category: Technical Note   **October 2008**
Updates/Obsoletes: None   **Backtrack Tech Note**

firstXterm = $(b^2$ * planar_const)/scale = $(0.19377^2$ * 0.13784) / 0.42179 = 0.01227
secondXterm = ((b*radical)/scale) = (0.19377 * 0.29395) / 0.42179 = 0.13504

So

x = (-planar_const + firstXterm - secondXterm) / a = (-0.13784 + 0.01227 - 0.13504) / 0.61987 = -0.42043
y = (-(b*planar_const) + radical) / scale = (-(0.19377 * 0.13784) + 0.29395) / 0.42179 = 0.633587
lon = atan2(y, x) = atan2(0.633587, -0.42043) = atan(-1.506998) + 180 = -56.43 + 180 = 123.567

OR

x = (-planar_const + firstX + secondX) / a = (-0.13784 + 0.01227 + 0.13504) / 0.61987 = 0.015277396
y = (-(b*planar_const) - radical) / scale = (-(0.19377 * 0.13784) - 0.29395) / 0.42179 = -0.76023
lon = atan2(y, x) = atan2(-0.) = atan(-49.762) = -88.849

The points we're looking for are the closest two. The other two are where the edge planes intersect the latitude plane on the other side of the sphere. The closest points are (40.5, -88.85) to the West and (40.5, -71.83) to the East. The sensor sees those points on the edge of the swath sometime prior to, or after, it sees the point of interest and a correction for the rotation of the Earth could be introduced here for increased accuracy. But the error is small so we skip that step here.

**6) Find crossings for the extreme orbits:**

The longitudinal distance from the point of interest (40.5, -80.22) to the swath edges is 8.63 degrees to the west and 8.39 degrees to the east. So if the satellite were to pass over the point 8.63 degrees east of Pittsburgh (40.5, -71.59) the sensor would see Pittsburgh on the west edge of the swath. And if the satellite were to pass over the point 8.39 degrees west of Pittsburgh (40.5, -88.61) the sensor would see Pittsburgh on the east edge of the swath. We have only to find the equatorial crossings for those two orbits and we'll know the range of crossings for which the sensor is guaranteed to see Pittsburgh somewhere in the scan on the ascending pass.

To find those crossings we use the same method as above:

$\sin(\text{theta}) = \sin(90 - \text{lat}_{\text{inf}}) / \sin(90 - \text{lat}_p)$
$\sin(\text{theta}) = \sin(8.78) / \sin(49.5) = 0.15264 / 0.76041 = 0.2007$
$\text{theta} = 11.5779$

$\text{lon}_{\text{ne}} = \text{asin}(\sin(\text{theta}) * \sin(\text{lat}_e) / \sin(\text{lat}_{\text{inf}})) + \text{lon}_e$
$\text{lon}_{\text{ne}} = \text{asin}(0.2007 * \sin(40.5) / \sin(81.22)) - 71.59$
$\text{lon}_{\text{ne}} = \text{asin}(0.2007 * 0.64945 / 0.98828) - 71.59$
$\text{lon}_{\text{ne}} = \text{asin}(0.13189) - 71.59$
$\text{lon}_{\text{ne}} = 7.57883 - 71.59 = -64.01$

$\text{RealLon}_{\text{ne}} = \text{lon}_{\text{ne}} + 2.88 = -64.01 + 2.88 = -61.13$
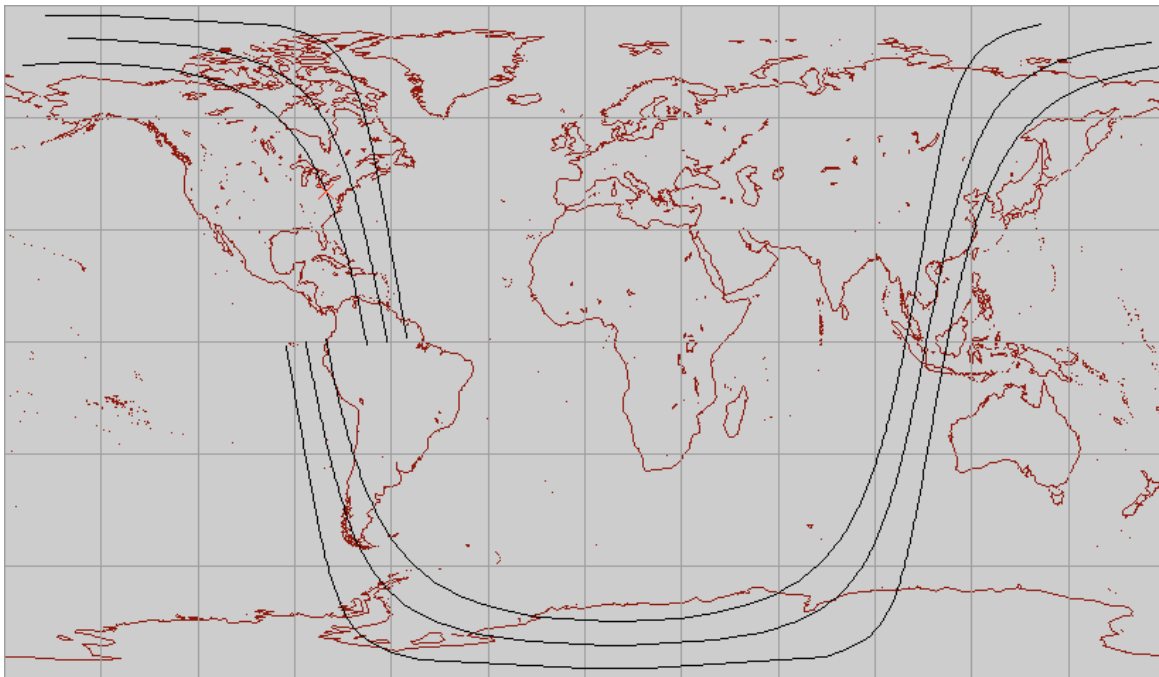
$\text{lon}_{\text{nw}} = \text{asin}(\sin(\text{theta}) * \sin(\text{lat}_w) / \sin(\text{lat}_{\text{inf}})) + \text{lon}_w$
$\text{lon}_{\text{nw}} = \text{asin}(0.2007 * \sin(40.5) / \sin(81.22)) - 88.61$
$\text{lon}_{\text{nw}} = \text{asin}(0.2007 * 0.64945 / 0.98828) - 88.61$
$\text{lon}_{\text{nw}} = \text{asin}(0.13189) - 88.61$
$\text{lon}_{\text{nw}} = 7.57883 - 88.61 = -81.03$

$\text{RealLon}_{\text{nw}} = \text{lon}_{\text{nw}} + 2.88 = -81.03 + 2.88 = -78.14$
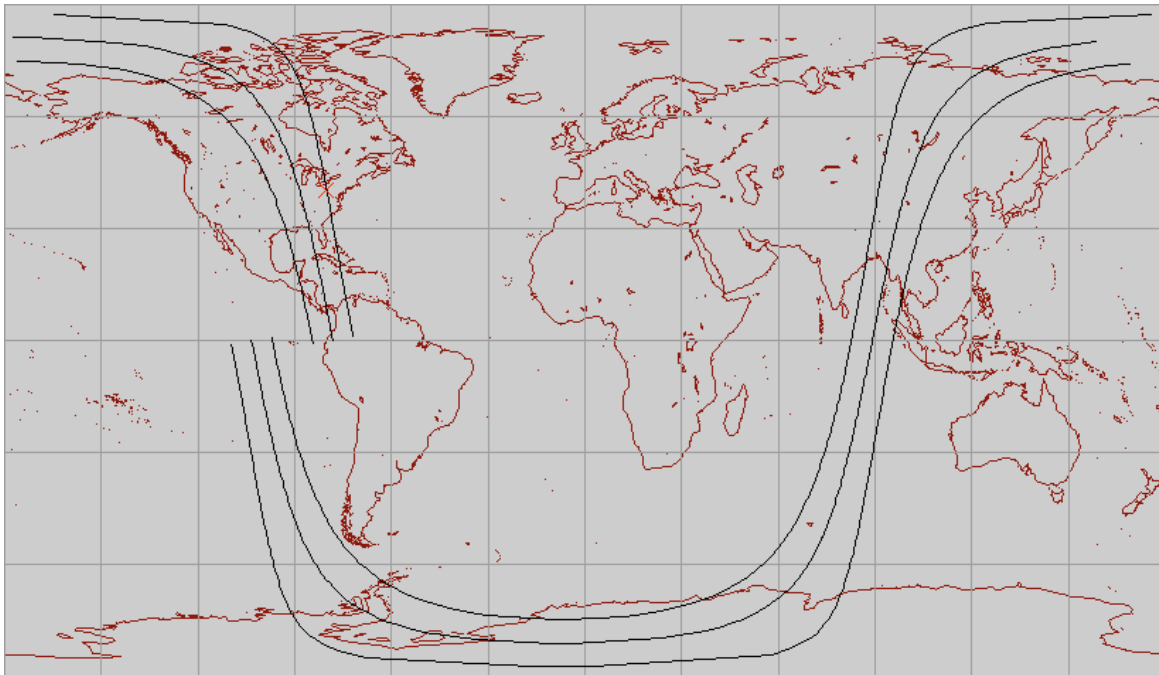
And we're done. For any orbit with an ascending crossing in the range [-78.14, -61,1] the sensor will see Pittsburgh on the ascending pass.

16.1 Special Cases:

Correcting the longitudinal swath widths for earth rotation:

The radius of the small circle that defines the swath edge is proportional to the radius of the sphere by the cos of the distance from the parallel great circle. That is: $radius_{sc}$ = radius * cos(distance) In this example the swath width is 12.6126 degrees and we're using a radius of 1. So $radius_{sc}$ = 1 * cos(6.3063) = 0.99395.

The distance between two points in Cartesian space is $sqrt((x_1 - x_2)^2 + (y_1-y_2)^2 + (z_1-z_2)^2)$ so the distance between left edge point and the west edge point is:
$sqrt((x_{left}-x_{west})^2 + (y_{left} -y_{west})^2 + (z_{left}-z_{west})^2)$ =
$sqrt((0.02479 - 0.01528)^2 + (-0.77722 + 0.76023)^2 + (0.62874 - 0.64945)^2)$ =
$sqrt((0.00951)^2 + (-0.01699)^2 + (-0.02071)^2)$ =
$sqrt(0.00009044 + 0.00028866 + 0.0004289)$ =
$sqrt(0.000808) = 0.028425$

Those two points, and the center of the small circle, create an isosceles triangle in the plane of the small circle. We can use ordinary Euclidean trigonometry to find the vertex angle of that triangle and hence the length of the arc on the small circle.

$\cos(v) = (2r^2 - D^2)/ 2r^2 = 1 - (D^2/2r^2) = 1 - (0.000808/1.97587) = 1 - 0.0004089 = 0.99959$
$v = \mathrm{acos}(0.99959) = 1.63862$ degrees

The correction would be the distance the earth rotated in the time it took the satellite to go 1.63862 degrees

$101 * 1.63862 * (15/60) / 360 = 0.11493$ degrees

Which would give us a west distance of 8.74 instead of 8.63

Similarly to the right/east the distance between the right edge point and the east edge point is:

$\mathrm{sqrt}((x_{right}\text{-}x_{east})^2 + (y_{right}\text{-}y_{east})^2 + (z_{right}\text{-}z_{east})^2) =$
$\mathrm{sqrt}((0.23195 - 0.237156)^2 + (-0.71247 + 0.72247)^2 + (0.66225 - 0.64945)^2) =$
$\mathrm{sqrt}((0.005206)^2 + (0.01)^2 + (0.0128)^2) =$
$\mathrm{sqrt}(0.0000271 + 0.0001 + 0.00016384) =$
$\mathrm{sqrt}(0.00029094) = 0.017057$

$\cos(v) = (2r^2 - D^2)/ 2r^2 = 1 - (D^2/2r^2) = 1 - (0.00029094/1.97587) = 1 - 0.000147246 = 0.999853$
$v = \mathrm{acos}(0.999853) = 0.9832$ degrees

The correction would be the distance the earth rotated in the time is took the satellite to go 0.9832 degrees

$101 * 0.9832 * (15/60) / 360 = 0.06896$ degrees

Which would give us an east distance of 8.32 instead of 8.39

17   Appendix C: Java Implementation

The Orbit class is part of the NSIDC Sphere's package. The Sphere's package is designed to perform various computations on the sphere and contains an implementation of the Backtrack algorithm in the Orbit class. This is the implementation that both NSIDC and ECHO are currently using. Only the Orbit class in attached as an example. The source code for the entire Spheres package is available from NSIDC.