Alex Kara
Advanced GIS
March, 2015

Arqueologos: First Steps Towards Condensing Archeological Data Collection, Management, and Analysis to 21st Century Standards

Archaeology is the study of material remains of humans. While historical data, meaning information textually and *intentionally* recorded by humans, offers very detailed accounts of the past, those studied by archaeologists are the only objective and quantitative evidence for analyzing human behavior through time. Archaeological data also represent an unparalleled breadth of social phenomena in regard to both space and time. Given such a nature, one would expect this valuable resource to be recorded with advanced equipment, managed in a robust DBMS, made accessible to global researchers, analyzed with numerical methods, and ultimately harnessed to render the Earth's biosphere a sustainable, thriving, indestructible machine. But it is not so. Archaeology, unlike all major branches of natural science and certain parts of social science, is not characterized by the above to a any respectable degree. Measurements are generally analog, recorded by instruments such as tape measures, line levels, and strings, while this primary data is recorded in paper format and kept private until published in a monograph, if at all. Visual representation is limited to two-dimensional illustrations. Numerical analysis, despite having some history in the discipline, is rare. This paper presents an embryonic manifestation of Arqueologos, a system designed to reverse this trend. Commendable efforts have applied technology to improve certain parts of the archaeological processes; Arqueologos goes further by unifying data collection, management, and analysis into a lean, singular process. It strives to be approachable to learn and practical to adopt, and all of its components are free and open-source, in theory.

General Description

Today it is an uncontroversial notion that computer technology has transformed the practice of archaeology. Advancements in computer hardware and subsequent ones in software have allowed archaeologists to record, manage, and analyze the material remains of past human activity with enhanced speed and accuracy. Archaeologists have especially leveraged the power of GIS, DBMS, and 3D graphics in their endeavor to understand humanity's past (Conolly and Lake 2006). Arqueologos was designed to combine these three types of software.

It is not straightforward to define what exactly Arqueologos is at this point, never mind what it wants to or will become. It is best described as a series of existing computer software that have been modified to automate their interoperation (Figure 1). This automated workflow is designed for archaeologists, but it might also provide utility to geologists and ecologists that collect subsurface samples in a similar fashion. The two most crucial components to Arqueologos are PostGIS and QGIS, which essentially manages the data in a server and mediates human interaction with the software, respectively. PostGIS is a relational DBMS designed for GIS information. It ensures the long-term integrity of an archaeological project's data, and allows allows sufficiently savvy user to perform powerful queries and transformations with the SQL language. The QGIS plugin allows a user to upload, extract, and analyze the data in PostGIS through a GUI . In this sense, QGIS immediately serves more as a convenient environment for the GUI than for traditional desktop GIS functionality, but said functionality is of course available and its optional use is encouraged.

It is much easier to define what Arqueologos actually does. Currently, it performs two technically distinct but certainly related functions. With the first function, “upload and process data” (UPD), a user can create 3d, georeferenced, polyhedral geometry that contains information about the archaeological material contained within its extent. One starts by providing two files:

the coordinates of excavation corners and a table containing what workers collected from each layer of excavation in addition to the depth of each layer's corners. Arqueologos processes this information, and for each layer it creates a row in the PostGIS table that contains both a 3d representation of its shape and the counts of different types of artifacts that were found there. The second function allows a user to interact with a 3d visualization of these same excavation layers. A user chooses some sub-sample of excavations to render along with what type of artifact they are interested in mapping. A window pops up that shows the selected excavation units in 3d. Individual layers are colored according to the density of the selected artifact that they once contained, and this model can be rotated and zoomed to the user's liking. These capabilities are admittedly limited, but they also reflect a careful prioritization over other potential features that could not be included due to the time restrictions (albeit leniently...) imposed on the author.

The PostGIS Database

PostGIS is not an independent application but is instead a 3d party extension of PostgreSQL (PSQL). PSQL is one of the leading RDBMS's and, like QGIS and PostGIS, is open source. For this project, PSQL's non-existent price-tag was its most lucrative feature over other RDBMS's. An RDBMS is superior to paper documents for storing data because it allows those data to be shared over the internet and processed with computer applications, but the advantages of RDBMS over simple file directories is far less known to the archaeological community. Firstly, it allows constrains to be placed on new entries to any given column. This avoids, or at least detects earlier, problems such as an accidentally entering space character in a column of numeric values or entering a ludicrously high or low value via a misplaced decimal. Secondly, when users update data remotely on a secure server hosting their project's RDBMS, those updates are instantly available to the other project members, which avoids inefficient emailing and the risk of confusing file versions. Of course, the same affect can be achieved by storing files in cloud storage, but this does not address the previous or following points. Finally, RDBM's like PSQL are beautifully interoperable with coded scripts and existing software. Data can be added, updated, transformed, queried, and to a degree even analyzed using a consistent syntax. As Arqueologos adds more features in the future, the versatility will render every step of development more efficient.

PostGIS extends PSQL by adding spatial functionality. In any RDBMS, every feature in a table shares specifications for the one or more attributes that describe them. The most important type of specification is an attribute's data type, which can be a number, a piece of text, a date, or raw binary code, among other options. PostGIS adds GIS data types to those naturally supported by PSQL: lines, strings, polygons, and a few less common ones. These values are associated with a geographic or projected coordinate system, as expected to be the case in any GIS software. PostGIS also adds a number of GIS functions to manipulate and convert spatial data using SQL. The most popular desktop GIS software, namely QGIS and ArcGIS Desktop, can connect to these spatial databases in order to manipulate and analyze these data in a more user-friendly environment.

DBMS's, both spatial and aspatial, have relatively long been used in archaeology (Kwamme 1989). Because operating a DBMS is most advantageous when handling large amounts of data, individual settlements are rarely the focus of such a system because only a small number will exist within an archaeological project's extent. However, they do generate returns when adopted by governments (Morcanu and Velicanu 2011, Buckland 2010, “Database”). Smaller projects have successfully used spatial DBMS's to represent the distributions of smaller features and artifacts across a landscape (van Haaster and Brinkkemper

1995, Sauerbiera et al. 2008, Contreras and Melero 2010, Deufemia et al. 2012). In each of these cases, the observed data were modeled as point vectors, due to the object's minuscule size relative to the study area. This is approach is therefore well suited for mapping distributions of archaeological sites across a region or individual artifacts across a site. One might expect, then, that for representing relatively large and/or importantly shaped objects, archaeologists would have designed a spatial DBMS incorporating more complex geometry types: lines, polygons, and polyhedra. These types of abstractions would better serve situations where objects are large relative to their study area. Examples include ancient roads or soil types within an a region and ruined structures, or, as this project presents, excavation levels within an archaeological site. Such applications of spatial DBMS, however, have not been realized. This results from the lack of analytic return a researcher has traditionally been able to garner from non-point vector formats. Lines, polygons, and polyhedrons are much less straightforward to study with spatial statistics, and such point patterns in many cases are easier to understand qualitatively as well. Therefore, archaeologists using spatial DBMS have had little incentive to invest in a more detailed type of geometry. The choice to adopt polyhedral geometry for Arqueologos is justified by the 3d visualization aspect, described later.

Arqueologos was inspired by recent updates to PostGIS that added support for advanced 3d functionality. The third dimension is not a new one to GIS software; DEM's have been around for just as long as, if not longer than, the concept of GIS itself, and most of today's desktop GIS software allows user's to visualize 3d surfaces vis-à-vis vector data that contain a “z” attribute. However, PostGIS is one of two spatial data infrastructures (the other being Oracle Spatial) to support true polyhedral geometry. The popular ESRI shapefile and geodatabase formats still do not support this type of geometry. The importance of this advance to archaeological GIS cannot be overstated, and its full implications are not presently reached by this project or any other. Although Arqueologos does not showcase the flexibility of polyhedral geometry, the author chose to incorporate these data structures into Arqueologos so that future versions of the software can model large archaeological features having complex 3d morphology.

Visualization

In addition to providing tools for archaeological data management, Arqueologos also provides those for analysis of such data. The scope of this project restricted how much could be included at this point in time. The author chose to prioritize the 3d visualization tool over all other potential inclusions as it is theoretically uncontroversial among the archaeological community. If I do continue to develop this, a full suite of statistical tools will eventually be included. In fact, Arqueologos is specifically built on top of robust open-source GIS and DBMS software in order to encourage future extension in this regard by its developers or even unaffiliated users.

In addition to merely granting practitioners a more efficient workflow, software such as GIS, DBMS, and 3d graphics categorically alter how archaeologists collect and interpret their data. In doing so, digital technologies influence the development of archaeological theory and vice versa (Zubrow 2006). Unfortunately, any mutualistic relationship between the two has only truly taken place at one of two spatial scales: that of either an entire region or a small area of excavation. Desktop GIS software allows a researcher to escape the confines of their terrestrial niche and analyze diverse geographic data at much broader spatial scales. The software can display archaeological data alongside an elevation raster, hydrological model, or other any other data regarding an ancient society's environment. This feature of GIS has spurred archaeologists to investigate distributions of sites, features, and artifacts across a landscape and how these patterns reflect human interactions with their environment and/or their conspecific neighbors. Desktop GIS software features powerful 2d visualizations that researchers have utilized in their qualitative analyses of regional settlement dynamics (Barker et al. 2006, Garrison and Dunning 2009). Certain GIS-specific algorithms, such as viewshed and least cost path analyses, are useful tools for archaeologists regardless of their theoretical approach (van Leusen 1999, Estrada-Belli and Koch 2007). GIS also facilitates the use of spatial statistics and other rigid quantitative analyses. These can be extremely useful for interpreting spatial patterning in the archaeological record or the lack thereof (Hodder and Orton 1976, Lloyd and Atkinson 2004, Bevan et al. 2013, Vandam et al. 2013).

Computers have also improved archaeological practice at smaller scales of analysis. More advanced field equipment and new software expedite excavation while recording data in a more accurate and disseminable fashion. Total stations were traditionally used for land surveying, but archaeologists have resourcefully applied them to recording the 3d coordinates of excavated materiel with incredible precision (Barcelo et al. 2003, Smith and Levy 2012). 3d scanners are also being used to record precise excavation data but at a much faster rate than allowed by total stations (Tokovinine 2013, Forte 2014). Ground-based photogrammetry has the similar capacity to produce detailed 3d models of excavation units but without necessitating expensive equipment (Orengo 2013, De Reu et al. 2013, Prins et al. 2014). Archaeological projects unearth thousands of objects significant to their aims, and relational databases enhance a project's ability to smoothly manage an otherwise unwieldy amount of information (Voorips 1998, Adams 2003, Lock 2003, Gugnali et al. 2011). Researchers have gone beyond merely recording the shapes and locations of objects per se; they have introduced broader frameworks for excavation that allows information from their total station, laser scanner, or photogrammetrically purposed camera to transfer information more or less seamlessly to their GIS or DBMS software (Orengo 2013, Prins et al. 2014, Forte 2014).

It is clear that computer technology has revolutionized how archaeologists study geographic distributions of material remains within each of these two scales: that of the region and that of the excavation. Such a dichotomy can never perfectly divide each project's use of technology in relation to spatial scale: a term that is sufficiently ambiguous in its own right (Quattrochi and Goodchild 1997, Harris 2006). Even considering this shortcoming, there is still reason to perceive that archaeologists have used digital technology to advance their theory within but not beyond some ideal spatial resolution. Archaeologists must strive to develop methods that operate across traditional spatial scales by using every computational tool at their disposal, including GIS, DBMS, and 3D graphics (Harris 2006). This is not to say neither that archaeologists do not look for multi-scale processes in past human societies nor that current data collection practices impede such an endeavor; the opposite is true (Carballo 2012, Erdal 2012). At the same time, archaeologists that connect phenomena operating at distinct spatial scales rarely do so as a direct result of new software but instead through their own analytical ability as a professional scholar. This existing, qualitative ability can be improved with new and appropriate technology.

The visualization aspect of Arqueologos strives to fill this void as it was explicitly designed to foster multi-scale analysis in archaeology. At present it focuses on erasing the boundary between excavation and region-scale research, which it does by rendering a symbolic representation of excavation data vis-à-vis a model of a site's topography and surface features, although the second part isn't quite ready yet. The visual simplicity and resulting power of this system, that is currently under development, is achieved by sacrificing many fine details bestowed by total stations, 3d scanners, and ground-based photogrammetry. This sacrifice does not involve deleting valuable data but instead temporally aggregating it in a way that facilitates visual digestion. Arqueologos provides a 3d graphics viewer that allows a user to explore deep, complex archaeological data in a flexible environment, allowing one to crucially develop a more human understanding of the material record than is accessible in a traditional desktop GIS program (Tilley 1994, Gaffney et al. 1996). Acevedo et al. (2001) presented a similar system that allowed researchers to visualize architecture and excavation data in a 3d virtual reality environment. They found that their system significantly improved a user's grasp of the complex primary data at hand. Arqueologos builds on the essence of this work using a modern suite of software componentry. Furthermore, the lenient requirements for the program, in terms of data, ensure that even a poorly funded and/or inexperienced project can benefit from its application.

Detailed Description

Arqueologos is not ready for distribution. Here, its current functionality is described using an imaginary excavation at the imaginary site of Beihr Kaaht. Although this site does not exist, the nomenclature adopted for this version of Arqueologos vaguely reflects that of the Motul de San Jose Periphery Project, where the author currently works as a GIS research assistant. (Hypothetically,) Beihr Kaaht was a small, bronze age settlement until is was destroyed by a volcanic eruption. The burning hot ash from this eruption burned the wood-framed homes of the Beihr Kaahtians where they stood, concealing any obvious remains of any domestic features. After completing one year of intensive surface survey, the horribly underfunded Beihr Kaaht Archaeological Project (BKAP) commenced a new phase of test-pit excavations in areas thought to contain buried remains of ancient settlement. All of these units failed to find anything interesting – save for within a subset that was classified as operation (op) 1. There are four “test-pit” excavations in op 1, which are further and equally divided into sub-operations (sub-op's) 1 and 2.

Figure 2, Figure 3, and Table 1 show how data from Op 1 would be visualized using traditional methods of drawing in archaeology. The plan view (Figure 2) offers an aerial view of each excavation unit's 2d extent parallel to the Earth's surface. The profile view (Figure 3) shows each excavation layer's 2d extent perpendicular to the Earth's surface. Finally, the excavation data table (table 1) lists the attributes of each layer, representing what was collected there. To know what artifacts were collected in a specific layer, one must refer to the data table entry associated with a layer in the profile drawing, and then, to contextualize this amount within the larger space of the site, one must relate this to the plan drawing. It is clear how this system impedes qualitative interpretation of even the most simple archaeological operations. To this day, plan drawings, profile drawings, and data tables are the primary means for disseminating archaeological data.

The first step to using Arqueologos, as it currently works, is to upload one's excavation data to the PostGIS database. This is done via a GUI interface in the QGIS desktop GIS software (Figure 4). First, a user must install Postgresql and properly create a PostGIS database before using Arqueologos. Next, one runs the “Upload/Process Data” function of the Arqeuologos plugin, which presents a GUI prompt to the user (Figure 4). In the leftmost panel are mandatory input fields for the PostGIS database connection information. The prompt also requires a user to upload two files: an ESRI shapefile (.shp) and and a comma separated values (.csv) file (Table 2). The .csv is essentially equivalent to the data table presented in excavation reports, except that it requires the depth of each layer's four corners to be included as attributes (which is indeed included in many data tables regardless). In this file, each “object”, or row in the table, represents a single layer in an excavation unit. The .shp is a point-type shapefile representing each excavation's four corners, with each object being the location of one corner. The “corner” attribute of each of the corner points indicates that corner's location relative to the excavation unit it defines. Corner 1 is the northwest corner, 2 is northeast, 3 is southeast, and 4 is southwest.

Each corner object has a many-to-many relation with level objects. The two types of objects can be joined according to shared combinations of op, sub-op, and unit designations. Arqueologos then runs an algorithm to create 3d polyhedral geometries using the associated corner locations from the .shp and corner depth values from the .csv. Each excavation unit's top layer's geometry has the locations of its four upper corners taken directly from the associated points in the excavation corners .shp. Next, this top layer has the locations of its lower four corners calculated as the location of the respective, overlying upper corner minus the respective “depth” value from the .csv. At this point there are eight corners for the active layer: four upper and four lower. These eight points define the extent of the newly created polyhedron geometry that represents the 3d area of an excavation layer. The algorithm then calculates polyhedra for the remaining lower layers in an excavation unit in the same fashion, calculating each layer's top four corners as the bottom four corners of its overlying layer, and then the algorithm proceeds do the same for the other excavation units. Finally, Arqueologos assigns the completed polyhedra their corresponding attributes from the layers .csv file. Once this uploading process is complete, the user's PostGIS database will contain a table of every excavation layer that the user uploaded, their attributes about archaeological findings, and their 3d polyhedral geometry. This data then can of course be accessed by anyone that can connect to the PostGIS database over a network. It can be loaded into QGIS (and ArcGIS as well) directly from this database, where it will appear as a 2d polygon vector (Figure 5).

Of course, this 2d perspective does not leverage all of the hard work Arqueologos has done to create the 3d geometry for each excavation layer. The next step in using Arqueologos is to use QGIS to visualize the excavation data in 3d. The first step in the visualization process is to select a color scheme for the layer using QGIS's built in style editor (Figure 6). This should be a familiar process to anyone that has experience in QGIS or ArcGIS. The user selects an attribute of interest, a classification scheme, and a color ramp to represent their layer in the active canvas. To first demonstrate the software's accuracy after all of the crazy calculation describes above, I first just chose to color it according to the level number of each layer. This variable is 1 for the highest layer in an excavation unit, 2 for the layer below 1, etc. Then, select and highlight the excavations layer in the QGIS layer panel. The next step is to select the “3d Excavation Viewer” tool from the QGIS plugins menu. It is below the “Upload/Process Data” tool used earlier. This tool displays a prompt where the user again enters the PostGIS excavation data and can also query what excavations they want to visualize (Figure 7). When the user clicks 'OK' Arqueologos displays the OpenGL 3d scene rendered using Python through the Vispy library. The scene does not automatically adjust to the loaded data, however, so at this time the user needs to manually zoom out using the mouse wheel. Once zoomed out, they will see their excavation layers rendered in 3d and colored according to their level number (Figure 8). They can use the mouse to rotate the scene and view their excavations from a different angle. Users cannot pan because when I wrote this Vispy did not allow this feature when I coded this software. Note that the shade of blue in Figure 8 correctly represents each layer's relative depth like it is supposed to.

Arqueologos is meant to visualize more interesting variables than layer depth, however. Figures 9 and 10 show the results of visualizing Beihr Kaht Op 1 according to the amount of lithic and ceramic artifacts collected from each layer, respectively. Much unlike the traditional way of visualizing excavations, one can now see interesting patterns in the data. For example, the lithic data shows that, early on, there was a higher concentration of lithic artifacts at the bottom of the hill than at the top. During this period, the Beir Kahtians probably farmed this hill, cutting trees and crops with lithic tools. These tools would flake over time, and these flakes were deposited at the bottom of the hill from erosion. However, in later periods there is more lithics at the top of the hill than at the bottom. Possibly the Beir Kahtians were producing some stools at the top of the hill, which explains the lithic artifacts found there, but had stopped farming the hill at this point. These hypotheses would be more difficult to reach without the aid of the 3d visualization tool of Arqueologos.

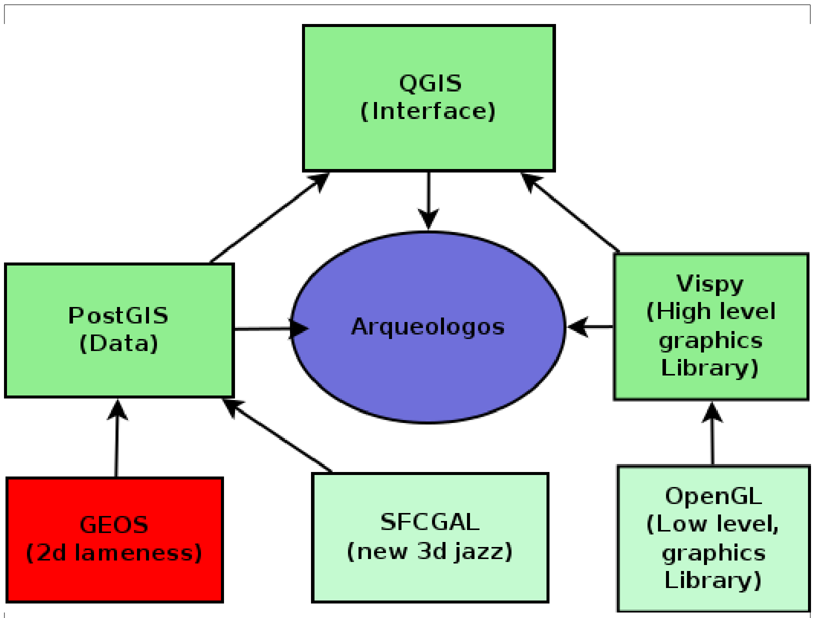


Figure 1: Diagram of Arqueologos dependencies

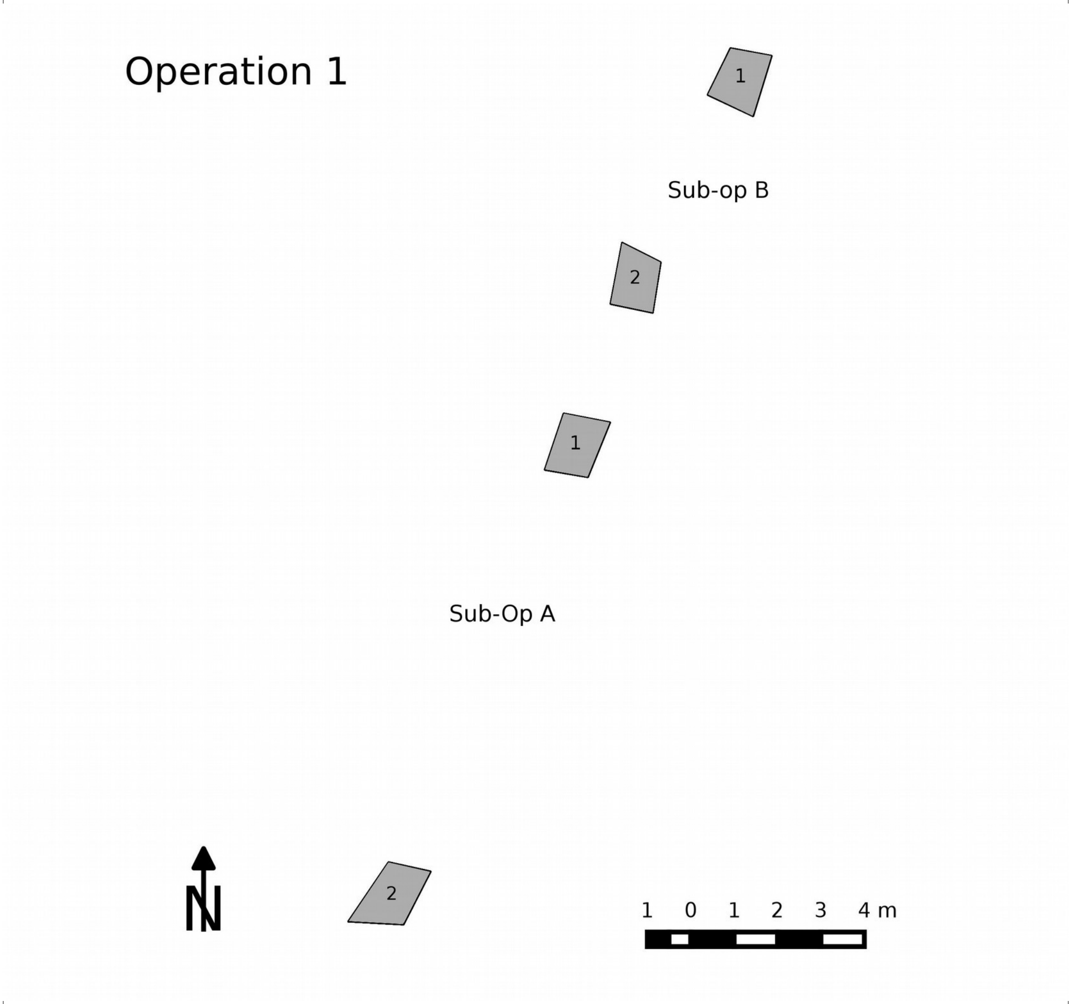


Figure 2: Plan view of Behr Kaht operation 1

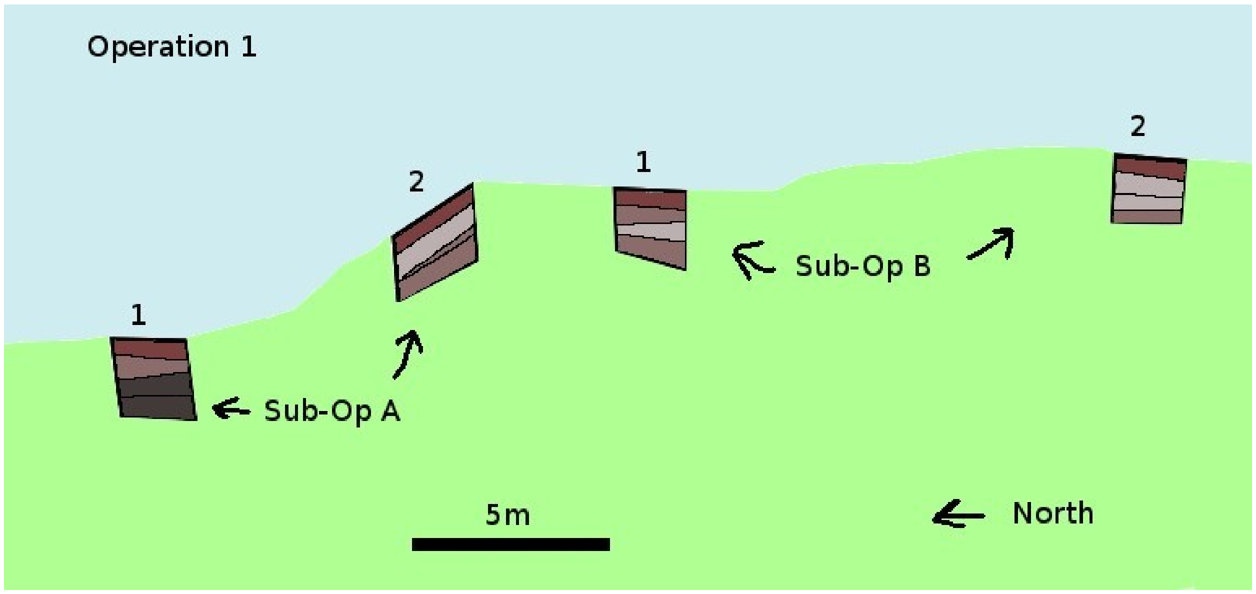


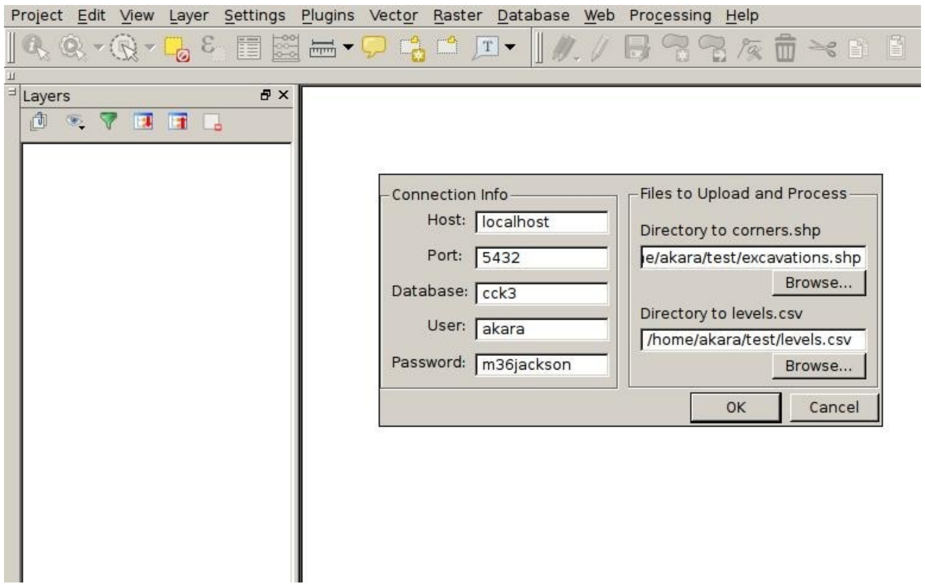
Figure 3: Plan view of Behr Kaht operation 1

op	subop	unit	level	lithic	ceramic	age	depth1	depth2	depth3	depth4
1	a	1	1	0	0	1	0.35	0.41	0.44	0.44
1	a	1	2	2	4	2	0.98	0.98	0.88	0.7
1	a	1	3	56	11	3	1.38	1.08	1.11	1.29
1	a	1	4	342	102	4	1.84	1.52	1.4	1.88
1	a	2	1	0	0	1	0.38	0.5	0.38	0.38
1	a	2	2	5	4	2	1	1	0.74	0.92
1	a	2	3	13	13	3	1.05	1.05	1.44	1.14
1	a	2	4	105	66	4	1.52	1.64	1.48	1.76
1	b	1	1	0	0	1	0.4	0.49	0.46	0.41
1	b	1	2	15	399	2	0.92	0.72	0.92	0.72
1	b	1	3	11	79	3	1.14	1.17	1.29	1.23
1	b	1	4	70	421	4	1.52	1.84	1.44	1.92
1	b	2	1	0	0	1	0.36	0.37	0.39	0.49
1	b	2	2	682	20	2	0.98	0.78	0.8	0.98
1	b	2	3	14	32	3	1.44	1.38	1.32	1.32
1	b	2	4	20	23	4	1.8	1.4	1.56	1.68

Table 1: Data table for excavation contexts in Behr Kaht operation 1

Corners .shp	**Levels .csv**	**Excavations PostGIS Table**
Each row represents a corner of an entire excavation unit as a point in 3d geographic space	*Each row represents a level within an excavation unit without any geographic information*	*Each row represents a level within an excavation unit as a 3d polyhedron*
op	op	op
sub-op	sub-op	sub-op
unit	unit	unit
corner	level	level
3d point geometry	Depth 1	3d Polyhedral Geometry
	Depth 2	lithic
	Depth 3	ceramic
	Depth 4	age
	lithic	
	ceramic	
	age	

Table 2: Various data tables of Arqueologos. Each column of this table lists the variables of the


Project Edit View Layer Settings Plugins Vector Raster Database Web Processing Help
Layers

Connection Info
Host: localhost
Port: 5432
Database: cck3
User: akara
Password: m36jackson

Files to Upload and Process
Directory to corners.shp
e/akara/test/excavations.shp
Browse...
Directory to levels.csv
/home/akara/test/levels.csv
Browse...

OK Cancel

Figure 4: Arqueologos GUI prompt to upload data

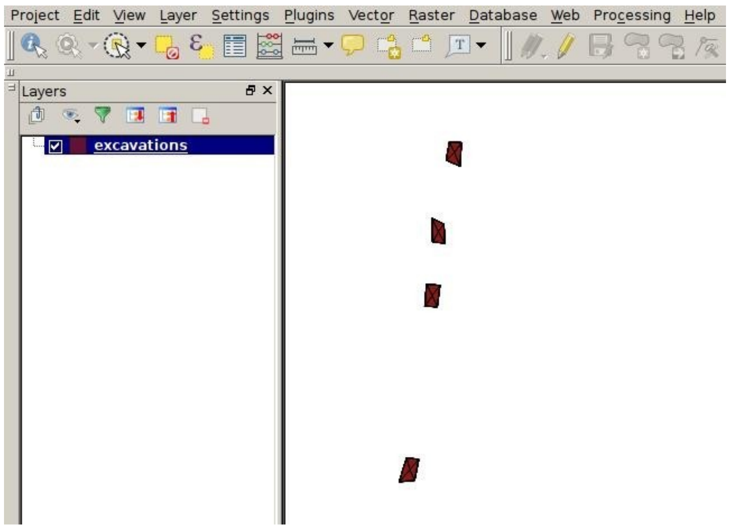
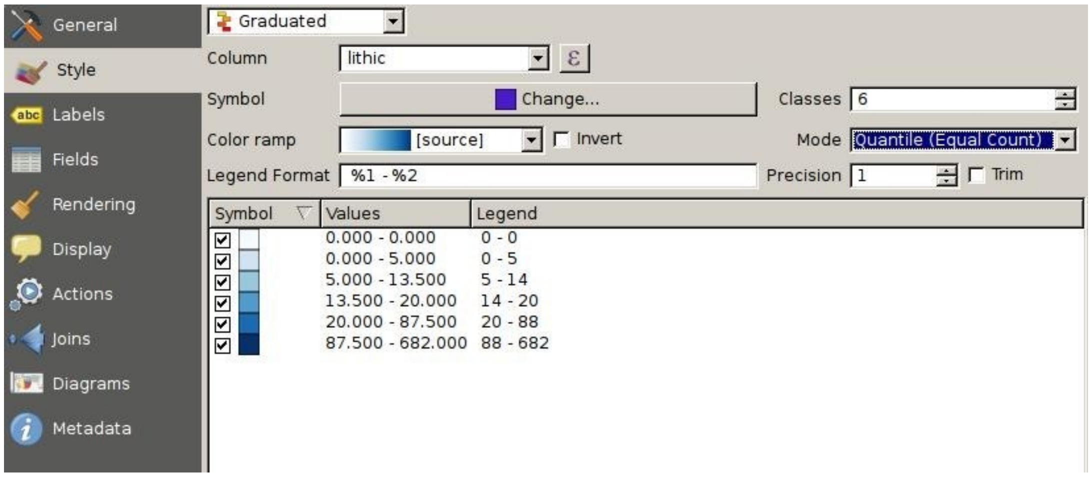

Project Edit View Layer Settings Plugins Vector Raster Database Web Processing Help
Layers
excavations

Figure 5: 2d representations of recently created 3d excavation layers on the QGIS canvas



Graduated

Column: lithic

Symbol: Change...

Classes: 6

Color ramp: [source] ☐ Invert

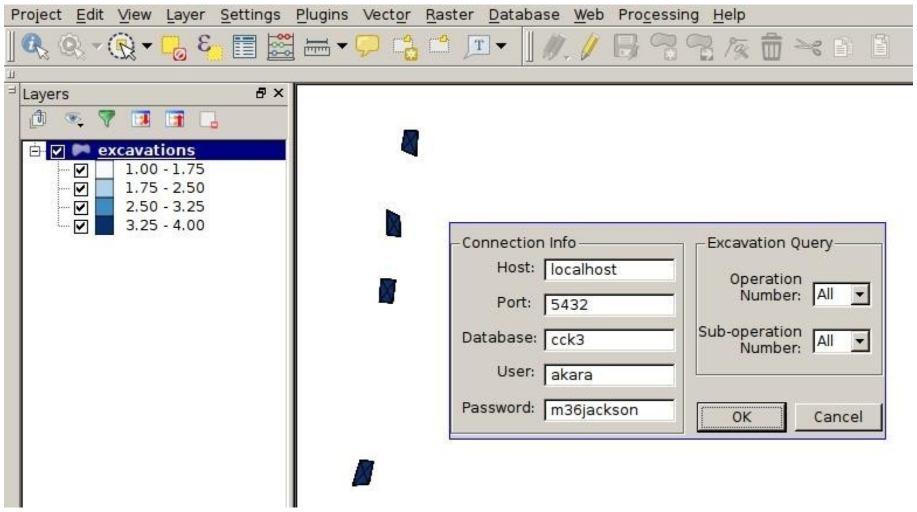
Mode: Quantile (Equal Count)

Legend Format: %1 - %2

Precision: 1 ☐ Trim

Symbol	Values	Legend
✓ ▢	0.000 - 0.000	0 - 0
✓ 🟦	0.000 - 5.000	0 - 5
✓ 🟦	5.000 - 13.500	5 - 14
✓ 🟦	13.500 - 20.000	14 - 20
✓ 🟦	20.000 - 87.500	20 - 88
✓ 🟦	87.500 - 682.000	88 - 682

Figure 6: QGIS interface where users can choose a graduated color scheme for excavation layers



excavations
- ✓ 1.00 - 1.75
- ✓ 1.75 - 2.50
- ✓ 2.50 - 3.25
- ✓ 3.25 - 4.00

Connection Info

Host: localhost

Port: 5432

Database: cck3

User: akara

Password: m36jackson

Excavation Query

Operation Number: All

Sub-operation Number: All

Figure 7: Arqueologos GUI prompt for the 3d visualizer

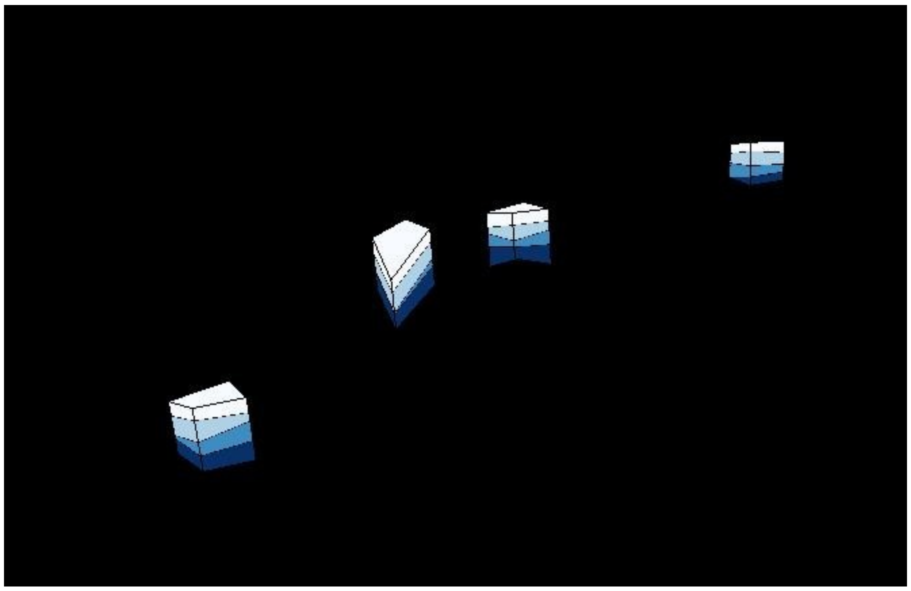


Figure 8: 3d visualization with excavation layers colored according to the age of their contents

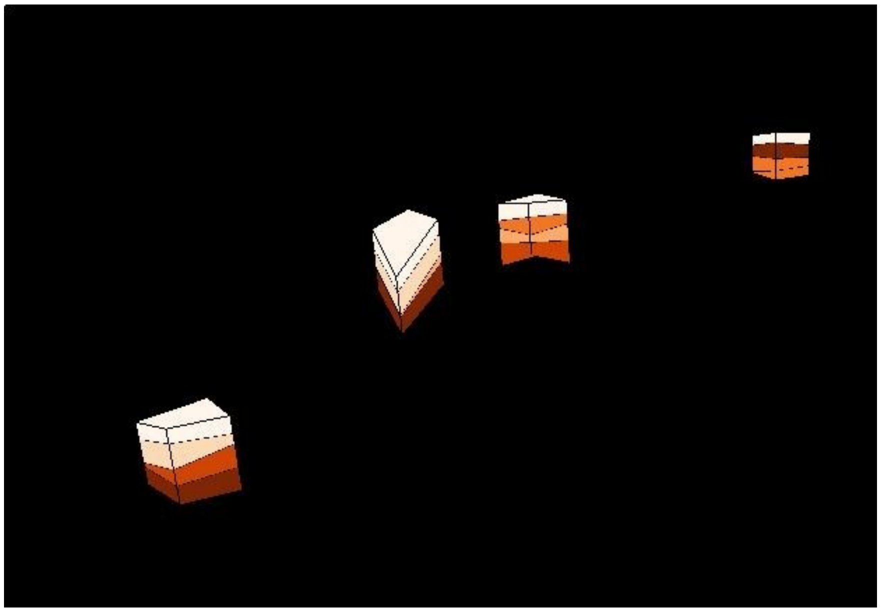


Figure 9: 3d visualization with excavation layers colored according to the amount of lithic artifacts discovered in each layer.

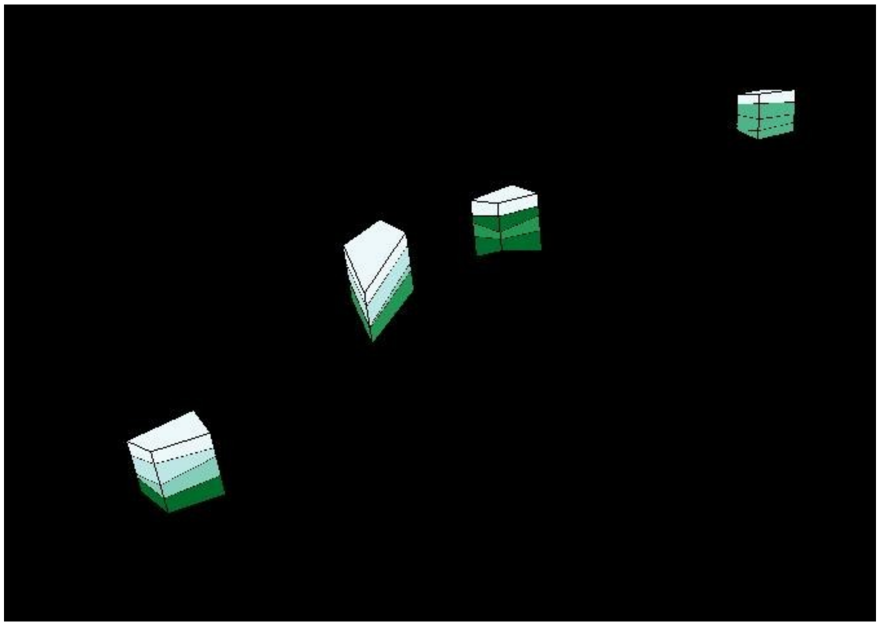


Figure 10: 3d visualization with excavation layers colored according to the amount of ceramic artifacts discovered in each layer.

```
from qgis.core import *
from qgis.analysis import *
from qgis.gui import *

from PyQt4.QtCore import *
from PyQt4.QtGui import *
from PyQt4 import QtCore, QtGui
from osgeo import osr, ogr
import os, sys, psycopg2, subprocess
import os.path

import resources
from arqueologos_dialog import ArqueologosDialog
from upload_helper import UploadHelper
from visualize_helper import VisualizeHelper

class Arqueologos:

    def __init__(self, iface):
        self.iface = iface
        self.plugin_dir = os.path.dirname(__file__)
        locale = QSettings().value('locale/userLocale')[0:2]
        locale_path = os.path.join(
            self.plugin_dir,
            'i18n',
            'Arqueologos_{}.qm'.format(locale))
        if os.path.exists(locale_path):
            self.translator = QTranslator()
            self.translator.load(locale_path)
            if qVersion() > '4.3.3':
                QCoreApplication.installTranslator(self.translator)

        self.actions = []
        self.menu = self.tr(u'&Arqueologos')
        self.toolbar = self.iface.addToolBar(u'Arqueologos')
        self.toolbar.setObjectName(u'Arqueologos')

    def tr(self, message):
        return QCoreApplication.translate('Arqueologos', message)

    def initGui(self):
        self.tableAction = QAction(QIcon(':/plugins/Arqueologos/icon.png'),
             "Upload/Process Data", self.iface.mainWindow())
        self.displayAction = QAction(QIcon(':/plugins/Arqueologos/icon.png'), "3d
             Excavation Viewer", self.iface.mainWindow())
        QObject.connect(self.tableAction, SIGNAL("triggered()"), self.makeTable)
        QObject.connect(self.displayAction, SIGNAL("triggered()"), self.display3d)
        self.iface.addPluginToMenu("&Arqueologos", self.tableAction)
        self.iface.addPluginToMenu("&Arqueologos", self.displayAction)

    def unload(self):
        for action in self.actions:
            self.iface.removePluginMenu(
                self.tr(u'&Arqueologos'),
                action)
            self.iface.removeToolBarIcon(action)

    def makeTable(self):

        self.dlg = UploadHelper()

        self.dlg.ui.textHost.setText('localhost')
        self.dlg.ui.textPort.setText('5432')
        self.dlg.ui.textDatabase.setText('cck3')
        self.dlg.ui.textUser.setText('akara')
        self.dlg.ui.textPassword.setText('m36jackson')
        self.dlg.ui.textPassword.setText('m36jackson')
        self.dlg.ui.cornersPath.setText('/home/akara/test/excavations.shp')
        self.dlg.ui.levelsPath.setText('/home/akara/test/levels.csv')

        def selectCorners():

self.dlg.ui.cornersPath.setText(QFileDialog.getOpenFileName(filter='*.shp'))
        def selectLevels():

self.dlg.ui.levelsPath.setText(QFileDialog.getOpenFileName(filter='*.csv'))
        self.dlg.ui.browseCorners.clicked.connect(selectCorners)
        self.dlg.ui.browseLevels.clicked.connect(selectLevels)

        self.dlg.show()
        result = self.dlg.exec_()

        if result:

            hostString = self.dlg.ui.textHost.text()
            portString = self.dlg.ui.textPort.text()
            databaseString = self.dlg.ui.textDatabase.text()
            userString = self.dlg.ui.textUser.text()
            passwordString = self.dlg.ui.textPassword.text()

            with psycopg2.connect("dbname=" + databaseString + ' user=' + userString
             + ' host=' + hostString + ' password=' + passwordString) as con:
                cur = con.cursor()

            def pydrize (polys):
                pydrStr = 'POLYHEDRALSURFACE Z ('
                for poly in polys:
                    pydrStr += poly[0][10:] + ', '
                pydrStr = pydrStr[:-2]
                pydrStr += ' )'
                return pydrStr

            def pygnize (points):
                stupid = range(len(points)) + [0]
                pygnStr = 'POLYGONZ(('
                for s in stupid:
                    pygnStr += points[s][9:-1] + ', '
                pygnStr = pygnStr[:-2]
                pygnStr += ' ))'
                return pygnStr

            arqTables = ['exc_corners', 'exc_levels', 'pygns', 'exc']
            cur.execute("SELECT tablename FROM pg_tables WHERE schemaname='public';")
            tables = cur.fetchall()
            for table in tables:
                if table[0] in arqTables:
                    cur.execute('DROP TABLE ' + table[0] + ' CASCADE;')
                    con.commit()

            cur.execute('CREATE TABLE exc_corners (op integer NOT NULL, subop char
             NOT NULL, unit integer NOT NULL, corner integer NOT NULL, lvl_no integer
             DEFAULT 0, pz geometry(POINTZ));')
            cur.execute('CREATE TABLE exc_levels (op integer NOT NULL, subop char NOT
             NULL, unit integer NOT NULL, lvl_no integer NOT NULL, lithic integer NOT
             NULL, ceramic integer NOT NULL, age integer NOT NULL, depth1 real NOT
             NULL, depth2 real NOT NULL, depth3 real NOT NULL, depth4 real NOT
             NULL);')
            con.commit()

            shapefile = ogr.Open(self.dlg.ui.cornersPath.text())
            layer = shapefile.GetLayer(0)
            prj_text = layer.GetSpatialRef().ExportToWkt()
            srs = osr.SpatialReference()
            srs.ImportFromESRI([prj_text])
            srs.AutoIdentifyEPSG()
            epsg = srs.GetAuthorityCode(None)
            for i in range(layer.GetFeatureCount()):
                feature = layer.GetFeature(i)
                op = str(feature.GetField("op"))
                subop = str(feature.GetField("subop"))
                unit = str(feature.GetField("unit"))
                corner = str(feature.GetField("corner"))
                geojson = feature.GetGeometryRef().ExportToJson()
                cur.execute("INSERT INTO exc_corners (op, subop, unit, corner, pz)
             VALUES (" + op + ", '" + subop + "', " + unit + ", " + corner + ",
             ST_SetSRID(ST_GeomFromGeoJSON('"  + geojson + "'), " + epsg + "));")
            con.commit();

            with open(self.dlg.ui.levelsPath.text(), 'rb') as csvfile:
                sql_copy = "COPY exc_levels "
                sql_from = "FROM STDIN "
                sql_with = "WITH (FORMAT csv, HEADER TRUE, DELIMITER ',');"
                cur.copy_expert(sql_copy + sql_from + sql_with, csvfile);
                con.commit()

            cur.execute('ALTER TABLE exc_corners ADD COLUMN cid SERIAL PRIMARY KEY;')
            cur.execute('ALTER TABLE exc_levels ADD COLUMN lid SERIAL PRIMARY KEY;')
            cur.execute('ALTER TABLE exc_corners ADD COLUMN lid integer REFERENCES
             exc_levels (lid);')
            con.commit()

            for c in range(1,5):
                sql_insert = "INSERT INTO exc_corners "
                sql_select = "SELECT op, subop, unit, " + str(c) + ", l.lvl_no,
                    ST_Translate(pz, 0, 0, (0 - depth" + str(c) + ")) "
                sql_from = "FROM exc_levels l "
                sql_join = "INNER JOIN exc_corners c USING (op, subop, unit) "
                sql_where = "WHERE c.corner = " + str(c) + "; "
                cur.execute(sql_insert + sql_select + sql_from + sql_join +
                    sql_where)
                con.commit()

            cur.execute("UPDATE exc_corners SET pz = ST_Translate(pz, 0, 0, -170);")

            cur.execute('INSERT INTO exc_levels '\
                         "SELECT op, subop, unit, 0, 0, 0, 0, 0, 0, 0, 0 FROM
             exc_levels GROUP BY op, subop, unit;")

            cur.execute('UPDATE exc_corners AS c SET lid = l.lid FROM exc_levels AS l
' \
                'WHERE l.op = c.op AND l.subop = c.subop AND l.unit = c.unit AND
l.lvl_no = c.lvl_no;')
            con.commit()

            sql_alter = 'ALTER TABLE exc_levels '
            sql_add = 'ADD COLUMN centroid geometry(POINTZ, 32616); '
            cur.execute(sql_alter + sql_add)
            con.commit()

            sql_update = 'UPDATE exc_levels AS l '
            sql_set = 'SET centroid = c.centrd '
            sql_from = 'FROM (SELECT op, subop, unit, lvl_no,
             ST_SetSRID(ST_MakePoint(avg(ST_X(pz)), avg(ST_Y(pz)), avg(ST_Z(pz))),
             32616) AS centrd '
            sql_from2 = '   FROM exc_corners '
            sql_group = '    GROUP BY op, subop, unit, lvl_no) AS c '
            sql_where = 'WHERE l.op = c.op AND l.subop = c.subop AND l.unit = c.unit
             AND  l.lvl_no = c.lvl_no; '
            cur.execute(sql_update + sql_set + sql_from + sql_from2 + sql_group +
             sql_where)
            con.commit()

            cur.execute("CREATE TABLE pygns (lid integer REFERENCES exc_levels(lid),
             pygn geometry(POLYGONZ, 32616), pid SERIAL PRIMARY KEY);")
            con.commit()

            cur.execute('SELECT op, subop, unit, lvl_no, lid FROM exc_levels;')
            levels = cur.fetchall()
            for level in levels:
                if level[3] == 0:
                    pass
                else:
                    botLid = level[4]
                    cur.execute('SELECT ST_AsText(centroid) FROM exc_levels WHERE lid
= ' + str(botLid) + ';')
                    centroidBot = cur.fetchall()[0][0]

                    cur.execute('SELECT lid FROM exc_levels WHERE op = ' +
str(level[0]) + " AND subop = '" + level[1] + "' AND unit = " + str(level[2]) + " AND
lvl_no = " + str(level[3] - 1) + ";")
                    topLid = cur.fetchall()[0][0]
                    cur.execute('SELECT ST_AsText(centroid) FROM exc_levels WHERE lid
= ' + str(topLid) + ';')
                    centroidTop = cur.fetchall()[0][0]

                    dumb = [1,2,3,4,1]
                    for i in range(4):
                        cur.execute("SELECT lvl_no, ST_AsText(pz) FROM exc_corners
" \
                         "WHERE corner = " + str(dumb[i]) + " AND (lid = " +
str(botLid) + " OR lid = " + str(topLid) + ") " \
                         "ORDER BY lvl_no DESC;")
                        corner1 = cur.fetchall()

                        cur.execute("SELECT lvl_no, ST_AsText(pz) FROM exc_corners
" \
                         "WHERE corner = " + str(dumb[i+1]) + " AND (lid = " +
str(botLid) + " OR lid = " + str(topLid) + ") " \
                         "ORDER BY lvl_no DESC;")
                        corner2 = cur.fetchall()

                        botTri = pygnize([centroidBot, corner1[0][1], corner2[0][1]])
                        topTri = pygnize([centroidTop, corner2[1][1], corner1[1][1]])
                        sides = pygnize([corner1[0][1], corner1[1][1], corner2[1][1],
corner2[0][1]])
                        cur.execute("INSERT INTO pygns VALUES (" + str(botLid) + ",
ST_GeomFromText('" + botTri + "', 32616));")
                        cur.execute("INSERT INTO pygns VALUES (" + str(botLid) + ",
ST_GeomFromText('" + topTri + "', 32616));")
                        cur.execute("INSERT INTO pygns VALUES (" + str(botLid) + ",
ST_GeomFromText('" + sides + "', 32616));")
                        con.commit()

            cur.execute('CREATE TABLE exc (op integer NOT NULL, subop char NOT NULL,
unit integer NOT NULL, ' \
                        'lvl_no integer NOT NULL, lithic integer NOT NULL, ceramic
integer NOT NULL, age integer NOT NULL, ' \
                            'depth1 real NOT NULL, depth2 real NOT NULL, depth3 real
NOT NULL, depth4 real NOT NULL);')
            sql_copy = "COPY exc "
            sql_from = "FROM '/home/akara/test/levels.csv' "
            sql_with = "WITH (FORMAT csv, HEADER TRUE, DELIMITER ',');"
            cur.execute(sql_copy + sql_from + sql_with);
            cur.execute("ALTER TABLE exc ADD COLUMN area geometry(POLYHEDRALSURFACEZ,
32616)")
            cur.execute('ALTER TABLE exc ADD COLUMN lid integer REFERENCES exc_levels
(lid);')
            cur.execute('UPDATE exc AS c SET lid = l.lid FROM exc_levels AS l ' \
                'WHERE l.op = c.op AND l.subop = c.subop AND l.unit = c.unit AND
l.lvl_no = c.lvl_no;')
            cur.execute('ALTER TABLE exc ADD PRIMARY KEY(lid);')
            con.commit()

            cur.execute('SELECT lid FROM exc;')
            exc = cur.fetchall()
            for lid in exc:
                print lid[0]
                cur.execute('SELECT ST_AsText(pygn) FROM pygns WHERE lid = ' +
str(lid[0]) + ';')
                pyhn = pydrize(cur.fetchall())
                cur.execute("UPDATE exc SET area = ST_GeomFromText('" + pyhn + "',
32616) WHERE lid = " + str(lid[0]) + ";")
                con.commit()

            uri = QgsDataSourceURI()
            uri.setConnection(hostString, portString, databaseString, userString,
passwordString)
            uri.setDataSource("public", "exc", "area")
            vlayer = QgsVectorLayer(uri.uri(), "excavations", "postgres")
            QgsMapLayerRegistry.instance().addMapLayer(vlayer)


    def display3d(self):
        self.dlg = VisualizeHelper()
        import numpy as np
        from vispy import geometry, app, scene
        import vispy
        from vispy.scene import cameras

        self.dlg.ui.textHost.setText('localhost')
        self.dlg.ui.textPort.setText('5432')
        self.dlg.ui.textDatabase.setText('cck3')
        self.dlg.ui.textUser.setText('akara')
        self.dlg.ui.textPassword.setText('m36jackson')

        self.dlg.show()
        result = self.dlg.exec_()
        if result:
                hostString = self.dlg.ui.textHost.text()
                portString = self.dlg.ui.textPort.text()
                databaseString = self.dlg.ui.textDatabase.text()
                userString = self.dlg.ui.textUser.text()
                passwordString = self.dlg.ui.textPassword.text()

                with psycopg2.connect("dbname=" + databaseString + ' user=' +
userString + ' host=' + hostString + ' password=' + passwordString) as con:
                    cur = con.cursor()
                    subOpString = 'null'
                    opString = 'null'
                    if self.dlg.ui.comboBoxSubOp.currentText() == 'All':
                        subOpString = "subop != 'fubar'"
                    else:
                        subOpString = "subop = '" +
self.dlg.ui.comboBoxSubOp.currentText()       + "'"
                    if self.dlg.ui.comboBoxOp.currentText() == 'All':
                        opString = 'op != 9999'
                    else:
                        opString = 'op = ' + self.dlg.ui.comboBoxOp.currentText()
                    whereClause = subOpString + " AND " + opString

                    canvas = scene.SceneCanvas(keys='interactive')
                    view = canvas.central_widget.add_view()
                    view.parent = canvas.scene
                    view.clip_method = 'viewport'
                    view.camera = scene.TurntableCamera(parent=view.scene, fov=45)

                    def pwnygons (pygSeq, breakC):

                        seq = []
                        start = 0
                        end = 0
                        for i in range(len(pygSeq)):
                            if (pygSeq[i] == breakC or i == len(pygSeq) - 1):
                                end = i
                                if breakC == " ":
                                    seq += [float(pygSeq[start:end])]
                                else:
                                    seq += [pwnygons(pygSeq[start:end], " ")]
                                start = i + 1
                        if breakC == ',':
                            seq = seq[0:-1]
                        return(seq)

                    cur.execute("SELECT lid, ST_NumGeometries(area) FROM exc WHERE "
+ whereClause + ";")
                    levels = cur.fetchall()
                    if len(levels) == 0:
                        error = QErrorMessage()
                        error.showMessage("No results. Try another query.")
                        error.exec_()
                    else:
                        lvlVerts = 0
                        newmesh = 0
                        excavations = self.iface.activeLayer()
                        renderContext = QgsRenderContext()

                        for i in range(len(levels)):
                            lid = levels[i][0]
                            nfaces = levels[i][1]
                            cur.execute("SELECT ST_AsText((ptts).geom) FROM (SELECT
ST_DumpPoints(area) AS ptts FROM exc WHERE lid = " + str(lid) + ") AS foo;")
                            rows = cur.fetchall()
                            lvlVerts = []

                            color = 0
                            exp = QgsExpression('"lid" = ' + str(lid))
                            req = QgsFeatureRequest(exp)
                            feats = excavations.getFeatures(req)
                            for feat in feats:
                                rend = excavations.rendererV2()
                                rend.startRender(renderContext,
excavations.pendingFields())
                                symb = rend.symbolForFeature(feat)
                                rend.stopRender(renderContext)
                                red = (symb.color().red())/255.0
                                green = (symb.color().green())/255.0
                                blue = (symb.color().blue())/255.0
                                color = np.array([red, green, blue, 1])
                                color = np.array([color, color, color])
                                color = np.array([color] * 16)

                            lineColor = np.array([0,0,0,1])
                            lineColor = np.array([lineColor] * 4)


                            for j in range(nfaces):
                                cur.execute("SELECT ST_AsText(ST_GeometryN(area, " +
str(j+1) + ")) FROM exc WHERE lid = " + str(lid) + ";")
                                faces = cur.fetchall()
                                face = faces[0][0]
                                faceVerts = pwnygons(face[12:-2], ',')
                                faceVerts = np.array(faceVerts)
                                faceVerts = np.around(faceVerts, decimals=3)
                                centering = np.array([1630600, 4822360, 0])
                                faceVerts = faceVerts - centering
                                if (faceVerts.shape == (3, 3)):
                                    lvlVerts.append(faceVerts)
                                else:
                                    lvlVerts.append(faceVerts[0:3])
                                    lvlVerts.append([faceVerts[0], faceVerts[2],
faceVerts[3]])
                                    line = scene.visuals.Line(faceVerts,
connect='strip', color=lineColor, parent=view.scene, method='gl', antialias=False)

                            lvlVerts = np.array(lvlVerts)
                            newmesh = scene.visuals.Mesh(lvlVerts, face_colors=color,
parent=view.scene)
                            #~ newmesh.update_gl_state(depth_test=True)

                        canvas.show()
                        app.run()
```