Topological Error Correction, Encoding, PostGIS Management, Topology, & Simplification for the American Red Cross Administrative Boundaries GIS Web Platform

Open source solutions & methods

American Red Cross
International Services
Geospatial Information Management

Estabrook, S. & Kunce, D.

TABLE OF CONTENTS

PURPOSE

The purpose of this project was to build a comprehensive and hierarchical administrative boundary geometry database. The geometry database (GeoDB) is a backbone for a new project management system being deployed by the American Red Cross (ARC). The in-house geotagging ensures data continuity and quality, as well as establishing a cutting-edge implementation of a global geospatial project management and support system. By managing our projects including geospatial considerations, we are able to better measure and evaluate different perspectives _____ and provide the needed information management support.

There are many benefits to a **propriety**, yet open source system and process. The current availability of data and software is unprecedented, and many GIS applications and features are underutilized, undocumented, and/or unrealized.

The exciting

SCOPE

To provide global coverage of geospatial information, the largest and most detailed dataset, (GADM) is the only available, free dataset.

METHODS

Our methods for this process were iteratively considered; the processes were developed independently as needed. This was accomplished by proper management, well scoped direction, and flexibility. The first process developed was topology correction, the small, sometimes imperceivable data errors that are a legacy of non-topologically considerate data producers. This process was developed knowing that a solution would be elusive, and failures inevitable.
The initial process used established tools and methods, but did not produce the desired result. GRASS and QGIS were pivotal in understanding the breadth of the errors, but failed to correct them. Research continued until PPRepair was discovered, and its utility realized. Over the course of development, the data errors were persistent, and a new approach had to be developed. PPRepair also failed to correct the errors. The errors are so small that it became apparent that the tools to fix these type of errors were not developed to be scaled to such large and intricate geometries. The new approach was to correct the smallest amount of errors with each process. This is less intuitive as it requires a step-wise methodology that attempts to fix as little as possible, and move from one process to the next, each time moving closer to the goal of total error correction. This process worked well, and resulted in the full correction of topological errors.
Once the topology was corrected, the movement from a file-base dataset to a database structure was required for the deployment of the GeoDB. The database setup, PostgreSQL with PostGIS

Open source Platforms, Programs, Datasets, and Format Standards
"-" denotes major open source structure as a dependence
"*" denotes experimental; documented, not tested

GADM	Dataset
GDAL (see below)	Platform

MapShaper (GDAL?)	Program
-GDAL	Platform
-Node.js	Platform
PPRepair	Program
-GDAL	Platform
PostgreSQL	Program/Platform
Ubuntu	Platform
-Linux	Base-Platform

GDAL-supported Open source:

GeoJSON	Format Standard
GRASS	Program
PostGIS	Program
QGIS	Program
Shapefile (for all intents and purposes)	Format Standard
*TopoJSON	Format Standard

RESULTS

Success!
How do we know? Indicators and Validation

TUTORIAL

TUTORIAL TABLE OF CONTENTS

Topology Data Model, Validation, & Simplification
-PostGIS Topology
-PostGIS Topology Simplification
-MapShaper Simplification
—Validation
—Application

[Advanced Hierarchical Topology Diagram]
[Flow Chart]

Tutorial Appendix
-Topology Discussion
-PostgreSQL & PostGIS Reference Commands

INTRODUCTION

Processes are numbered followed by example input/output metrics and indicators.

The process can be completed with different methods depending upon the data. In this paper we will use GADM v. 2 as the example dataset, as its correction is the purpose for our use of these methods. The processes are most likely extendible (with a certain about of caution) beyond the GADM dataset, but some of the reasoning and construction of the methods here are based on our needs for a specific GADM setup. Our setup here, the complete documentation following, is GADM v. 2, topologically corrected, re-encoded, loaded into PostGIS, dissolved into six layers (one for each GADM administrative layer), and topologically represented for hierarchical topologic simplification. Other, alternative methods are also demonstrated, as well as certain failures and issues.

The GADM data needs to be exact and the edits very well documented and understood, so a comprehensive approach with the use of topological measures and tools in GRASS is implemented. This way the data can be validated and rebuilt with topological considerations before run through a program like PPRepair. It may seem that the GRASS procedure is useless or unnecessary, but using the GIS to validate and measure indicators of the data issues helps us as the data specialists to be more confident in our (hopeful) conclusion of topology correction.

If only using PPRepair because the needs of the fix are for visualization, due to time-constraints, or simplification, please skip the GRASS instructions [steps 1 – 9] and go to step 10 and review the methods of PPRepair to see if it will meet your needs.

PostGIS is used as the database-server solution and for different geoprocessing needs. The main use in this paper is for storing and dissolving the topologically-corrected dataset to build new, dissolved datasets. This is a requirement for our own setup, since the different administrative levels must be independently queried to allow for a flexible and nested geographic perspective. There are alternatives to many of these processes which are explored within each section. ————————-Specifically, topological simplification in PostGIS is explained completely but is not scalable so was not used for our dataset.————————

Two types of topology can be used with the PostGIS setup. The preferable method, but more difficult to implement, is a hierarchical topology of the geometries. This is best described in Figure ___ (show each layer built from the child-layer). The other option breaks each geometry layer into separate topologies, so their relationships can be independently evaluated or simplified. This is described in Figure ____. This will be covered more in the Topology Discussion

The tool MapShaper is preferable for simplification as it allows for the preservation of small areas and has an automated intersection correction algorithm as well as preserving topology. This means if the simplification produces a shape that creates an overlap or intersection that didn't exist in the original data

it will locally fix the data such that it represents the original relationship – removing the overlap or intersection. See steps ___ for directions for MapShaper.

These processes were developed on Ubuntu and Desktop GRASS and QGIS, but Ubuntu processes can be easily adapted for Mac OS X. Most of the processes were completed on a Mac, while some others used Ubuntu 13.04 Server. For a dataset like GADM, a server-grade setup is recommended.

Quick Command-line Tutorial:

NOTE: Do not type the $ into a Terminal command. This is simply showing where the prompt ends and the command begins.

Ubuntu & Mac OS Basics

Print current directory:
```
$ pwd
```
List items in directory:
```
$ ls
```
List items and their size:
```
$ ls -sh
```
Move to a directory:
```
$ cd /home/usr/
```
Move to directory (usr) from within current directory (home)
```
$ cd ./usr/
```
Move back one directory:
```
$ cd ..
```
Remove file:
```
$ rm file_name
```
Remove directory:
```
$ rm -rf directory_name
```
Connect to server using ssh:
```
$ ssh username@host
```
Connect to server using sftp (file transfer)
```
$ sftp username@host
```
sftp has its specific methods, look up before trying it

Ubuntu Package Installation & Management

Install package:
```
$ sudo apt-get install package_name
```
Update repository listings:
```
$ sudo apt-get update
```
Upgrade all packages:
```
$ sudo apt-get upgrade
```
Upgrade packages without altering version-based dependencies
```
$ sudo aptitude safe-upgrade
  [right???]
```

```
Mac OS X Package Installation &
  Management
```

```
Install package:
$ brew install package_name
```
Update repository listings:
```
$ brew update (?????????)
```
Upgrade all packages:
```
$ brew upgrade (?????????)
```
Upgrade packages without altering version-based dependencies
```
$ brew ???????????
```

TOPOLOGY CORRECTION

GRASS GUI with QGIS Plugin

If using GRASS, the User must setup a Location and a Mapset into which the data will be placed. This is unlike other GIS applications in that it requires a sort of local database structure that is controlled within the GRASS environment. One cannot use the system directory (clicking through folders) to access or import data (like ArcMap or QGIS). Please follow the instructions, or find resources online that will benefit the specific setup for your workstation or server. Using QGIS GRASS Plug-in is the best way to accomplish the setup on a workstation. There are multiple was to set this up, here is an example process:

GRASS Setup:

1. If opening GRASS for the first time, make sure the $GISDBASE is setup properly. This is usually just a directory names ‘grassdata’ somewhere in the User directory. I find it’s easiest to set through the GRASS GUI interface (not the GRASS Plug-in in QGIS).
2. Then move to GRASS Plugin in QGIS, select ‘New Mapset’ in the GRASS Plug-in menu. This will allow the creation of a new Location ($LOCATION_NAME) and Mapset ($MAPSET) specific to your data.
3. Make sure the projection information is correct. If you don’t know the extent (usually it’s not important), set the variable ‘set extend from data’ or ‘set projection from data.’ In the case of the GADM dataset, I wanted the entire world extent with WGS 84, so it was a simple set up (HOW?).
4. Once the new Location and Mapset are created, navigate to GRASS Toolset and select ‘Data,’ ‘Vector,’ ‘File Management’ to find in.v.org (or the input command needed for your data type). Then follow the given instruction set for the proper use of topological snapping tools built into the import function. The output function is located in the GRASS Toolset as well.

GRASS Indicators:

Indicator 1: Number of overlaps (GRASS-determined)
Indicator 2: Number of multipolygon features (GRASS and MapShaper (???)-determined)

GRASS Workflow:

1. Start with original dataset in original form, load into GRASS:
-For the example, GADM v. 2 shapefile.

??. Find Indicator 2 (HOW?)

1.1 Load mapset (____/____/____) and open tools. Select v.in.ogr (or appropriate input for data type).
1.2 Make sure under Advanced Options ‘Remove small areas’ is set to 0 [zero] otherwise data will be lost.
1.3 For the baseline, allow no snapping [-1 is the designated threshold].
-For GADM, the baseline takes close to two hours on a standard Mac OS X workstation.

2. Collect Indicator 1 from GRASS output of topological issues.
-GADM produces 92,000 overlaps (overlaps are a good measure of topological characteristics, as ‘gaps’ could be misinterpreted spaces that are true to form. Another good measure is ____.)

3. Rerun the import dataset with a very low threshold for snapping. A suggestion for the threshold is given at the end of the baseline run.
-For GADM, the suggestion was 1E-14. I find that 1E-13 works well with shapefiles in WGS 84. Remember, the threshold value is based on the map units. If using a UTM or Mercator projection, the units are meters, not degrees, like WGS 84. This takes about an hour longer than the baseline due to the snapping.

4. Evaluate the output text from the snapping input process. A good idea, to understand the breadth of the topology issues and the extent to which GRASS altered the data, is to run multiple thresholds and compare the number of corrected features (using our Indicator 1, overlaps).
-GADM with snapping at 1E-13 still contained 52,000 overlaps. 1E-14 contained _____, and 1E-12 contained ____.

5. Choose snapped data. So as not to "overcorrect" or introduce artifacts, we only allow the snapping threshold to correct a portion of the errors. A better explanation is provided in the Methods and the Appendix discussion.
-GADM snapping threshold 1E-13 was chosen for the next correction steps.

6. Move data out of GRASS. Use out.v.ogr to pull the data into a shapefile. The multipart features were split in the GRASS processes. This is the reason the dataset has ballooned in size.
-GADM doubled in size.

————————————————-FINISH ABOVE

7. Now we use a Dissolve function to restructure the data back into multipolygon features.
7.1 Two ways to do this. GRASS provides no proper way of doing this well. (RIGHT?) The attribute created by GRASS, 'cat,' denotes the number of the original multipart feature from which the single feature was pulled. Use this to Dissolve the data.
7.1.1 QGIS allows for a Dissolve, but the process is less than scalable. For a scalable alternative (like one needed for GADM), convert the data and place it into a File Geodatabase and use the Dissolve tool in ArcGIS through ArcCatalog. Then pull the data out into a shapefile.
NOTE: As noted by the MapShaper guys, it appears that the ArcMap 10.1 Dissolve function will exacerbate topological errors already in the dataset.

8. Evaluate the number of features. The dissolved dataset should have the same number of unique features as the original dataset. (INDICATOR 2)**********
NOTE: If there were duplicate features in the original dataset this value might be different.
-For GADM, the numbers are the same (218,238 features)

FIXXXXXXXX - not true, right?
9. PPRepair's output contains no attribution or projection information. This data needs to be pulled from the original data. The data is in its original order, as long as entire multipart features weren't removed as duplicates. This join could be done with a simple field-based join. If the data is particularly different from the original, a spatial join of some sort might be necessary to automate the re-attribution of the polygons.
-For GADM, the data was zipped back together with the OID number.

To finish the topological corrections, we will use a new tool PPRepair with the Dissolved shapefile.

PPRepair Workflow:

https://github.com/tudelft-gist/pprepair

PPRepair is the true shining knight of this process. It won best presentation at OSGIS UK in 2012 because it's just so darn useful. It's also a bit scary due to its completely different approach to the problem

of correcting geometries for use in a GIS. The reason this approach is useful and scary because it's a completely automated, geometric process. It doesn't use typical GIS data model considerations. It takes no account of SRID, units, or topology. This seems strange because we're using GIS data models and fixing topology, but this is the reason PPRepair is successful. It simply looks for the error and makes a decision about how to fix it based on the surrounding shape characteristics. But this is where we must understand the consequences - there is no topological process by which it can check its result, and there aren't metrics that allow for thresholding of alterations (like in GRASS).

Download PPRepair from GitHub at the above link, or using git clone:

```
$ git clone https://github.com/tudelft-gist/pprepair
```

Use a package manager, like apt-get or Homebrew, to install the needed dependencies. It needs GDAL, CDAL, CMake, and build-essential. Most Ubuntu versions come with build-essential tools, but some don't.

NOTE: If you get an error when compiling PPRepair with the CMake command (steps __ - ____), your build of Ubuntu didn't come with the build-essential package, so it's unable to compile the C++ code. The Linux compiler in build-essential is called g++, and is required.

(the -y tag forces a 'yes' to install - remove if you wish to get a prompt):

Ubuntu

```
$ sudo apt-get install build-essential cmake libcgal-dev libgdal-dev -y
```

```
Mac OS X
$ brew install cmake gdal cgal
```

(If GRASS or QGIS is installed on your Mac, be careful with the configuration as Homebrew will install a second version of already installed software, specifically GDAL. This is because Homebrew cannot access the existing installed applications, but it can check if they're there. It will prompt the Terminal with a caution message if there are two installed (one in the /tmp/___ Terminal-based package storage, and one in the User directory. A second install of PostgreSQL seems to be okay, while a second version of GDAL breaks most functionality.)

Once downloaded and the dependencies installed, navigate to the ./pprepair-master directory. Here you will 'install' an instance of PPRepair to use on your data. Each time you use the tool, make sure to follow these steps as it must be 're-installed' for each use. [[[There is probably a better way of doing this]]]

```
$ cmake .
$ make
$ ./pprepair -i inputfile.shp -o outputfile.shp -fix
```

Example (from Mac OS X command line):

```
$ cd ~/Desktop/arc_maps/pprepair_master
$ cmake .
$ make
$ ./pprepair -i ~/Desktop/map_data/GADM.shp -o ~/Desktop/map_data/
  GADM_output.shp -fix
```

NOTE: PPRepair creates an output shapefile, no need to provide anything other than the directory and a name with .shp extension (as shown above).

-PPRepair with GADM consumed about 56 GB of RAM on an Amazon EC2 server. Smaller datasets are much quicker. See Topology Checking (page ____) for an example.

NOTE: PPRepair is brute-force, and has no consideration for the data other than the geometric issues it's made to fix. The tool uses non-geospatial libraries to correct geometry: GDAL is used only for shapefile compatibility. This is slightly risky as the data is not able to be evaluated before or during the process with the usual geospatial considerations. Just an FYI. The tool is extraordinarily useful, but I have experienced errors in even small datasets – I believe it was a result of GRASS "overcorrection," but further investigation in the cause of errors is needed.

11. Now we have a complete dataset that should be topologically corrected. At this point it is a good idea to run it either into GRASS to measure overlaps (make sure there are none) or use Topology Checker in QGIS or a similar tool. GRASS is fast, but it will also alter the data again. Only use GRASS as a way to verify our measure of topology (overlaps), so do not output this data from GRASS. It also helps to use "check invalid geometries" in the Topology Checker in QGIS. Both measures (overlaps and invalid geometries) should return zero. If not, a different threshold or some other topology correction may be needed.

Now continue to **Encoding**, **Database Management**, and/or **Simplification** for continued development of data management tools and methods.

ENCODING

Encoding is simple yet difficult due to the relaxed nature of the documentation and the gaps in tools that properly translate between encodings. Here we look at encoding in GADM v. 2 and properly convert from the given encoding type to our database type (UTF-8).

The GADM v. 2 encoding seems to be well described by "Latin1," though another actual encoding type could be the correct original.

Here we assume the encoding for GADM v. 2 is Latin1, so the procedure to convert is simple.

1. Open QGIS, select "Import Vector Layer"
2. Select encoding of input vector file - for GADM it's Latin1
3. Load vector file.
4. Right-click on vector layer, select "Save as.."
5. Select shapefile for output, and the select new encoding in pull-down menu (eg. UTF-8)
6. Save file. Encoding is now converted!

DATABASE MANAGEMENT

This section covers ___these___ aspects of PostGIS database management

Useful commands for PostgreSQL:

Open a PostgreSQL database via the terminal command front-end tool psql:

```
$ psql -d database_name
```

Once the database is open, the prompt will change from $ to #
For example, from Ubuntu to PostgreSQL:

```
sam@ubuntu:~$  psql -d database_name
```

to

```
database_name=#
```

The main differences with commanding/using psql for the front-end to the database:

1. To execute an SQL query, it must end in a semi-colon ;

```
# SELECT * FROM table_name WHERE column_name = value;
```

2. To command the database, each command is preceded by a backslash \
   List databases:

```
# \list
```

   List tables in current database:

```
 # \dt
```

   Quit:

```
# \q
```

SIMPLIFYING - see associated scripts/code for all GADM-related processes

These instructions assume familiarity with terminal commands using Ubuntu 13.01 (saucy). The tools can be easily interpreted for use with Mac OS. For example, use Homebrew as a package manager for installation of the components.

Examples:
Install Node.js with Homebrew:
$ brew install nodejs
Note:
—don't try to restart PostgreSQL in Mac, it's a lot different from Linux commands

PostGIS Topology
http://postgis.net/docs/Topology.html

For a database simplification solution, use PostGIS Topology. Please be aware that creating the topogeom is intensive and slow for medium to large datasets. If the input is large or using the database is unnecessary, the other option, MapShaper, is recommended.

1. Enable topology in database (this assumes your data is loaded already and spatially enabled).

CREATE EXTENSION postgis_topology;

NOTE: If the database already has loaded tables, reloading the database is required - the new topology extension doesn't appear unless new data is added to the database, or it's reloaded.

If exiting psql/PGAdmin and reopening doesn't work, try restarting the database server (if possible):

$ service postgresql restart

If reloading and restarting the database isn't possible, input a table (some dummy data) so the extension is forced to load upon addition of a table.

2. If needed, project the geometries and/or change geometry type (both required for GADM):

For reference:
gadm_level refers to the existing tables, for GADM there are six (0-5).
gadm_level_topo is the topology schema, one is needed for each table, even though they all have the same SRID - the reason for this apparent when we run simplification.

SELECT UpdateGeometrySRID('public', 'gadm_level', 'geom', 3857);

ALTER TABLE gadm_level ALTER COLUMN geom TYPE geometry(MultiPolygon, 3857) USING ST_Multi(geom);

3**.** Create topology schema:

SELECT CreateTopology('gadm_level_topo', 3857);

4**.** Add topogeom column (for topogeom) and new geom column (for simplified output):

SELECT AddGeometryColumn('gadm_level', 'geom_simp', 3857, 'MULTIPOLYGON', 2);

SELECT AddTopoGeometryColumn('gadm_level_topo', 'public', 'gadm_level', 'topogeom', 'MULTIPOLYGON');

5. Create topogeom from geom:
[WARNING: VERY TIME CONSUMING - INCREASING RAM DOESN'T HELP]

UPDATE gadm_level SET topogeom = toTopoGeom(geom, 'gadm_level_topo', 1);

6. Load custom simplify script into database:
http://strk.keybit.net/blog/2012/04/13/simplifying-a-map-layer-using-postgis-topology/

```
# CREATE OR REPLACE FUNCTION SimplifyEdgeGeom(atopo varchar, anedge int,
maxtolerance float8)
RETURNS float8 AS $$
DECLARE
  tol float8;
  sql varchar;
BEGIN
  tol := maxtolerance;
```

```
  LOOP
    sql := 'SELECT topology.ST_ChangeEdgeGeom(' || quote_literal(atopo) || ',
' || anedge
      || ', ST_Simplify(geom, ' || tol || ')) FROM '
      || quote_ident(atopo) || '.edge WHERE edge_id = ' || anedge;
    BEGIN
      RAISE DEBUG 'Running %', sql;
      EXECUTE sql;
      RETURN tol;
    EXCEPTION
     WHEN OTHERS THEN
      RAISE WARNING 'Simplification of edge % with tolerance % failed: %',
anedge, tol, SQLERRM;
      tol := round( (tol/2.0) * 1e8 ) / 1e8; -- round to get to zero quicker
      IF tol = 0 THEN RAISE EXCEPTION '%', SQLERRM; END IF;
    END;
  END LOOP;
END
$$ LANGUAGE 'plpgsql' STABLE STRICT;
```

7. Simplify through the topology schema:

> Note:
> This simplifies all topogeoms with this schema, which is why we created different schemas for each topogeom - in case different levels of simplification are required, or if it doesn't work it would corrupt the other topogeom configurations. I think. The topology setup is not as forgiving as other database actions, so if something goes wrong sometimes starting from step one is the only remedy.

```
# SELECT SimplifyEdgeGeom('gadm_year_topo', edge_id, .01) FROM
  gadm_year_topo.edge;
```

8. Copy simplified topogeom to new simplified geom column as a geom:

```
# UPDATE gadm_level SET geomsimp = topogeom::geometry;
```

9. Convert geometry types back to the required type (don't forget there are two (2) geometries):

```
# ALTER TABLE gadm_level
ALTER COLUMN geom TYPE geometry(Geometry, 3857) USING geom;

# ALTER TABLE gadm_level
ALTER COLUMN geomsimp TYPE geometry(Geometry, 3857) USING geomsimp;
```

10. Project back into the required projection:

```
# SELECT UpdateGeometrySRID('public', 'gadm_level', 'geom', 4326);
```

MapShaper Simplification
https://github.com/mbloch/mapshaper

If the geometries you're simplifying are in PostGIS, first pull them out into shapefiles in step 1. If shapefiles are already available, skip to step 2.

1. Export shapefiles from database:

```
$ pgsql2shp -f gadm_level.shp -h localhost -p 5432 gadm_database
  public.gadm_level
```

2**.** Install Node.js and npm:

$ sudo apt-get install nodejs

If the following error occurs:

"$ /usr/bin/env: node: No such file or directory"

Follow this fix - create a symlink to 'node' location:
(be aware it redirects the system when node is called - if another package named 'node' exists elsewhere, this fix will make it inaccessible to the system. I don't know of another package with the name 'node,' but it's important to understand.)

```
$ sudo ln -s /usr/bin/node /usr/bin/env/node
```

3. Then use the Node.js package installer npm to install MapShaper. Works much the same way as apt-get or Homebrew. The -g tag makes the scripts executable systemwide.

```
$ npm install -g mapshaper
```

4. For the sake of simplicity all possible commands aren't cover here; the following methods were used to implement the simplification with GADM:

```
$ mapshaper --keep-shapes -p 0.5 gadm_level.shp -o simp_gadm_level.shp
```

For certain levels of GADM a -p (percent retained) value of 0.6 worked better. The program automatically corrects intersections that result from simplification, but sometimes they can't be automatically fixed. It prints the number of intersections fixed and not so that the analyst can judge the proper threshold. Example result message:

"Repaired 21 intersections; unable to repair 1 intersection."

The full command reference is available at:
https://github.com/mbloch/mapshaper/wiki/Command-Reference

Now simplification is complete. If the simplified geometries need to be added to a database in PostGIS, continue with the tutorial.

Inserting Simplified Geometries into Existing Database Tables

Here we will add the simplified geometries to the original tables - there will be two geometry columns for each entry. This way the attribute data doesn't need to be duplicated, only an option for simplified or original geometry, depending on which geometry column you call in your query.

The simplified geometry tables and original geometry tables should be side-by-side in a database. See Figure ___ for illustration.

1. Input shapfiles to database as new tables - alongside the tables you wish to add the simplified geometries:

```
$ shp2pgsql -I -s 4326 simp_gadm_level.shp simp_gadm_level_table | psql -d
  gadm_database
```

2. Convert geometry type if necessary (type should match the target table):

```
# ALTER TABLE gadm_level
ALTER COLUMN geom TYPE geometry(Geometry, 4326) USING geom;
```

3. Create new geometry columns:

```
# SELECT AddGeometryColumn('gadm_level', 'geomsimp', 4326, 'GEOMETRY', 2);
```

4. Insert new geometries with UPDATE:

```
# UPDATE gadm_level
SET geomsimp = simp_gadm_level.geom
FROM simp_gadm_level
WHERE gadm_level.id_# = simp_gadm_level.id_#;
```

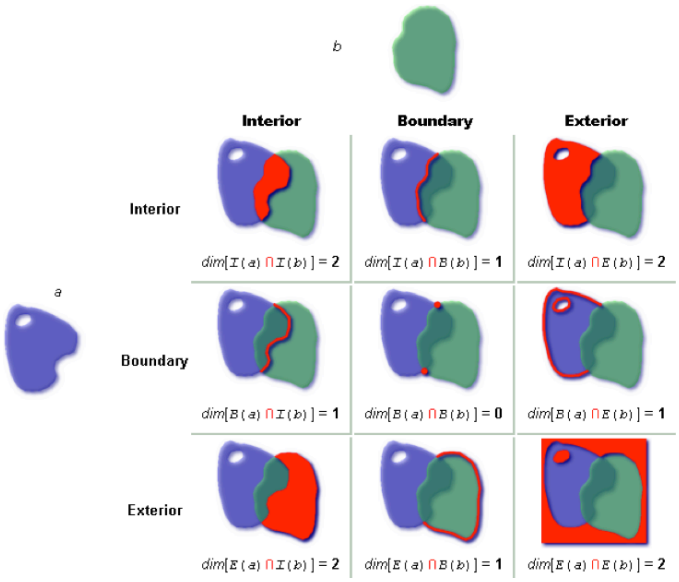
Note:
If using dataset like GADM, make sure to extend the WHERE clause to include restrictions on the update. For example, for GAMD level 5:

```
UPDATE gadm_5
SET geomsimp = simp_gadm_5.geom
FROM simp_gadm_5
WHERE gadm_5.id_0 = simp_gadm_5.id_0 AND gadm_5.id_1 = simp_gadm_5.id_1
     AND gadm_5.id_2 = simp_gadm_5.id_2 AND gadm_5.id_3 = simp_gadm_5.id_3
     AND gadm_5.id_4 = simp_gadm_5.id_4 AND gadm_5.id_5 = simp_gadm_5.id_5;
```

Appendix

Topological correction perspective for the GIS analyst

Brief review of topology for GIS:
DE-9IM standard defines the relationships we need to describe for geometries in GIS to be topologically valid. A much more in-depth review is available on Wikipedia. A brief discussion follows.
The standard describes the nine constructs by which we define 2-d spatial topological relationships. A graphic from Wikipedia helps describe them:



This chart can be represented by a matrix that describes the possible relationships:

$$\begin{bmatrix} II & IB & IE \\ BI & BB & BE \\ EI & EB & EE \end{bmatrix}$$

Source: _____

The relationships between these different types describes a total of 512 possible relationships (too many, as some of them are useless to us), while we have descriptions for about ten of the most useful ones (Equals, Contains, Covers, CoveredBy, Crosses, Disjoint, Intersects, Overlaps, Touches, and Within). For example, if we want to know if a feature EQUALS another, we have to ask if [II ~IE ~EI ~BE ~EB] are all in agreement, with ~ meaning "not equal." So, the interior interiors are the same (II), the external interiors aren't the same (EI and IE), and the external boundaries aren't the same (BE and EB).

Topology and GIS software

Topological constructs in GIS software aren't made equal. Usually we talk about topology as the ability to measure adjacency, containment, and connectivity. For polygons, we only must consider adjacency and containment, leaving connectivity for network and other, more intricate topological models. When the topology is incorrect, such as an overlap where an adjacency should be or an improperly coded island (the contained topology type) different topological models in different software will react improperly or fail to function. So the topology must be corrected. The challenge, at least when using Open Source and User-oriented solutions, is maintaining a basic understanding of what's happening with each process and how the different data models, as we move from one solution to the next, will properly (and improperly) represent the topological characteristics. [Black box discussion_____]

TL;DR: different software, different topological model.

This becomes an issue because GRASS uses a topology-based data model, unlike the typical shapefile data model (pure geometry and attribution). GRASS is useful in this way because it gives us measures of

topology faster than other solutions, and it tries to automatically correct any errors it encounters. The topological model consideration: with a GRASS output, issues arise specifically from an invalid geometry type “self-intersecting lines.” GRASS allows these geometry types, per its topological data model, while all other software and data types do not.

This is where PPRepair becomes very handy. If given a dataset with nothing but self-intersecting line errors, it corrects the errors without altering the data. It also maintains the multipart polygon feature that the feature was originally structured as; the feature with the error is split into separate features, but remains one, multipart feature. If PPRepair is given a dataset with errors other than self-intersecting lines, it will attempt to correct them as well. This is done with geometric triangulation, and will fill spaces and clip overlaps automatically. This is useful to correct a dataset that has minimal, or lessened topological errors.

*If used in conjunction with GRASS, as proposed here, the snapping processes in GRASS can ‘overcorrect’ and produce artifacts that PPRepair will try to fix. This can cause LARGE errors. This is why the snapping topological correction in GRASS is scaled down such that the smallest amount of errors are fixed. This seems to be the goldilocks level of correction – first GRASS, then PPRepair seems to produce less/no errors compared to using one or the other.

Other Useful Resources

Geometric Relationships (Topology relationships in PostGIS; not part of topology schema organization):
http://postgis.refractions.net/documentation/manual-1.3/ch06.html#id2574517

Acronyms & Names

GeoDB
ARC
GIS
GRASS
QGIS
PPRepair

Experimental Support Documentation for TopoJSON

TopoJSON supported software configuration:

WARNING: This is “bleeding edge” so is difficult to implement unless an experienced user. Please note our shop did not implement these methods.

(still need to get correct configuration running: ——————-)

use dpkg for manually install packages:
sudo dpkg -r PACKAGE_NAME
PostGIS 2.2
PostgreSQL Version 9.3
Geos Version 3.4

GDAL Version 1.11

Once dpkg is figured out….
install missing dependencies:
sudo apt-get install -f
to reconfigure/repair dpkg install
sudo dpkg-reconfigure packagename

GLOSSARY

Defintions:

Dependence
Solution
Technique
Method
Program
Application
Format
Data Model
Feature

Sources

http://postgis.net/docs/Topology.html

http://strk.keybit.net/blog/2012/04/13/simplifying-a-map-layer-using-postgis-topology/

https://github.com/mbloch/mapshaper

Arroyo Ohori, Ken, Ledoux, Hugo and Meijers, Martijn (2012). Validation and automatic repair of planar partitions using a constrained triangulation. *Photogrammetrie, Fernerkundung, Geoinformation (PFG)*. 5:613–630.

http://en.wikipedia.org/wiki/DE-9IM