Idea about using twkb with zoom levels twkbZL

The focus when developing twkb have been to get a format that is slimmed and flexible. The idea have been that it shall be used to send the whole geometry to the client and let the client do any necessary simplifications. The problem is that the download size to get an overview map is too big.

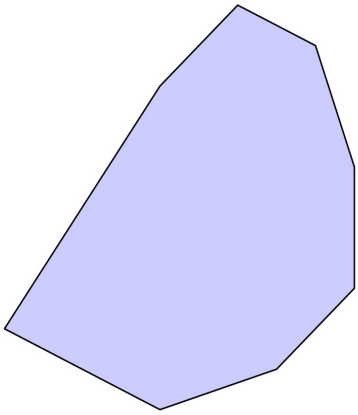
So here comes an idea how to handle that and in the same way handle smaller amounts of data in the client when zoomed out.

What we want is to send a simplified version of a polygon, and then send more details without resending what is already sent.

So, we need some way to tell the client from what data set to get what points. We also want the overhead, both in computing and size to be minimal.

Here comes an example to show the idea.

We have a polygon with 8 vertex points.



Point nr	x	y
1	50	50
2	60	45
3	65	30
4	65	15
5	55	5
6	40	0
7	20	10
8	40	40

If we encode this with delta coordinates like in twkb we get coordinates like this:

Point nr	deltaX	deltaY
1	50	50
2	10	-5
3	5	-15
4	0	-15
5	-10	-10
6	-15	-5
7	-20	10
8	20	30

But that we can forget about for a little while.

Now we simplify this polygon into 3 zoom levels with the PostGIS function ST_SimplifyPreserveTopology.

Our third zoom level is the original polygon, the secon uses tolerance 10 and the first zoomlevel uses tolerance 5.

Then our first and second zoom level has polygons like this with the original point numbers:

First:

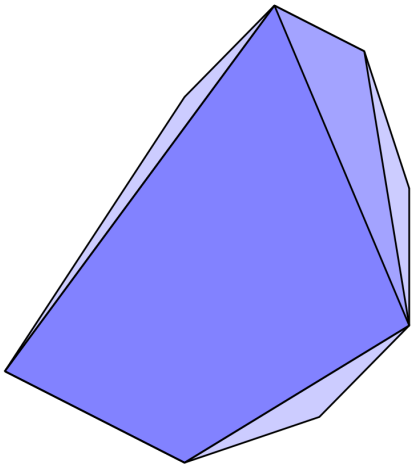
Point nr	**x**	**y**
1	50	50
4	65	15
6	40	0
7	20	10

Second:

Point nr	**x**	**y**
1	50	50
2	60	45
4	65	15
6	40	0
7	20	10

Third (the original)**:**

Point nr	**x**	**y**
1	50	50
2	60	45
3	65	30
4	65	15
5	55	5
6	40	0
7	20	10
8	40	40



So, what we want is when the user asks for zoom level 1 we send point 1,4,6,7.
Then when he asks for zoom level 2 if he already has level 1 we send point 2
And finally when the user wants the full resolution polygon we send point 3,5,8

If the user asks for zoom level 3 at once we have to send all three zoom levels.

Ok, so this is what we want, how to do it?

First zoom level
For the first zoom level we don't have to do anything. We just send the coordinates. We don't care about the original numbers, we just send them.

Like full coordinates:
50 50 65 15 40 0 20 10

or delta coordintes:
50 50 15 -35 -25 -15 -20 10

Second zoom level
Ok, now we have to tell the client where to get the points from to get it right. So in fron of our coordinates we give the client a map. Each point in the polygon of theis zoom level is represented by a bit.
The points that is in a lower zoom level is unset and the points that is in this zoom level (2 in this case) is set.

So the map for our second zoom level will look like this:
01000

This means that ther will be 3 unused bits in this byte. The client omnly reads 5 bits because it knows the number of points in the polygon from the nPoints property.

So what we send here is our map and the coordinates for point 2:

01000 60 45

If we send it like delta coordinates it will be like this:
01000 0 -15

Third zoom level
Follows the same pattern as the second so we send:

full coordinates:
00101001 65 30 55 5 40 40

delta coordiantes
00101001 5 -15 -10 -10 20 30

How the client reads this

When the client gets this zoom levels it has to put everything together again.

Here is the rules.

With full coordiantes:
The client always starts at the highest available zoom level
Reads the first byte of that map.
If that byte is set it reads a point from that zoom level.
If it is unset, it counts how many unset bits there is and moves the cursor to before the first set byte. Let's say there were 2 unset bits, meaning that the lower zoom levels holds those points.
Then it starts reading the map of the next level. If the first bit is set it reads a point from htat data set and reds the next bit. If that is unset it jumps to the even lower zoom level and tries there.
When the 2 points is collected in the lower zoom levels it is vack on the highest level, reading the point there.
Then checking the next bit and all continues until the whole map is read.

With delta coordinates
Then it gets a little more complicated, because we need to decide from what point we get the delta value.
The most slimmed version is like this (and is used in the encoding examples above).
When reading multiple points from the same zoom level, our point is registered as the delta values from our last read point.

If we get from a lower zoom level to a higher, we also have our point as the delta values from the last read point.

But if we get from a higher zoom level to a lower our point is represented as the delta values from the last point in the same zoom level.

This is because a zoom level is only aware of the zoom levels lower than itself, not the higher.

So if we look at all our points a nd tells in what zoom level it belongs and how to get the value representing the point (3-2 means the deltavalue between point 3 and 2)

1	**2**	**3**
1-0		
	2-1	
		3-2
4-1		
		5-4
6-4		
7-6		
		8-7

What about sizes?

If we look at our polygon encoded as twkb:

The whole polygon as twkb uses 23 bytes. Then the id of the polygon is included and an extra vertex point in the end as used in PostGIS

If we divide the polygon to the 3 levels discussed above we get something like this.

Zoom level 1
Ordinary twkb with 5 points (including the last one identical with the first and ID of the geometry)
uses 15 bytes

zoom level 2
A twkb with the first header byte but not the second (because we already know the type and number of dimensions) It still holds the ID of course, If it shall hold a “nrings” property or a map of rings in the zoom level I am not sure of but we use 1 byte for that anyway)
uses 7 bytes

zoom level 3
uses 13 bytes (can also be 14 if we use a varINT way of encoding the map, but that is not necessary)

So in total we use 35 bytes instead of the 23 for the original polygon. That is a 52% overhead.

But in real world the overhead will probably be much lower.

1) Most real world polygons holds a lot more vertex points than 8 which makes the initial byte and nPoints byte and nRings byte a lot smaller part of the whole. If the polygons only holds 8 points they should not be simplified.
2) In real world each vertex point will need more space than 1 byte because thee is higher precission and longer distances (higher delta values) This also makes the bytes always needed being a smaller part of the whole.

About computing speed

If encoding in PostGIS it should be fast. The function can produce all zoom levels at once returning them as a set. That shouldn't give any big overheads.

Reading them should also be fast as long as all the zoom levels for one polygon is stored so they are fast accessed.

The client will need some structure (or object if you prefer) that holds information about the available zoom leveld, their respective maps, number of points and coordinates.

Then it should be very fast to put it back together again. Probably faster than simplifying at the client from the original polygon.