DOCUMENTACIÓN Y LEVANTAMIENTO FOTOGRAMÉTRICO DEL YACIMIENTO ARQUEOLÓGICO DE CÁPARRA. DESARROLLO DE UNA APLICACIÓN SIG 3D E INCLUSIÓN DE LOS DATOS OBTENIDOS.

CRISTINA TEJEDA SÁNCHEZ
TUTOR: ÁNGEL LUIS MUÑOZ NIETO

TRABAJO FIN DE MÁSTER
MÁSTER EN GEOTECNOLOGÍAS CARTOGRÁFICAS EN INGENIERÍA Y ARQUITECTURA. JUNIO 2017

UNIVERSIDAD DE SALAMANCA. CAMPUS DE ÁVILA

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS……..4

ÍNDICE DE TABLAS……..5

1. INTRODUCCIÓN……..6

1.1. LOCALIZACIÓN……6

1.2. CONTEXTO GEOGRÁFICO……8

1.3. CONTEXTO HISTÓRICO……9

- El Arco……9

- La Vía de la Plata……10

- Organización urbana……12

1.4. CONTEXTO ARQUEOLÓGICO……13

- Importancia del yacimiento……13

- Fases de excavación……14

- Interés/oportunidad del trabajo……16

1.5. REQUERIMIENTOS Y OBJETIVOS……17

2. LEVANTAMIENTO FOTOGRAMÉTRICO. METODOLOGÍA……19

2.1. ELECCIÓN DEL MÉTODO DE TRABAJO……19

- Valoración de alternativas……19

- Justificación de la alternativa elegida……21

2.2. TRABAJOS DE CAMPO……22

- Planificación de tomas……22

- Parámetros de la cámara……23

- Mediciones.....24

- Resultados.....24

2.3. PROCESADO FOTOGRAMÉTRICO.....25

- Selección de fotografías.....25

- Selección del software.....26

- Procesado por fases.....27

- Obtención de la nube de puntos.....27

- Puntos de referencia para el escalado.....29

- Filtrado de la nube.....29

- Vectorización.....29

2.4. PRODUCTOS OBTENIDOS A PARTIR DEL PROCESADO FOTOGRAMÉTRICO.....30

3. DESARROLLO DE UNA APLICACIÓN SIG MEDIANTE LIBRERÍAS QGIS Y VTK. INCLUSIÓN DE LOS DATOS OBTENIDOS.....32

3.1. ELECCIÓN DEL ENTORNO DE DESARROLLO.....32

- Elección del lenguaje de programación.....32

- Elección del software.....32

- Librerías.....33

3.2. DESARROLLO DE LA APLICACIÓN.....35

- Definición del proyecto.....35

- Funcionalidades del programa.....35

3.3. GENERACIÓN DE LA BASE DE DATOS.....39

3.4 PRODUCTOS OBTENIDOS.....42

4. CONCLUSIONES.....44

- Discusión de resultados.....44

- Líneas futuras……45

- Consideraciones finales……45

BIBLIOGRAFÍA……47

AGRADECIMIENTOS……49

ANEXO I. CÓDIGO DE LA APLICACIÓN……50

ANEXO II. INTERFAZ GRÁFICA DE LA APLICACIÓN……85

ANEXO III. NUBE DE PUNTOS DENSA……86

ANEXO IV. ALZADO, SECCIÓN Y PLANTA DEL ARCO……87

ANEXO V. POSIBLE RECONSTRUCCIÓN 3D. VISTAS DESDE EL INTERIOR DE LAS TERMAS Y EL FORO……88

ÍNDICE DE FIGURAS

Figura 1.1.1. Lusitania romana……6

Figura 1.1.2. Distribución de ciudades romanas según Fernández Corrales, 1989……..7

Figura 1.1.3. Localización (Mapa Topográfico Nacional 1:25000, hojas 575 y 598, IGN, CC-BY 4.0)……..7

Figura 1.2.1. Mapa de relieve y geológico del entorno de Cáparra (IDEE)……8

Figura 1.3.1. Ortofotos históricas del entorno de Cáparra (IDEEX)……..11

Figura 1.4.1. Plano de las Fases de excavaciones de Cáparra (Fuente: Centro de Interpretación de Cáparra)……..14

Figura 1.5.1. Diagrama de flujo de las fases del proyecto……18

Figura 2.2.1. Mediciones……24

Figura 2.3.1. Selección y organización de fotografías……25

Figura 2.3.2. Nube de puntos densa……28

Figura 2.3.3. Puntos de referencia……29

Figura 2.3.4. Vectorización……29

Figura 2.4.1. Nube de puntos y vectorización……30

Figura 2.4.2. Alzado SW y Secciones horizontal y vertical del Arco……31

Figura 2.4.3. Posible reconstrucción 3D. Vistas desde el interior de las termas y el foro ……30

Figura 3.2.1. Captura de la aplicación y diagrama explicativo……36

Figura 3.2.2. Apertura de atributos……37

Figura 3.2.3. Visualización de los datos referidos al arco mediante la aplicación desarrollada……38

Figura 3.3.1. Captura inicio QGIS Brighton……39

Figura 3.3.2. Apertura de capa vectorial en formato .dxf……39

Figura 3.3.3. Revisión y rectificación de entidades……40

Figura 3.3.4. Creación de atributos.....41

Figura 3.3.5. Búsqueda de servicio WMS en el Geoportal IDEE e incorporación a QGIS 41

Figura 3.3.6. Vista de QGIS con todas las capas incorporadas y procesadas.....42

Figura 3.4.1. Vista de la aplicación, con los datos recogidos.....42

Figura 3.4.2. Mapa de la zona principal del yacimiento, con ortofoto de fondo.....43

Figura 3.5.3. Mapa con la distribución de la domus.....43

ÍNDICE DE TABLAS

Tabla 2.1.1. Láser vs. Fotogrametría.....20

Tabla 2.1.2. Fotogrametría aérea vs. Terrestre.....21

Tabla 2.2.1. Planificación de tomas.....22

Tabla 2.2.2. Parámetros de la cámara.....23

Tabla 2.3.1. Fotografías seleccionadas/orientadas.....25

Tabla 2.3.2. Comparativa software fotogramétrico.....26

Tabla 3.1.1. Software empleado para el desarrollo de la aplicación.....33

Tabla 3.1.2. Librerías utilizadas en el proyecto.....33

1. INTRODUCCIÓN

1.1. LOCALIZACIÓN

El presente trabajo se centra en la documentación y levantamiento del Yacimiento arqueológico de Cáparra, ciudad romana hoy despoblada. Se situó dentro de la antigua provincia de Lusitania, al oeste de la Península Ibérica, provincia que ocupaba la mayor parte de la actual Portugal al sur del Duero y zonas fundamentalmente de Extremadura y la provincia de Salamanca, y que tenía la capital en Augusta Emerita (Mérida). A continuación se incluye un mapa del territorio que comprendía (Alonso Sánchez, Cerrillo M. de Cáceres, & Fernández Corrales, 1990):

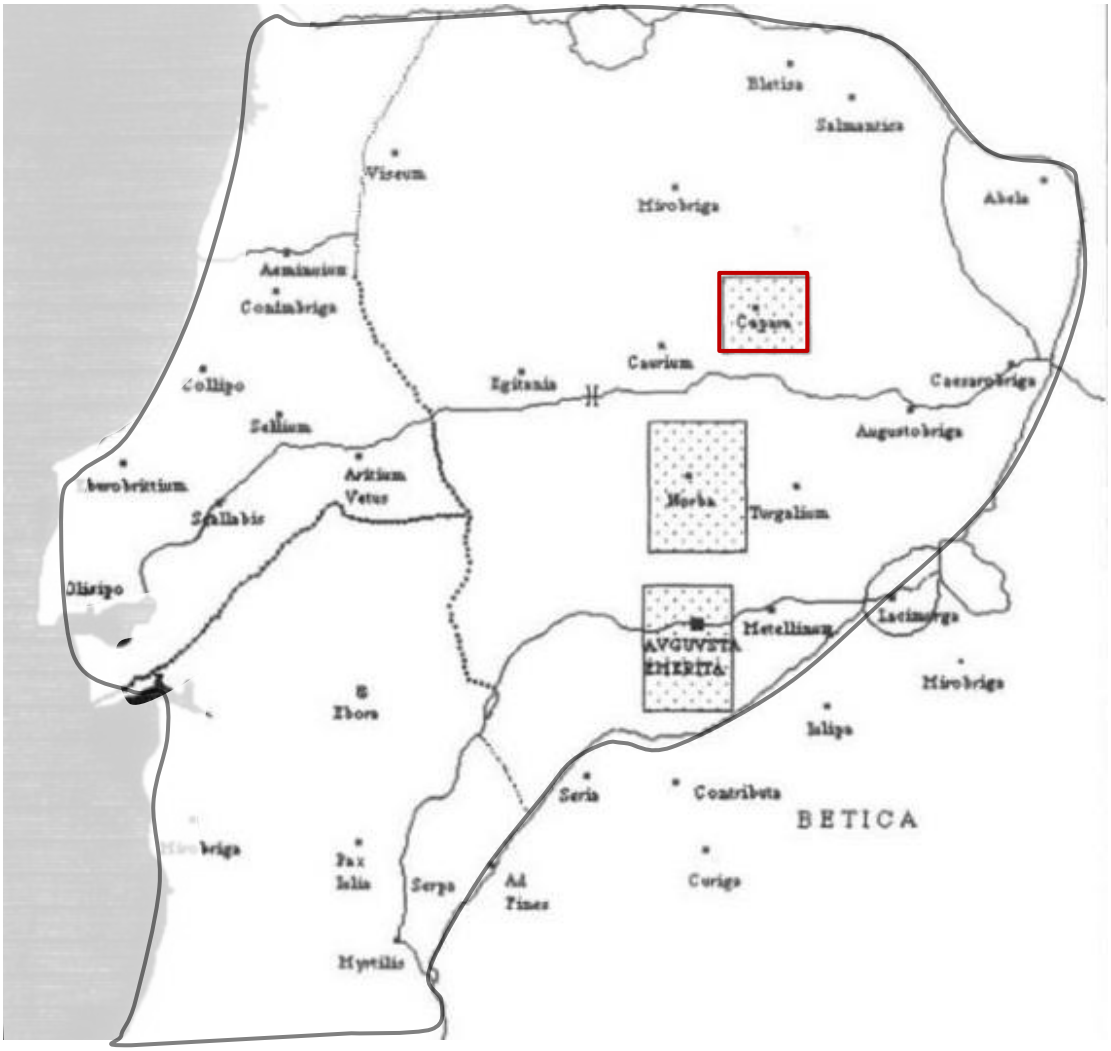


Figura 1.1.1. Lusitania romana (Alonso Sánchez, Cerrillo M. de Cáceres, & Fernández Corrales, 1990).

Aunque la ciudad de Cáparra no llego a tener el rango de ciudades como Augusta Emerita (Mérida) o Norba Caesarina (Cáceres), fue un municipio importante en la zona y estaba entre los principales núcleos urbanos entre el río Tajo y la Sierra de Gredos, núcleos que por otro lado no eran abundantes, respondiendo a la escasa urbanización que caracterizaba a la provincia de Lusitania, frente al mayor grado de presencia de ciudades que se observaba en la Bética.

El núcleo de Cáparra quedó despoblado con la invasión musulmana, y no se tiene constancia de que se incluyera entre las ciudades a repoblar con el retorno de los cristianos, por lo que quedó abandonado.

En el mapa de la derecha vemos la distribución de las ciudades romanas dentro de la actual Extremadura, con su zona de influencia. Mapa de Fernández Corrales, extraído del texto sobre el Poblamiento romano en Extremadura de (Cerrillo M. de Cáceres, 1997).

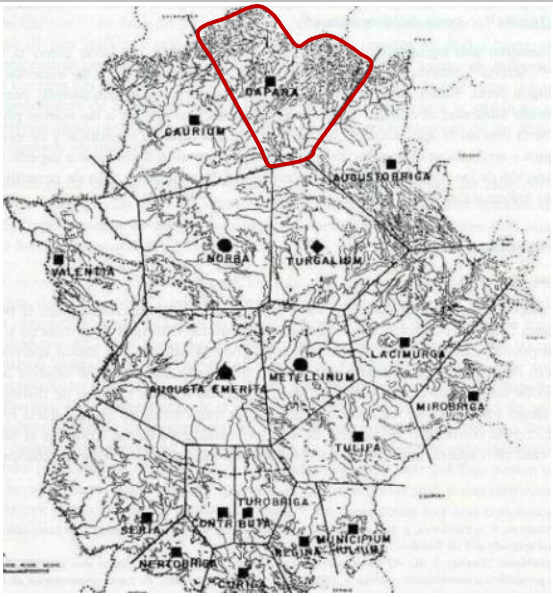


Figura 1.1.2. Distribución de ciudades romanas (Fernández Corrales, 1989).

En cuanto al entorno más cercano, Cáparra se encuentra situada en una de las zonas más fértiles de la provincia de Cáceres, el valle del Ambroz, y también próxima a la depresión del río Alagón. Se ubica en la dehesa Casablanca, entre los términos de Oliva de Plasencia y Guijo de Granadilla ("Ruinas romanas de Cáparra," n.d.), en las coordenadas:

Latitud: 40.1665651360024 (40º 9' 59.63" N)
Longitud: -6.100977953756683 (6º 6' 3.52" W)

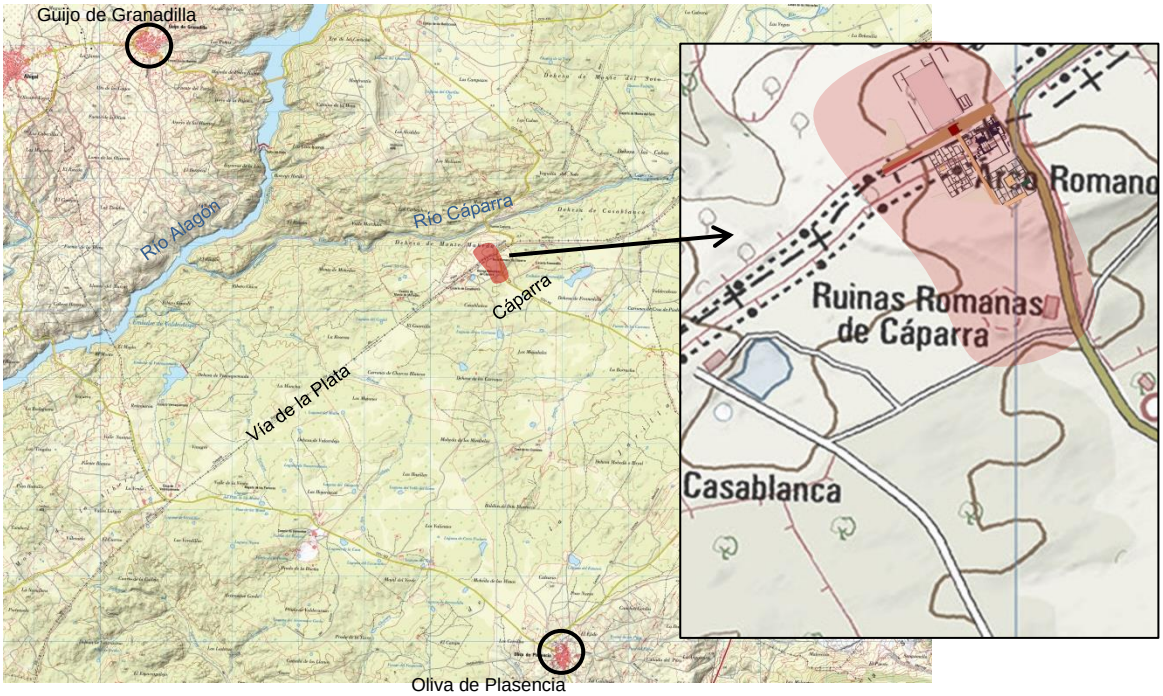


Figura 1.1.3. Localización (Mapa Topográfico Nacional 1:25000, hojas 575 y 598, IGN, CC-BY 4.0).

1.2. CONTEXTO GEOGRÁFICO

En el texto “Tres ejemplos de poblamiento rural romano en torno a ciudades de la Vía de la Plata: Augusta Emérita, Norba Caesarina Y Cápara” (Alonso Sánchez et al., 1990), se realiza un análisis exhaustivo y comparativo del marco geográfico en que estos asentamientos se producen y la forma en la que se pueblan estas tres ciudades. Quedándonos con lo esencial, si nos fijamos en la topografía del entorno, la ciudad de Cáparra se encuentra asentada sobre un espacio desigual en el que se unen zonas de topografía uniforme, con las primeras estribaciones del Sistema Central por el norte, las Sierras de Gredos, Béjar y Gata. Por tanto, encontramos zonas de dos tipos diferentes, por una parte terrenos terciarios integrados en la depresión del Alagón, y por otra, terrenos más elevados constituidos por el granito. En el núcleo principal del asentamiento no encontramos desniveles superiores a cinco metros, por tanto se trata de una zona bastante uniforme.

Se consulta en el Visualizador de la Infraestructura de Datos Espaciales de España-IDEE (“VisualizadorBasicoIDEE,” n.d.) los mapas de relieve, y los geológicos del IGME, a continuación vemos ambos superpuestos. Comprobamos que en efecto las características del terreno así como su composición, son las que se mencionan.

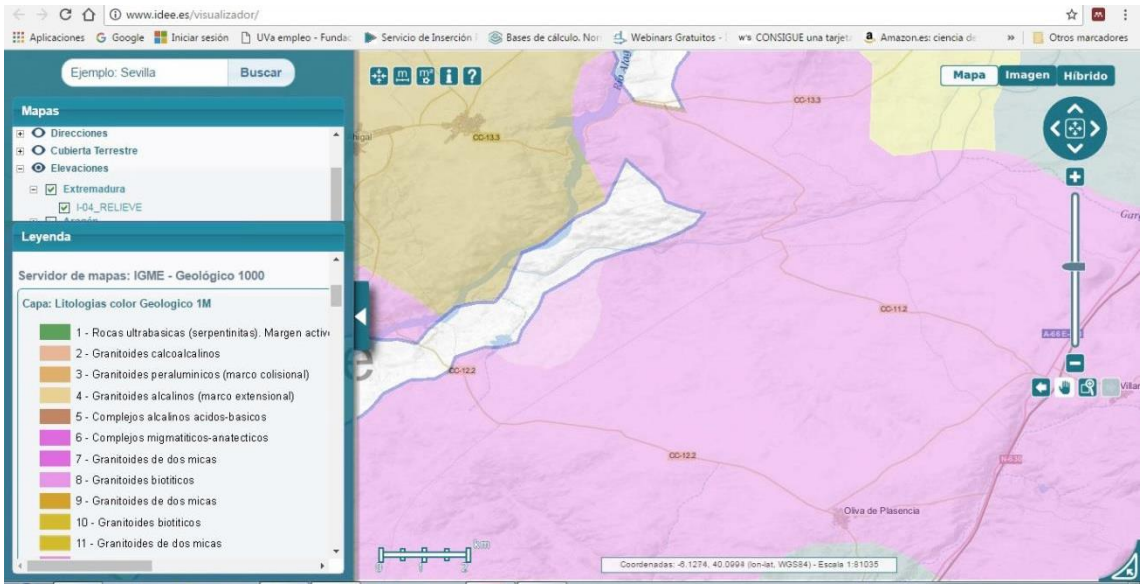


Figura 1.2.1. Mapa de relieve y geológico del entorno de Cáparra (IDEE).

Respecto a la ciudad y su territorio, la existencia de un asentamiento de época prerromana no está corroborada, aunque si existen escritos y algunos restos de cerámicas que parecen apuntar a esta posibilidad, aunque muy remotamente. Por tanto, la opinión más extendida es que se trata de un asentamiento romano. Los límites geográficos del territorio sobre el que esta ciudad extendió su poder, aunque no se pueden delimitar con exactitud, estarían afectados por las estribaciones del Sistema Central, las sierras situadas al norte de Plasencia y la fosa del Alagón contribuirían a marcar su área de influencia frente a otras ciudades lusitanas más o menos próximas, tales como Salmantica (Salamanca) al norte, Caurium (Coria) al suroeste y Augustobriga (Talavera la Vieja) al sureste. Aunque la ciudad de Cáparra no alcanzó rango colonial, tan solo el estatuto municipal, no implica que no contase con un territorio circundante sobre el que extender su influencia urbana.

En las zonas circundantes al área de Cáparra el número de asentamientos se reduce, con lo que las distancias entre los yacimientos crecen. Se trata de una característica propia de la provincia de Lusitania, como ya se ha dicho, el hecho de que los núcleos de población no fueran abundantes, y si más distanciados. Aunque este hecho en este entorno también puede tener otra justificación. Fuera de la depresión y la penillanura adyacente, la ocupación urbana casi desaparece, encontrando solo asentamientos puntuales. La ausencia de poblamiento en estas áreas estaría justificada por lo accidentado del terreno, y pasarían a ser terrenos destinados a espacios públicos y a ser explotados por la comunidad urbana.

1.3. CONTEXTO HISTÓRICO

Como ya se ha mencionado la ciudad de Cáparra se situó dentro de la antigua provincia romana de Lusitania. Es posible que el nombre de esta ciudad correspondiera ya a un asentamiento prerromano de las inmediaciones, aunque no existen indicios claros. Pero lo que hoy son las ruinas de la ciudad romana corresponden a una de las entidades de población de carácter estipendiario citadas por Plinio dentro de la Lusitania: los caparenses y convertida en los últimos años del siglo I d. C., en la época el emperador Vespasiano, en el Municipium flavium caparensis, tal como puede verse en el análisis de las inscripciones.

Se le conocen otros nombres, como Capara, Cappara, Capera o Kapasa, que podrían significar lugar de intercambio, trueque o mercado, ya que la situación de Cáparra es un cruce de caminos.

Como cuenta (Cerrillo M. de Cáceres, n.d.) en su texto “Cáparra. Municipium flavium caperensis”, la municipalización flavia sirvió para revitalizar a Cáparra con nuevas construcciones monumentales propias de la nueva situación a la que había ascendido. El ejemplo más claro es la construcción del tetrapylon, el conocido Arco de Cáparra, pero también se llevaron a cabo múltiples modificaciones dentro de la ciudad, como sucede en el foro o en otras áreas. Además, a partir de su conversión en municipio, también se puede observar un proceso de ornamentación urbana, primero con trabajos en granito, y ya en el siglo II, con la incorporación de elementos en mármol.

- El Arco.

La construcción del tetrapylon se incluiría dentro de la primera fase de ornamentación urbana, y tiene, como se ha mencionado, un carácter monumental y simbólico más que funcional. Se levanta en el punto en el que el kardo se encuentra con el decumanus, justo donde se hallaba el acceso al foro, y fue financiado como en muchos otros casos por un personaje influyente de la ciudad, Marcus Fidius Macer, como se deduce de las inscripciones encontradas.

Este edificio se convirtió en icono de la ciudad, incluso en nuestros días. Precisamente estos proyectos de monumentalización es lo que buscaban. Fue el único que se salvó

de la ruina, y el elemento que mejor se conserva hoy, visible ya antes de los procesos de excavación.

Está formado por cuatro pilares realizados en sillares de granito y núcleo de opus caementicium u hormigón romano, (del latín = caementum: escombros, piedra en bruto), un tipo de obra hecha de mortero y de piedras de todo tipo (de residuos, por ejemplo) y que tiene la apariencia del hormigón. Estos cuatro pilares dejan paso a cuatro arcos enfrentados dos a dos. De esos mismos cuatro pilares arranca una bóveda de arista que sostiene un ático, la zona más deteriorada actualmente. Es el único de este tipo conocido en la Península Ibérica. Dos de estos pilares sirvieron de soporte a dos inscripciones, de las que en la actualidad sólo se conserva una. En ella se cita al ciudadano Marcus Fidius Macer, que lo dedica a su madre, Bolosea Pelli f. y a su padre, Fidius Macri f. La otra la dedicó a su esposa, Iulia Luperca Luperci f. y ha desaparecido (Cerrillo M. de Cáceres, n.d.).

- La Vía de la Plata.

Se cuenta con abundante documentación de la ciudad de Cáparra desde la antigüedad. Desde las fuentes del mundo clásico, como Plinio, Ptolomeo o los Itinerarios, hasta los investigadores más recientes, han sido multitud los autores interesados en la ciudad y su entorno.

El hecho de que se encuentre atravesada por la calzada romana llamada Vía de la Plata ha favorecido esta situación. El discurrir de esta calzada romana, ha marcado la ruta a muchos investigadores, que han centrado su atención en esta ciudad, en su arco y en las abundantes inscripciones que han aparecido en su entorno y que han servido para poder contextualizarla históricamente.

La buena conservación de la ruta de la Plata ha facilitado el camino y atraído a viajeros y eruditos, que dieron buena cuenta de los restos que encontraban a su paso.

El Itinerario de Antonino, documento de la Roma antigua, que se supone redactado en el siglo III, en el que aparecen recopiladas las rutas del Imperio romano, ya sitúa la ciudad a unas 110 millas de Augusta Emerita (Mérida), en el discurrir de la mencionada vía, citándola como una de sus “mansiones” (núcleos de población): Cappara.

Según (Belloso, 2004):

> En su origen era una de las principales calzadas que con dirección sur-norte recorría la Hispania romana, prolongándose hasta Sevilla y Gijón, buscando salidas al mar, aunque se tiene constancia de su uso como camino natural desde tiempos prehistóricos. Ha servido al tránsito de legiones, ganado y peregrinos de distintas épocas y civilizaciones, convirtiéndose en un eje de comunicación cargado de historia.
>
> Los romanos pavimentaron la calzada y la dotaron de puentes, miliarios (grandes bloques graníticos que indicaban las millas) y mansios (lugares para el descanso de los viajeros, origen de poblaciones actuales). Árabes y cristianos la utilizaron durante las luchas por las tierras de la meseta, siendo también influyente el uso como ruta occidental de peregrinación a Santiago. Durante el

apogeo de la Mesta en el siglo XIII se convirtió en Cañada Real para la trashumancia del ganado, hasta el siglo XIX en que canaliza el trazado de la carretera Gijón-Sevilla (N-630), debiendo esperar más de un siglo para su transformación en Autovía de la Plata (A-66).

El recorrido discurre por 246 millas que supone un total aproximado de 361,128 kilómetros. Hoy día sigue siendo un eje de comunicación importante norte-sur, y aún transitado por peregrinos que se dirigen a Santiago.

A través del Visor de la Infraestructura de Datos Espaciales de Extremadura-IDEEX ("IDE Extremadura - Visualizador," n.d.), se han consultado los mapas del PNOA de Extremadura, así como las Ortofotos Históricas del PNOA y del Vuelo americano, para ver la evolución de la zona en los últimos años.

Vuelo americano (1956-1957)

PNOA-EX 2005-2006

PNOA-EX 2008-2011

PNOA-EX 2007-2010

Figura 1.3.1. Ortofotos históricas del entorno de Cáparra (IDEEX).

- Organización urbana.

Como muchas ciudades romanas la organización urbana de Cáparra corresponde a un planteamiento ortogonal, teniendo como eje principal la Vía de la Plata, que la atraviesa de noreste a suroeste.

Las dos vías principales presentes en este tipo de asentamientos, el kardo y el decumanus, perpendiculares entre sí, se cortan bajo el arco. El kardo, con disposición N-S, termina en el foro. El decumanus, en sentido SW-NE, coindice con la Vía de la Plata. Hacia el sur del decumanus se distribuyen las termas y las viviendas, y más al sur aún, una domus romana.

- Las Termas, como los demás elementos monumentales, datan del siglo I d.C., aunque sufrieron ampliaciones y remodelaciones a lo largo del tiempo. Situadas junto al Foro y al Arco, ocuparon una manzana completa y estuvieron dotadas de todos los elementos característicos de los edificios termales, que lograban la relajación corporal y muscular haciendo pasar al cuerpo por distintas temperaturas entre el frío y el calor. También supusieron un lugar donde hacer deporte o socializar.

Su distribución era la siguiente. Desde la entrada se accedía al vestuario o apodyterium en primer lugar, que servía para depositar en hornacinas en la pared los objetos personales. El frigidarium era la sala fría, el tepidarium la templada, y el caldarium la sala caliente. Otra estancia albergaba la sauna o laconicum. Caldarium, tepidarium y laconicum aumentaban su temperatura gracias a un doble suelo con una cámara de aire llamado hipocaustum, por el que circulaba aire caliente generado en un horno subterráneo (este sistema es el antecedente de las conocidas “glorias”). Existían también pequeñas piscinas conocidas como natatio, y una zona exterior en la que se podía hacer ejercicio, llamada palestra.

- Las viviendas, repartidas por las diferentes manzanas de la cuadrícula que no estaban ocupadas por elementos monumentales, podían ser de dos tipos: insulae o casas de vecinos, y las domus o grandes casas de tipo señorial. Por el tipo de construcciones encontradas se ha llegado a la conclusión que la manzana adyacente a las termas y próxima al decumanus maximus, albergaba bloques de casas y por tanto responde a la tipología de la insulae. Los bajos de estas construcciones estaban ocupados por espacios comerciales y de servicios independientes o tabernae, a los que se accedía desde los pórticos del kardo y el decumanus. Por otro lado, el espacio documentado al sur de las termas pertenecería a una importante domus de unos 1100 m2.

En dicha domus, tras traspasar el umbral de granito desde el decumanus minor, en la zona norte, se accede a un pequeño pasillo desde el cual se puede contemplar un patio porticado o peristilo, alrededor del cual se distribuyen las habitaciones de representación y otras pertenecientes a ambientes más privados (cubicula). En su lado occidental existe otro patio que distribuye a otras habitaciones relacionadas con el servicio. El fondo está ocupado por un espacio abierto, el hortus o jardín.

- El Foro, data de finales del siglo I d.C., dentro también del ya mencionado proceso de monumentalización. Centro político, religioso y lugar de encuentro de la ciudadanía. Ocupa una posición central, confluyendo frente a su entrada kardo y decumanus. El

recinto era rectangular y delimitado por edificios públicos excepto al SE. En el centro, un amplio espacio abierto de planta rectangular cubierto por losas de granito. A la izquierda, la plaza estaba flanqueada por un pórtico columnado, la basilica, donde se administraba justicia. En el lado opuesto se alzaba un edificio de iguales dimensiones pero cerrado, la curia, donde se reunía el senado local, la clase política. Al fondo se situaban tres templos de iguales dimensiones. Por los restos encontrados, ya desde las primeras excavaciones se determinó la presencia de esta área abierta, típica de las ciudades romanas. Es ampliamente analizada en el texto "Forum municipii Flavii Caparensis" (Cerrillo M. de Cáceres, 1998).

1.4. CONTEXTO ARQUELÓGICO

- Importancia del yacimiento.

Como ya se ha mencionado la ciudad de Cáparra estaba entre los principales núcleos urbanos entre el río Tajo y la Sierra de Gredos, dentro de la provincia romana de Lusitania. Alcanzó el rango de municipio, aunque fue abandonado tras la invasión musulmana, y por eso ha llegado hasta hoy en forma de ruinas. Su posición estratégica dentro de la Vía de la Plata, importante eje de comunicación entre el norte y el sur de la Península, la existencia de multitud de documentación así como de inscripciones que datan de la época y que permiten conocer la vida en este tipo de ciudades, así como la conservación de un elemento monumental como es el Arco, único de su clase en la Península Ibérica, dotan al yacimiento de cierta relevancia dentro de los de su tipo. La conservación, al menos del trazado, de todos los elementos típicos de una ciudad romana, como son el foro, las termas o las viviendas, permiten el análisis y la comparación con otras ciudades con similar organización.

Es por ello, que un gran número de investigadores a los largo de los años se han interesado por ella, bien de manera única o dentro del contexto de la Vía de la Plata. Hasta que a partir del año 1929 y en múltiples fases, se procedió a la excavación de distintas partes, quedando aún hoy una gran porcentaje de la ciudad enterrada.

Los últimos descubrimientos han dejado al descubierto el anfiteatro, que aunque de menor envergadura que los de otras ciudades romanas de Extremadura, como es el caso de Mérida, revela que Cáparra fue un asentamiento con una población de tamaño importante. Su construcción un tanto efímera y poco robusta, con graderío completamente de madera, caracterizó a los anfiteatros de ciudades de tamaño medio. También la aparición de dos de las puertas de acceso descubre su condición de núcleo amurallado y defensivo, lo que de nuevo demuestra su relevancia dentro de su entorno.

- Fases de excavación.

Las primeras excavaciones las realizó A. C. Floriano en 1929 y continuó J. M. Blázquez en la década de los años sesenta. A partir de 1988 los terrenos fueron adquiridos por la Diputación de Cáceres y desde la Universidad de Extremadura se llevaron a cabo varias campañas hasta 1997 financiadas por la Consejería de Cultura de la Junta de Extremadura. En 2001 dentro del Proyecto Alba Plata, A. Bejarano dirigió una campaña a través de la cual ha permitido conocer las termas públicas frente al foro, un área doméstica, las tabernae, la puerta SE., la construcción del anfiteatro y un tramo de la vía romana conocida como de la Plata en su trayecto urbano.

Figura 1.4.1. Plano de las Fases de excavaciones de Cáparra (Centro de Interpretación de Cáparra).

El proyecto Alba Plata (1998-2004).

En 1997 se iniciaban los trámites para la declaración de la Vía de la Plata como Bien de Interés Cultural, bajo la categoría de Sitio Histórico. Asimismo, se solicitaba a la UNESCO la declaración de Patrimonio de la Humanidad, en razón del valor histórico, arqueológico, etnográfico y ecológico de la Vía.

En su trazado se ubican dos ciudades Patrimonio de la Humanidad (el Conjunto Arqueológico de Mérida -1993- y la Ciudad Vieja de Cáceres -1986-), además de varios núcleos urbanos declarados conjuntos histórico-artísticos (Zafra -1965-, Galisteo -1991-, Plasencia -1958- y Hervás -1969-).

Por tanto se vio la necesidad de revitalizar de forma integral la propia Vía de la Plata y todo el patrimonio natural y cultural conectado por este eje, y dotarlos de la infraestructura necesaria para su conversión en productos turístico-culturales, labor realizada por la Oficina de Gestión del Proyecto Alba Plata, mediante la financiación del Banco Europeo de Inversiones y la Junta de Extremadura.

Su objetivo, convertir la antigua calzada romana a través de la recuperación, adecuación, señalización y puesta en valor de sus recursos naturales, culturales e históricos.

Actuaciones del proyecto Alba Plata (Belloso, 2004):

1. La Vía de la Plata como itinerario señalizado:

   Se han realizado actuaciones sobre la calzada romana en todo su recorrido por Extremadura (adquisiciones de terrenos, excavaciones, miradores, zonas de descanso) y procedido a su señalización con cubos de granito, pintados con trazos de amarillo (camino transitable), de verde (calzada romana), o ambos.

2. Los Centros de Interpretación:

   Ayudan a conocer la historia a través de las nuevas tecnologías expositivas. Entre ellos se encuentra el de Cáparra.

3. Los Albergues:

   Donde el viajero puede dormir, descansar, comer y disfrutar de la zona. En el entorno de Cáparra el de Oliva de Plasencia.

4. El patrimonio restaurado visitable:

   Con intervenciones de excavación y consolidación en yacimientos arqueológicos (Cáparra), consolidación de elementos de arquitectura militar, inmuebles religiosos, inmuebles civiles, e industriales.

5. Las aulas culturales:

   Uso que se le ha dado al convento de San Francisco en Arroyo de la Luz.

- Interés/oportunidad del trabajo.

Este trabajo se plantea buscando las ventajas y oportunidades de relacionar el mundo de la Arquitectura con la Geomática. La Geomática es fundamental para muchas de las ciencias que utilizan datos espacialmente referenciados y ha tenido un importante crecimiento en su aplicación en multitud de áreas, entre ellas la Arquitectura. Se pueden encontrar multitud de estudios recientes que analizan sus posibilidades: "Advanced technologies for archaeological documentation: Patara case" (Ergincan, Çabuk, Avdan, & Tün, 2010), "Digital Representation of Archeological Sites. Recent Excavation at Alba Fucens" (Paris, Liberatore, & Wahbeh, 2012), "Archeological excavation monitoring using dense stereo matching techniques" (Dellepiane, Dell'Unto, Callieri, Lindgren, & Scopigno, 2013).

En un entorno arqueológico como el que es objeto de este proyecto, podemos hacer y es lo que se ha venido haciendo, un levantamiento clásico, tomas medidas, dibujas el objeto en tres dimensiones, bien sea a mano o por medios informáticos, buscando la representación del elemento para conocerlo mejor, para documentarlo, para tener referencias de medidas, de donde se sitúa cada descubrimiento. Pero si además de todo esto, utilizamos medios geomáticos, podemos obtener múltiples ventajas. Las posibilidades de tener un levantamiento preciso con medios manuales son más reducidas, porque existe el factor humano, siempre puedes equivocarte al tomar una medida o incluso tener un error de interpretación. Pero si por el contrario, el modelo 3D se obtiene de manera precisa, a partir de fotos o escaneos del lugar, la posibilidad de error es casi nula. Además, en un levantamiento manual tienes que tomar cada medida que necesites in situ, pero si tomas datos mediante fotogrametría o láser y reconstruyes el modelo a escala, podrás tomar a posteriori las medidas que sean necesarias. Desde este punto de vista se aborda el trabajo en cuestión.

Se considera interesante el levantamiento del yacimiento por medios fotogramétricos, para su estudio desde un punto de vista arquitectónico, ya que mucha de la información recogida durante las excavaciones ha sido mediante métodos tradicionales. Entre los productos obtenidos tendremos alzados, plantas y modelo 3D precisos y métricos, de forma que se podrán utilizar con fines técnicos, para documentar el yacimiento, o ante futuras intervenciones.

Pero no solo desde un punto de vista técnico pueden sernos útiles los datos aportados, también desde un punto de vista de representación. La fotogrametría nos aporta un modelo a escala, real, pero también con toda la información de color y texturas, por tanto, muy representativo del elemento. Para una persona que no conozca el monumento, es mucho más visual una reconstrucción como esta, real, que una interpretación propia, aunque siempre se pueden combinar, como veremos, la representación real de lo existente, con una interpretación de lo que pudo haber sido.

Además, se plantea la posibilidad de recoger todos los datos, tanto los existentes como los generados ahora o en el futuro, en una base de datos, e incorporarlos a una aplicación, una especie de sistema de información patrimonial, que pueda ser utilizado tanto como por un público especializado como por el general, un medio de difusión. Se considera una aportación más, no solo la recopilación de la información sino también su gestión, con la ventaja de tratarse no solo de una aplicación al uso, sino un producto geomático, un sistema de información, y por tanto con georeferenciación.

1.5. REQUERIMIENTOS Y OBJETIVOS

Se pretende documentar el entorno arqueológico de la Ciudad Romana de Cáparra. Se va a desarrollar la documentación 2D, planos, en formato shp, y los atributos asociados que correspondan a cada elemento según la información de que se disponga. Por otro lado se va a desarrollar la documentación 3D, nube de puntos, por fotogrametría.

Debido a que el software SIG existente está más orientado a la documentación 2D o a la generación de documentos 3D vinculados al terreno, pero no tanto a una visualización paralela y conectada de la documentación 2D y 3D, por ejemplo, desde un punto de vista arquitectónico, relacionando cada capa con su nube de puntos, alzados o secciones, se plantea el desarrollo de una aplicación propia a partir de librerías externas, que permita lo dicho anteriormente.

- Objetivos:

· Levantamiento fotogramétrico de la Ciudad Romana de Cáparra.
· Generación de la nube de puntos mediante el software PhotoScan.
· Desarrollo de cartografía y atributos con AutoCAD/ QGIS.
· Desarrollo de una aplicación SIG 3D en lenguaje c++ y a partir de las librerías de QGIS y VTK. Inclusión de los datos obtenidos.

- Productos a generar:

· Sistema de Información 2D/3D, a modo de base de datos con toda la información referida al yacimiento (cartografía, atributos, nube de puntos…) con la posible finalidad de abordar una intervención sobre el mismo, o con fines de divulgación o turísticos.
· Otros posibles productos: mapas temáticos, alzados, secciones, 3d, recorrido en vídeo…

- Requerimientos:

· En este proyecto, y desde un punto de vista arquitectónico, se prima el escalado del objeto por la necesidad de toma de medidas, y no tanto que se encuentre referenciado en un sistema de coordenadas global.
· Además, lo que se busca es una representación arquitectónica en dos y tres dimensiones, más que una representación del terreno. Por tanto, se requiere mayor detalle en rango cercano, y que se puedan incorporar fácilmente a los datos nuevos elementos que aparezcan en sucesivas excavaciones.

Se incluye un diagrama con las fases en las que se estructura el proyecto, que se explican en sucesivos puntos, y los objetivos esperables de cada una de ellas.

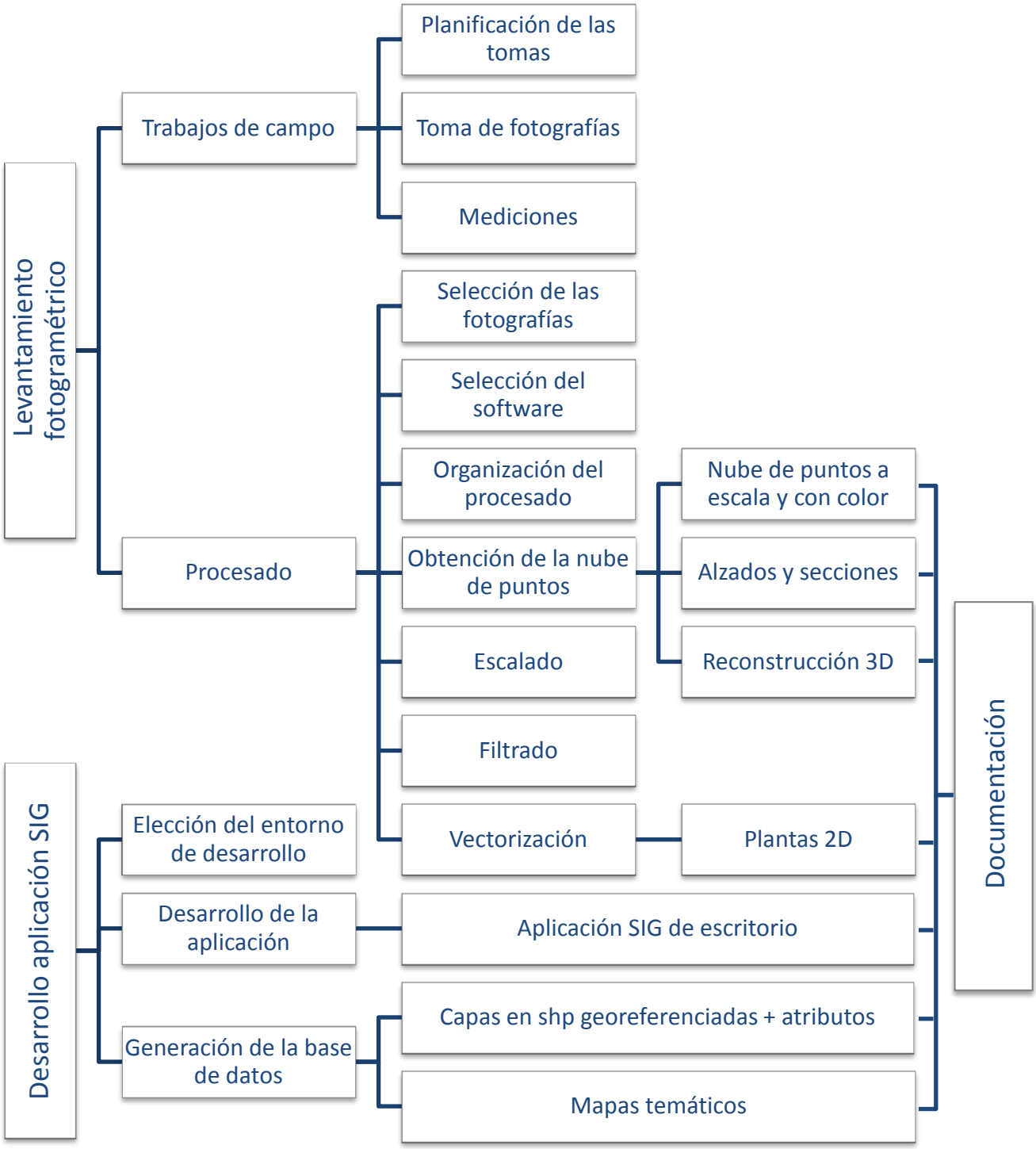


Figura 1.5.1. Diagrama de flujo de las fases del proyecto.

2. LEVANTAMIENTO FOTOGRAMÉTRICO. METODOLOGÍA

2.1. ELECCIÓN DEL MÉTODO DE TRABAJO

- Valoración de alternativas.

Se comienza con análisis de estudios ya existentes en este ámbito. Más concretamente los relacionados con el patrimonio y con las técnicas de levantamiento, y en especial aquellos que hacen comparativas entre las distintos métodos de trabajo.

Por ejemplo, en el artículo "A Multi-Data Source and Multi-Sensor Approach for the 3D Reconstruction and Web Visualization of a Complex Archaelogical Site: The Case Study of Tolmo De Minateda" (Torres-martínez, Seddaiu, Rodríguez-gonzálvez, Hernández-lópez, & González-aguilera, 2016) se valora la complejidad de levantamiento de ciertos sitios arqueológicos, donde la posibilidad de abordar con ciertos métodos se ve dificultada, y concluyen con la necesidad de un enfoque multi-sensor como la solución óptima para proporcionar una reconstrucción en 3D y la visualización de estos sitios complejos. De forma que la toma de datos se produce desde una plataforma aérea ultraligera, un vehículo aéreo no tripulado, un escáner láser terrestre y fotogrametría terrestre. El modelo 3d se obtiene combinando todos los datos obtenidos mediante las diferentes técnicas, aprovechando los beneficios de cada una: la alta densidad de puntos de la fotogrametría, el potencial de las plataformas aéreas ligeras y no tripuladas…

Por otro lado, Kersten, Mechelke, & Maziull, en su artículo "3d Model Of Al Zubarah Fortress In Qatar - Terrestrial Laser Scanning Vs. Dense Image Matching" (Kersten, Mechelke, & Maziull, 2015) contraponen directamente datos láser frente a fotogrametría y analizan las desviaciones, siendo más precisos los tomados con láser, aunque llegando a valores similares con Agisoft PhotoScan, no con otros programas, con la utilización de puntos de control en lugar de distancias de referencia para escalar. Con los puntos de control la fiabilidad es similar a la de láser, no obstante los resultados obtenidos mediante distancias de referencias tampoco es malo, sobre todo si no se trata de un objeto de gran tamaño como es el caso, en el que las desviaciones se van acumulando.

No obstante, son significativas las ventajas que cualquiera de estas técnicas ofrecen sobre la metodología tradicional en entornos arqueológicos, pues como se analiza en el artículo "Point cloud vs drawing on archaeological site" (Alby, 2015), ofrecen las posibilidad de representar el entorno completo de una forma bastante más rápida que por levantamiento tradicional, y más precisa.

Si analizamos las diferencias entre el láser y la fotogrametría, vemos como una y otra ofrecen sus ventajas e inconvenientes. A continuación se incluye una tabla comparativa entre ambos métodos de adquisición de datos.

Láser	**Fotogrametría**
Mejor precisión en z	Mejor precisión en xy
Accesibilidad a zonas difíciles	Problemas de oclusiones
Sensor activo: no requiere luz	Sensor pasivo: requiere luz
Difícil extracción de contorno y aristas	Fácil extracción de contorno y aristas
Baja densidad de puntos	Alta densidad de puntos
Sensor monocromático	Sensor pancromático
Cobertura discontinua	Cobertura continua de la imagen
Adquisición directa coordenadas	Adquisición indirecta coordenadas
Tiempo de escaneo	Toma rápida de fotografías
Campo de visión según equipo	Mayor campo de visión
Alto coste del equipo	Bajo coste del equipo

Tabla 2.1.1. Láser vs. Fotogrametría.

Si analizamos las características de cada método de adquisición de datos, y las valoramos en base al objeto de este proyecto, podemos establecer como elementos a favor del láser la adquisición directa de coordenadas, mientras que a favor de la fotogrametría estarían la mejor precisión en xy, la alta densidad de puntos, el sensor pancromático, el tiempo de toma de datos y el coste del equipo.

En este caso no se consideran relevantes los problemas de oclusiones, porque no existen elementos interpuestos, ni que el sensor requiera luz, pues la toma se va a realizar de día y el objeto es exterior.

La adquisición directa de coordenadas sin duda es una ventaja, no obstante en este proyecto, y desde un punto de vista arquitectónico, se prima el escalado del objeto por la necesidad de toma de medidas, y no tanto que se encuentre referenciado en un sistema de coordenadas global.

Si atendemos a las ventajas de la fotogrametría para este trabajo, nos quedaría discernir entre la toma de fotos terrestre o aérea, con vehículos no tripulados. De igual forma que en la comparación láser/fotogrametría, existen ventajas y desventajas en cada elección. Si la finalidad es la ortofoto, probablemente la fotogrametría aérea mediante UAVs sería más adecuada, pues nos ofrece fotografías con una vista paralela al terreno, por tanto la distribución de puntos será mejor. Si por el contrario, queremos una representación en detalle de ciertos elementos, la fotogrametría terrestre de rango cercano será mejor.

Fotogrametría aérea	Fotogrametría terrestre
Mayor calidad en la ortofoto	Menor calidad en la ortofoto
Menor calidad en rango cercano	Mayor calidad en rango cercano
No disponibilidad de equipo	Disponibilidad de equipo

Tabla 2.1.2. Fotogrametría aérea vs. Terrestre.

Está claro, en base a todo el análisis previo, que cada método presenta ventajas e inconvenientes, y la elección de uno u otro deberá verse apoyada por los objetivos que se persiguen en cada caso y las necesidades que deben ser satisfechas.

- Justificación de la alternativa elegida.

Finalmente se ha optado por fotogrametría terrestre como método de adquisición, por diversos motivos, aunque siendo consciente de que también existen puntos en contra. En primer lugar las ventajas descritas anteriormente frente al láser: mejor precisión en xy, alta densidad de puntos, cobertura continua, el sensor pancromático, etc. También un punto a tener en cuenta es el hecho de que se dispone del equipo necesario (réflex Nikon D3300) y la manejabilidad de este, por lo que el coste es reducido y el tiempo de trabajo en campo menor.

Se pretendía comprobar las posibilidades que pueden llegar a ofrecer medios de bajo coste, y se ha comprobado que son aceptables, si bien no se alcanza la precisión a la que se puede llegar con otros medios. Se valora en este sentido el hecho de que ante nuevas excavaciones, se puede hacer una rápida adquisición de datos para incorporar a los existentes, sin necesidad por ejemplo, de hacer un vuelo completo, en el caso de haberse empleado fotogrametría aérea mediante UAVs.

Respecto al empleo de fotogrametría terrestre frente a aérea, está precisamente el hecho de que la terrestre nos permite mayor detalle en rango cercano, y que se pueden incorporar a los datos nuevos elementos que aparezcan desvinculados, por ejemplo elementos del hogar, nuevos tramos de calzada…o incluso nuevas manzanas al completo.

En contra se tiene la menor calidad de la ortofoto respecto a la que se podría haber obtenido con un vuelo de dron, por quedar zonas sin cobertura, pero queda patente que lo que se busca en este trabajo es más una representación arquitectónica en dos y tres dimensiones, que una representación del terreno. Y también la adquisición de coordenadas, que debería hacerse con apoyo externo, pero que de nuevo, dado los fines que se buscan, se considera suficiente el escalado del modelo.

2.2. TRABAJOS DE CAMPO

- Planificación de tomas.

Previamente a la consecución de los trabajos de campo se hace necesario el análisis del entorno a procesar y la planificación de las tomas, con el fin de que los datos recogidos sean adecuados para su procesamiento posterior. Una incorrecta planificación puede llevarnos a la imposibilidad de tratar los datos o a que los resultados obtenidos no sean los óptimos. De hecho para el presente trabajo se llevó a cabo una rápida adquisición de fotografías, para una primera aproximación al elemento, pero sin un adecuado planteamiento, y como cabe esperar los resultados se alejaban mucho de ser buenos, aunque sí orientativos para que la toma definitiva fuera correcta.

Tras un análisis exhaustivo, por tratarse de un yacimiento arqueológico que corresponde a una ciudad romana, con un trazado ortogonal, en el que pueden diferenciarse claramente cada una de las manzanas que la constituían, se decide dividir el objeto de estudio por zonas, correspondiendo con dichas manzanas. De esta forma, se realizan anillos entorno a cada subdivisión, haciendo capturas de forma paralela a las caras y con un solape entre ellas de un 30% aproximadamente.

La siguiente tabla recoge una relación de las zonas en las que se ha trabajado y los datos obtenidos en cada una de ellas:

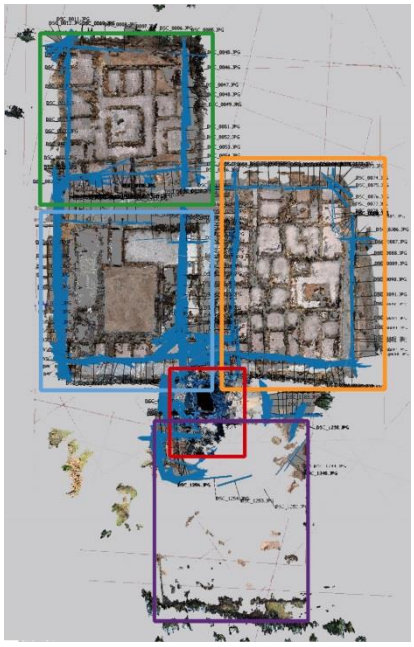
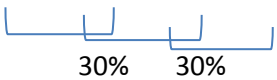
Anillos				**Esquema**
	Anillo	**Zona**	**Nº Fotos**	
	Anillo 1	Domus	55 fotografías	
	Anillo 2	Viviendas	86 fotografías	
	Anillo 3	Termas	72 fotografías	
	Anillo 4	Arco	27 fotografías	
	Anillo 5	Foro	28 fotografías	
Solape: 				
Distribución: Paralela				

Tabla 2.2.1. Planificación de tomas.

- Parámetros de la cámara.

Para la toma de fotografías se ha empleado una cámara réflex Nikon modelo D3300, con objetivo 18-55 mm. Se ha utilizado en modo manual, prestando especial atención a los siguientes parámetros:

- Distancia focal más pequeña, 18 mm, la que nos proporciona mayo campo.
- Selección de enfoque automático, pero se enfoca siempre a infinito.
- ISO-100 a 400. Se selecciona un día y un momento del día en concreto con buena luminosidad, pero sin sol directo, de este modo no es necesario incrementar el valor del ISO, que introduciría ruido en las fotografías.
- Modo de medición:

Los parámetros específicos de la cámara más importantes se recogen en la siguiente tabla:

Modelo Nikon D3300	
Tipo	Cámara digital reféx
Montura del objetivo	Montura F de Nikon
Ángulo de visión efectivo	Formato DX de Nikon; distancia focal equivalente a aprox. 1,5 veces la de los objetivos con un ángulo de visión de formato FX
Píxeles efectivos	24,2 millones
Sensor de imagen	Sensor CMOS de 23,5 x 15,6 mm
Tamaño de imagen (píxeles)	6000 x 4000 px (Grande)
Formato de archivo	JPEG
Visor	Visor réflex de objetivo único con pentaespejo a nivel de los ojos
Cobertura del fotograma	Aprox. 95 % horizontal y 95 % vertical

Tabla 2.2.2. Parámetros de la cámara.

Los parámetros que más afectan a la calibración de la cámara:

- Distancia focal: 18 mm
- Formato: 23,5 x 15,6 mm
- Tamaño de imagen: 6000 x 4000 px

- Mediciones.

Como complemento a la captura de fotos se llevan a cabo una serie de mediciones sobre el terreno, con el fin de poder situar los diferentes elementos y poder escalar el modelo en la fase de procesado. Para el presente trabajo, y desde un punto de vista más arquitectónico, se hace muy necesario el escalado del modelo, pues una de las finalidades principales del mismo es la posibilidad de tomar medidas reales sobre él. De esta forma se prima este punto sobre la situación en un sistema de coordenadas global, que no se considera imprescindible en este caso, situándose de forma relativa en el espacio. El siguiente esquema muestra las medidas recogidas en campo:

Figura 2.2.1. Mediciones.

- Resultados.

Como resultado de los trabajos en campo, y como se mostraba en la tabla 2.2.1, se han obtenido un total de 268 fotografías de la zona a levantar: 55 de la Domus, 86 de las Viviendas, 72 de las Termas, 27 del Arco, 28 del Foro. Sin tener en cuenta tomas previas, y algunas fotografías más del entorno, exclusivamente las capturas a procesar.

Además, se han recogido una serie de medidas para poder escalar el modelo en la fase de procesado.

2.3. PROCESADO FOTOGRAMÉTRICO

- Selección de fotografías.

Una vez en oficina, se procede a llevar a cabo un análisis, selección y organización de las fotografías tomadas en campo. Se separan en carpetas en base a los anillos establecidos en la planificación, ciñéndonos a los mismos para el procesado y dejando el resto de fotografías a parte por si son necesarias más adelante para situar o analizar elementos del entorno.

Incorporaremos al proyecto de procesado todas las fotos de cada grupo y será ya dentro del programa donde se determinará si hay capturas que deben ser omitidas por ser redundantes, tener elementos que modifiquen la foto (como brillos u otras irregularidades) o salirse fuera de los anillos establecidos. De esta forma, se procederá a orientar solamente las fotos que se estipulen como buenas. A continuación se analizan el número de fotografías totales por anillo y cuantas de ellas han sido orientadas:

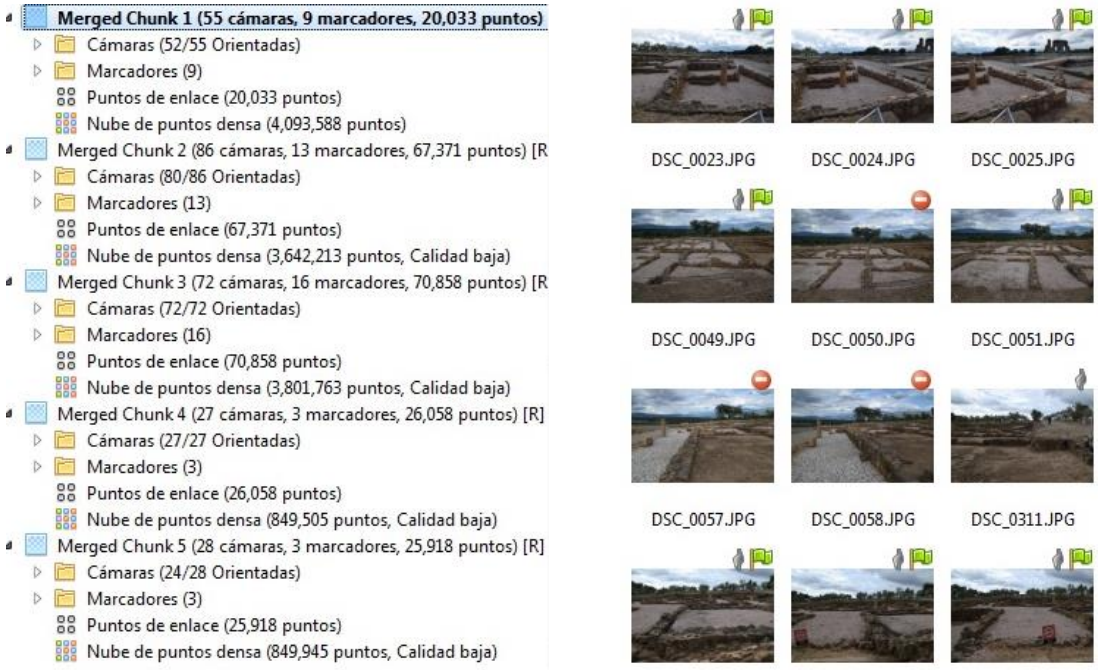


Figura 2.3.1. Selección y organización de fotografías.

Anillo	Zona	Nº Fotos Seleccionadas	Nº Fotos Orientadas
Anillo 1	Domus	55	52
Anillo 2	Viviendas	86	80
Anillo 3	Termas	72	72
Anillo 4	Arco	27	27
Anillo 5	Foro	28	24

Tabla 2.3.1. Fotografías seleccionadas/orientadas.

- Selección del software.

Existe a día de hoy multitud de software para procesado fotogramétrico. El creciente uso de drones está favoreciendo también el crecimiento de este tipo de software. Tenemos la posibilidad de utilizar programas de bajo coste, la mayoría de ellos con funcionamiento online, con la ventaja de tener un precio más bajo y el procesado en la nube, y por tanto desvinculado de las capacidades de nuestro esquipo. La desventaja es que suele ir dirigido a un público menos especializado, con lo cual aspectos como la precisión o la densidad de puntos se ven disminuidos. Se trata de programas como Autodesk 123D Catch, Microsoft Photosynth o Autodesk Recap.

Para un trabajo más profesional conviene recurrir a software de escritorio, entre los que destacan también marcas con más recorrido y otras de más reciente aparición. Para el presente trabajo nos centramos en este tipo de software, con el que tenemos más control sobre los procesos y sobre los parámetros, aunque solo hasta cierto punto, porque suelen ser programas bastante cerrados en cuanto a los procesos que llevan a cabo. Más concretamente se analiza y compara software con un cierto recorrido, como son Agisoft PhotoScan, Pix4D o PhotoModeler Scanner. En la siguiente tabla se puede ver un resumen de esta comparativa:

	Agisoft PhotoScan	**Pix4D**	**PhotoModeler Scanner**
Sistema operativo	Windows, Linux, MAC	Windows, Linux, MAC	Windows
Multiplataforma	No	Escritorio, web, móvil	No
Tipo fotogrametría	Aérea, UAV, Rango cercano	Aérea, UAV, Rango cercano	Rango cercano
Modelado automático	Sí	Sí	Sí
Creación	2010	2011	2008
Licencia	180-3500$ 60-550 $ Educación		3500$
Versión de prueba	Completa. 30 días	15 días	15 días

Tabla 2.3.2. Comparativa software fotogramétrico.

Finalmente se ha elegido como software de procesado Agisoft PhotoScan, principalmente por tener conocimientos previos de trabajo con él, por disponibilidad de licencias, existe una versión de prueba bastante larga y completa, y en caso de querer adquirirlo hay múltiples opciones según el precio que queramos pagar. Pix4D se está convirtiendo en una alternativa bastante competitiva, con sus versiones web y de móvil, y la posibilidad de manipulación dentro del programa, aunque la calidad del procesado sigue siendo un poco mejor la de PhotoScan.

- Procesado por fases.

Una vez se tienen los datos de campo listos y organizados, y se ha decidido el software a utilizar, se comienza con el procesado. Este atenderá a los distintos anillos establecidos para la captura de datos. Por tanto, se realizaran los pasos correspondientes, que se verán en el siguiente apartado "Obtención de la nube de puntos", para cada uno de los mencionados anillos.

Se ha encontrado que el alineamiento de fotografías no era óptimo en todos los casos, y se ha procedido a subdividir a su vez cada anillo en distintas partes, para que la detección de puntos fuera correcta, y posteriormente se han alineado las partes entre sí para obtener el anillo completo. De esta forma, se procesan las fotos para cada una de las caras de la manzana correspondiente a cada anillo, y se alinean después automáticamente o bien, en algún caso que presentaba más problemas, de forma manual, dando puntos homólogos en cada una de las nubes.

De la misma forma se realiza el alineamiento de los distintos anillos, hasta obtener la nube de puntos al completo.

Recordar que se han procesado los datos correspondientes al conjunto más significativo del yacimiento: domus, viviendas, termas, arco y foro. Existen otros elementos desconectados que podrían tratarse por parte. La ventaja de aplicar fotogrametría terrestre es que se pueden ir incorporando nuevas partes sin necesidad de hacer un vuelo de dron cada vez que se produzca un nuevo descubrimiento.

- Obtención de la nube de puntos.

En general, el flujo de trabajo de PhotoScan es el siguiente: añadir fotos (importación de datos), alinear fotos (generación de una nube de puntos no densa, orientación mediante el algoritmo SIFT y el ajuste de haces), construir nube de puntos densa, construir malla (mediante triangulación 3D), y construir textura (mapeo de la textura).

Ventajas de este software, la potencia en la búsqueda de puntos homólogos en las imágenes, y en el calibrado de la cámara, incluso no teniendo datos de ello.

En cuanto al algoritmo SIFT, se trata de un método para extraer puntos característicos invariantes y distintivos de una imagen que pueden ser usados para mejorar la correspondencia entre dos vistas diferentes de un objeto o una escena. Los puntos característicos obtenidos por SIFT son invariantes a escala y rotación de la imagen, y son mostrados para proporcionar un matching robusto a pesar de que exista un rango amplio de distorsiones afines, cambios en la vista 3D, adición de ruido a la escena y cambios en la iluminación. Se puede dividir la estructura de SIFT en cuatro módulos (Flores & Braun, 2011)(Aracena-Pizarro & Daneri-Alvarado, 2013):

- *Detección de escala-espacio* (DEE): La primera fase de computación busca en todas las escalas y localizaciones de la imagen. Se implementa eficientemente al emplear la función de diferencias gaussianas para identificar los puntos de interés que son invariantes a escala y orientación.

- *La localización del "punto clave"* (LK): En cada localización candidata se adecua un modelo cuadrático detallado para determinar la localización y la escala. Los puntos claves se seleccionan basándose en su estabilidad.
- *Asignación de orientación* (AO): Una o más orientaciones se asignan a cada localización de los puntos claves, basándose en la dirección del gradiente de la imagen. Todas las futuras operaciones se ejecutan con los datos de la imagen que ha sido transformada de acuerdo con la orientación, escala y localización asignada para cada característica, de este modo se proporciona invariancia a estas transformaciones.
- *Descriptor del "punto clave" (*DK): Los gradientes de la imagen son medidos en la escala seleccionada en la región alrededor de cada punto clave. Estos son transformados a una representación que permite, para niveles significativos, distorsión de la forma local y cambios en la iluminación.

Para el procesamiento de datos con PhotoScan, el conjunto de datos de imagen se redujo a 268 imágenes, cubriendo todas las manzanas de la parte principal del yacimiento.

El ordenador utilizado para este tratamiento de datos: HP Pavilion dv6-3330ss, Intel Core i5 CPU M460 2,53 GHz, procesador de 64 bits, tarjeta gráfica ATI Mobility Radeon HD 5650 1GB dedicada, 4 GB RAM y Windows 7 como sistema operativo.

Los parámetros para el flujo de trabajo: alinear fotos (alta precisión con 400000 puntos clave por foto), construir nube densa (alta calidad, filtrado de profundidad agresiva), construir malla (tipo de superficie arbitraria, nube de puntos densa como fuente de datos, interpolación habilitada), y construir textura (modo de asignación genérica, textura de todas las cámaras, modo de fusión mosaico)

En la fase de alineación u orientación, se obtiene una nube de puntos no densa de 210238 puntos. En la fase de construcción de nube densa de puntos, una con 13,2 millones de puntos.

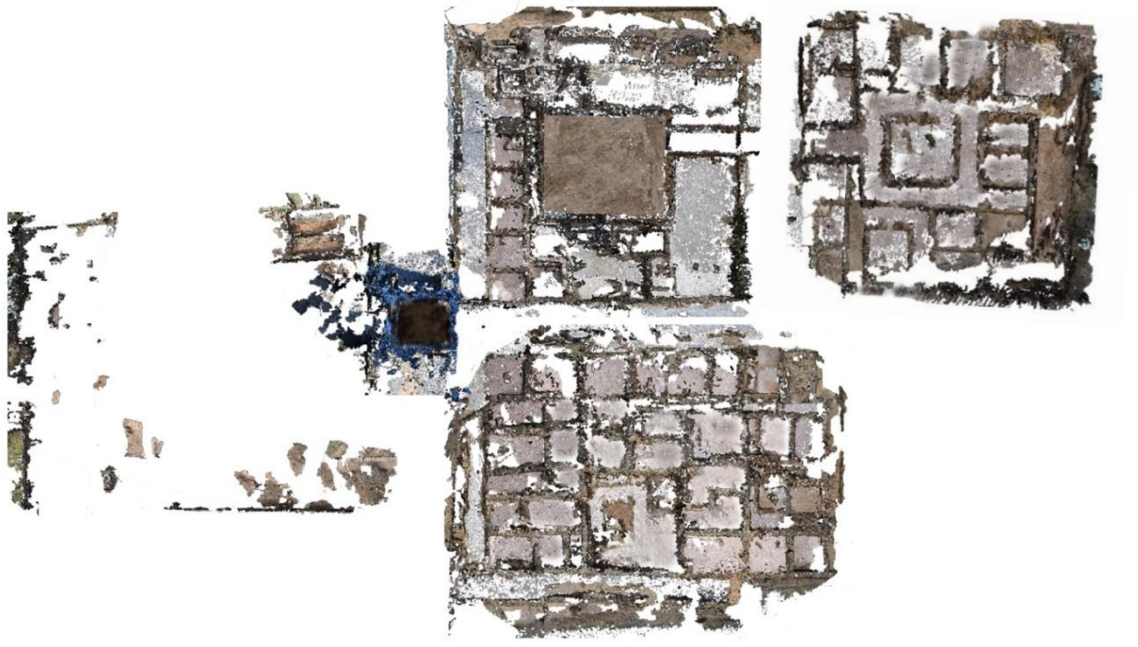


Figura 2.3.2. Nube de puntos densa.

- Puntos de referencia para el escalado.

Se sitúan una serie de puntos, y se establecen distancias de referencia a partir de las mediciones tomadas en campo. Este proceso sirve para tener el modelo a escala y poder tomar medidas sobre él.

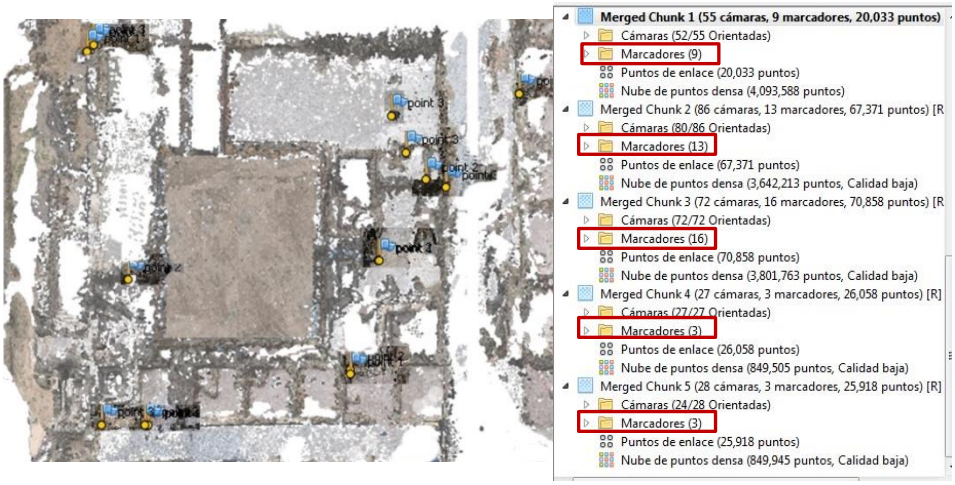


Figura 2.3.3. Puntos de referencia.

- Filtrado de la nube.

Como se ha dicho en el punto anterior, en la fase de construcción de nube densa de puntos, se obtiene una nube con 13,2 millones de puntos. Se procede a una operación de filtrado de la nube para su incorporación a la aplicación/base de datos que se desarrolla, para que el coste de procesado sea menor.

Como método de subdivisión se emplea una estructura octree, de manera que se particiona el espacio tridimensional dividiéndolo recursivamente en ocho octantes. Este proceso se lleva a cabo mediante el software CloudCompare.

- Vectorización.

Se procede a la vectorización de las distintas capas, a partir de secciones obtenidas y con ayuda de AutoCAD. Estas capas se guardan en formato .dxf, para más tarde ser tratadas en QGIS y transformadas a archivos .shp.

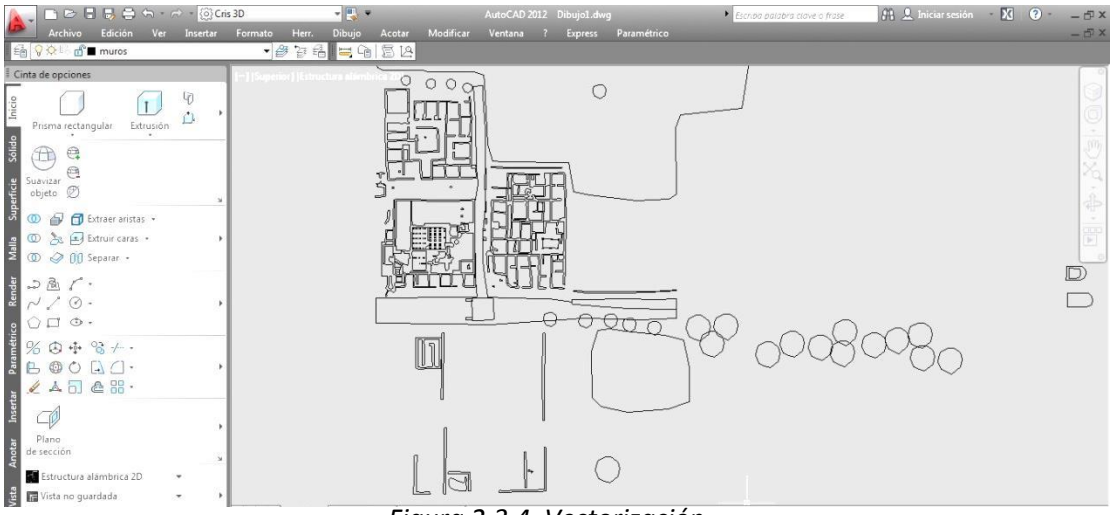


Figura 2.3.4. Vectorización.

2.4. PRODUCTOS OBTENIDOS A PARTIR DEL PROCESADO FOTOGRAMÉTRICO

Por un lado tenemos los productos que se obtienen de una forma directa, como es la nube de puntos, o las capas vectorizadas a partir de dicha nube. Pero también podemos obtener otros productos de interés, como pueden ser alzados y secciones de los elementos del yacimiento, muy interesante desde el punto de vista arquitectónico, sobre todo para el arco. También, a partir de la vectorización y de la información obtenida mediante la nube de puntos, y con ayuda de las investigaciones existentes e indagaciones propias acerca de cómo eran este tipo de asentamientos, se procede a hacer un levantamiento 3D, a partir del 2D, mediante el programa Google SketchUp 8. Se considera interesante desde el punto de vista analítico del yacimiento, y también por las posibilidades divulgativas.

Figura 2.4.1. Nube de puntos y vectorización.

A continuación se incluyen alzados y secciones extraídas de la nube de puntos con PhotoScan. El flujo de trabajo previo había sido: añadir fotos, alinear fotos, construir nube de puntos densa, construir malla, y construir textura. Adicionalmente se procede a: crear ortomosaico, y exportar ortomosaico a imagen .jpeg.

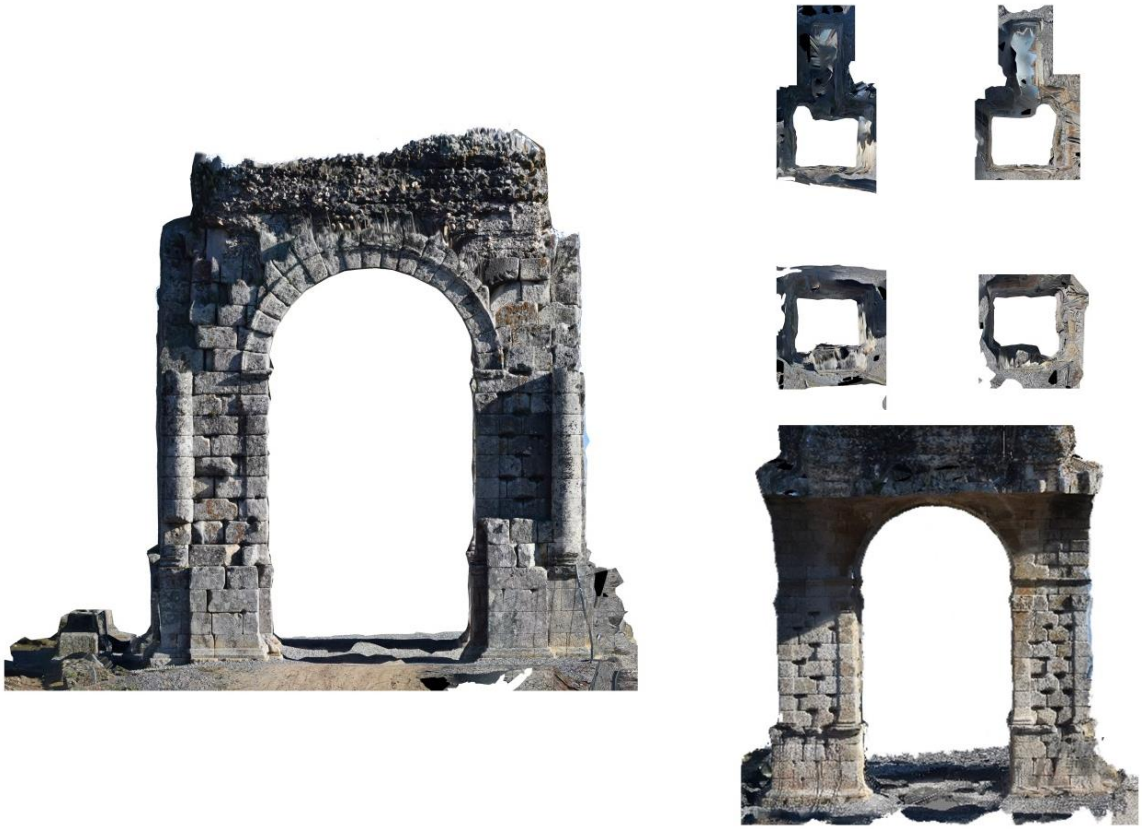


Figura 2.4.2. A la izquierda Alzado SW, a la derecha Secciones horizontal y vertical del Arco.

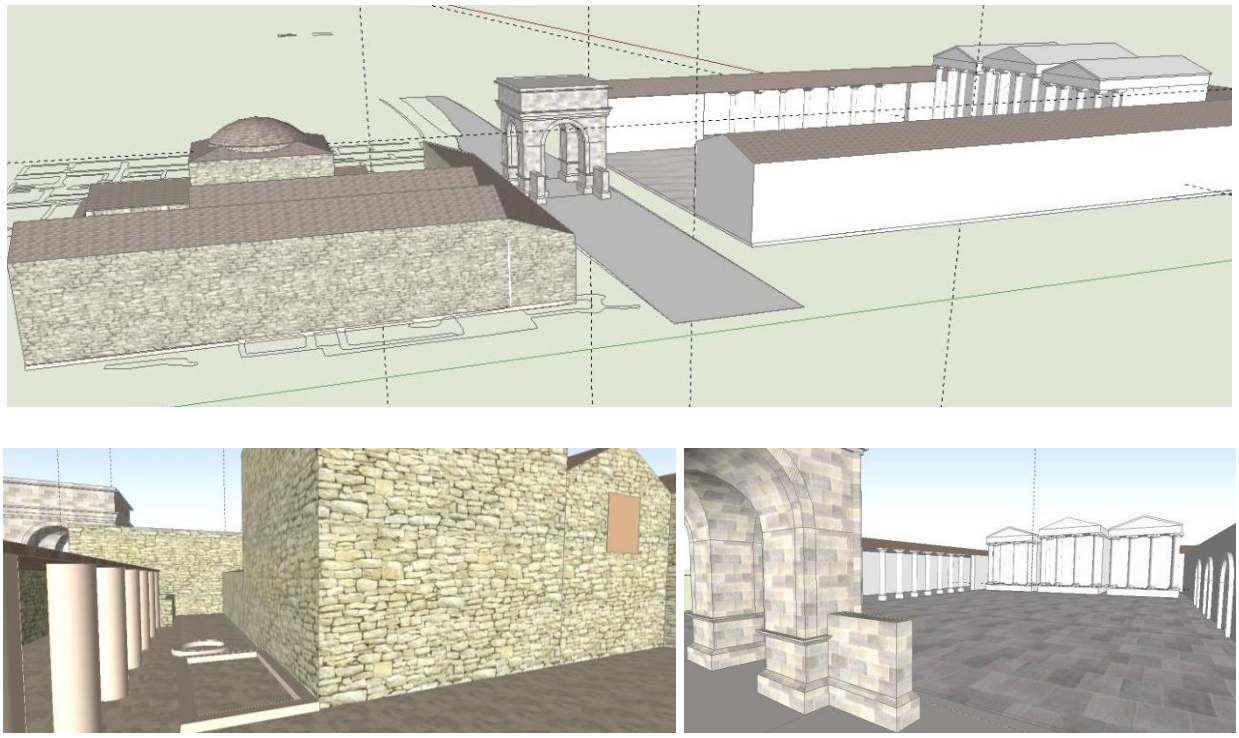


Figura 2.4.3. Posible reconstrucción 3D. Vistas desde el interior de las termas y el foro.

3. DESARROLLO DE UNA APLICACIÓN SIG MEDIANTE LIBRERÍAS QGIS Y VTK. INCLUSIÓN DE LOS DATOS OBTENIDOS

3.1. ELECCIÓN DEL ENTORNO DE DESARROLLO

- Elección del lenguaje de programación.

Debido a que el software SIG existente está más orientado a la documentación 2D o a la generación de documentos 3D vinculados al terreno, pero no tanto a una visualización paralela y conectada de la documentación 2D y 3D, por ejemplo, desde un punto de vista arquitectónico, relacionando cada capa con su nube de puntos, alzados o secciones, se plantea el desarrollo de una aplicación propia a partir de librerías externas, que permita lo dicho anteriormente. Se han encontrado ideas similares a las que aquí se plantean en el proyecto "The mayaarch3d project: A 3D webgis for analyzing ancient architecture and landscapes" (Von Schwerin, Richards-Rissetto, Remondino, Agugiaro, & Girardi, 2013).

Entre los lenguajes de programación que se manejan, se opta por C++ ("Programación en C++," n.d.) por la existencia de multitud de librerías en el ámbito en el que se quiere desarrollar la aplicación. Así se pueden aprovechar funciones ya elaboradas, para incorporarlas al programa que se pretende crear.

El objetivo es una aplicación que funcione a modo de base de datos, recopilando todos los datos obtenidos y permitiendo su consulta, de forma que cualquier persona pueda disponer de la información, bien sea a modo de consulta técnica o con fines más divulgativos, sin necesidad de tener otros programas instalados. De este modo se pretende que el programa sea a su vez un visor de datos vectoriales y de toda su información asociada, así como un visor de nubes de puntos fotogramétricas. Con la ventaja no solo de no necesitar software adicional, sino que además en una misma aplicación se conjuntan datos vectoriales y de nubes de puntos, conectados entre sí.

- Elección del software.

Como software para el desarrollo se ha escogido Qt, por tratarse de un programa bastante sencillo de manejar e intuitivo, con manuales en línea ("Qt 4.8," n.d.). Además cuenta con licencia gratuita siempre que no se tengan fines comerciales. En concreto la versión Qt 4.8.4 vs2010 para Windows 7 64 bits, y Qt Creator 2.6.0.

Como software adicional se ha empleado:

- Windows SDK para Windows 7 y .NET Framework 3.5 SP1 (para el Debugger: CDB Engine).
- Visual Studio 2010 Express Edition (para el Compilador: Microsoft Visual C++ Compiler 10.0 x86).
- CMake-3.5.2-win32-x86 (build system).
- QGIS Brighton 2.6 32 bits con las herramientas para desarrolladores, para que se instalen todas las librerías que se emplearán en este proyecto.
- La librería PCL 1.6.0 que incluye como elementos de terceros las librerías de VTK, que son las que finalmente se han utilizado, o directamente las librerías de VTK 5.8.0 con soporte para Qt.

La siguiente tabla muestra un resumen del software empleado:

Software	**Uso**
Qt 4.8.4	Librerías generales
Qt Creator 2.6.0	Desarrollo de la aplicación
Windows SDK	Debugger
Visual Studio 2010	Compiler
CMake-3.5.2	Build system
QGIS Brighton 2.6	Librerías SIG
VTK 5.8.0	Librerías Nubes de puntos

Tabla 3.1.1. Software empleado para el desarrollo de la aplicación.

- Librerías.

Una librería, o biblioteca, es un conjunto de clases y funciones, codificadas en un lenguaje de programación (en este caso C++), que pueden ser llamadas y utilizadas por otro programa.

Este proyecto se ha apoyado en las librerías de QGIS relativas a funcionalidades SIG ("Documentación para QGIS 2.6," n.d.)(Sutton et al., n.d.), y en las de VTK para manejar nubes de puntos ("VTK/Examples," n.d.), ambas de código abierto, para completar las funcionalidades del programa a desarrollar. Por tanto, se direccionan dichas librerías en el proyecto, para más tarde ser llamadas cuando son necesarias.

En la siguiente tabla se muestra una relación de las librerías utilizadas y de las funciones que se han empleado en el proyecto:

Tabla 3.1.2. Librerías utilizadas en el proyecto.

Librerías QGIS	**Funciones**	**Uso**
qgis_core qgis_gui	qgsapplication qgsproviderregistry	Iniciar la aplicación
	qgssinglesymbolrendererv2 qgsmaplayerregistry qgsvectorlayer	Crear capas
	qgsmapcanvas	Crear el visor
	qgslayertreemapcanvasbridge qgsproject qgslayertreemodel qgslayertreeview	Crear el árbol de capas
	qgsmaptool qgsmaptoolpan qgsmaptoolzoom	Crear herramientas para manipular el mapa

Librerías VTK	**Funciones**	**Uso**
QVTK vtkalglib vtkCharts vtkCommon vtkDICOMParser vtkexpat vtkFiltering vtkfreetype	vtkSmartPointer vtkSimplePointsReader vtkPolyDataMapper vtkActor vtkProperty vtkRenderer vtkRenderWindow vtkRenderWindowInteractor	Abrir nube de puntos .xyz
vtkftgl vtkGenericFiltering vtkGeovis vtkGraphics vtkhdf5 vtkHybrid	vtkVersion vtkSmartPointer vtkPolyData vtkPoints vtkCellArray vtkXMLPolyDataWriter	Abrir texto plano
vtkInfovis vtkIO vtkmetaio vtkNetCDF vtkproj4 vtkRendering vtksys vtkverdict	vtkVersion vtkSmartPointer vtkPoints vtkPolyData vtkPointData vtkCellArray vtkUnsignedCharArray vtkVertexGlyphFilter	Colorear la nube de puntos
vtkViews vtkVolumeRendering vtkWidgets vtkzlib	QVTKWidget	Crear el visor
Librerías QT	**Funciones**	**Uso**
Qt3Support QtCore4 QtGui4 qtmain QtNetwork QtSql QtSvg QtXml	QApplication	Crear la aplicación
	QWidget	Crear la ventana
	QVBoxLayout QHBoxLayout	Agrupar los elementos dentro de la ventana
	QMenuBar QMenu	Crear barra de menú
	QToolBar	Crear barra de herramientas
	QAction	Crear una acción
	QIcon	Crear un icono
	QFileDialog QFileInfo QFile QRegExp	Abrir un archivo
	QDockWidget	Crear una leyenda
	QTableWidget QTableWidgetItem	Crear una tabla
	QString QList QStringList QTextEdit QLineEdit	Manejar texto
	QInputDialog	Introducir texto en pantalla

3.2. DESARROLLO DE LA APLICACIÓN

- Definición del proyecto.

El proyecto se compone de varios archivos:

- Un archivo de texto CMakeLists que contiene la configuración para el programa. Aquí es donde se incluyen los enlaces a las librerías utilizadas:

```
FIND_PACKAGE(VTK REQUIRED)
INCLUDE(${VTK_USE_FILE})
TARGET_LINK_LIBRARIES(VisorShpXYZ ${VTK_LIBRARIES} QVTK)

FIND_PACKAGE(Qt4 REQUIRED)
INCLUDE(${QT_USE_FILE})
TARGET_LINK_LIBRARIES(VisorShpXYZ ${QT_LIBRARIES})

FIND_PACKAGE(QGIS REQUIRED)
TARGET_LINK_LIBRARIES(VisorShpXYZ
${QGIS_CORE_LIBRARY}
${QGIS_GUI_LIBRARY})
```

  El resto del código se incluye en el Anexo I, junto con el código completo de la aplicación.
- El archivo de cabecera visorshpxyz.h que contiene la definición de las clases creadas.
- El archivo visorshpxyz.cpp que contiene el código de desarrollo de la aplicación (ver Anexo I).
- El archivo visorshpxyz.qrc. Es un archivo de recursos, que incluye las imágenes de los iconos empleados en el programa para poder enlazarlos más fácilmente dentro del código.

- Funcionalidades del programa.

A continuación se va a hacer una descripción pormenorizada de la interfaz gráfica y de las funciones del programa. Como ya se ha mencionado el código completo se adjunta en el Anexo I. Muchas dudas acerca de la programación se han resuelto gracias a la página (“Stack Overflow,” n.d.).

Arriba se encuentran una barra de menú y unas barras de herramientas que corresponden cada una con el espacio del mapa o con la nube de puntos.

A la izquierda una barra de herramientas con una serie de botones que enlazan con la nube de puntos correspondiente a cada capa de la que existe dicha información. También el árbol de capas.

Y en el espacio central el mapa y el espacio para visualizar la nube de puntos.

A continuación, en la Figura 3.2.1, se incluye una captura de la interfaz de la aplicación y un diagrama explicativo de sus partes.

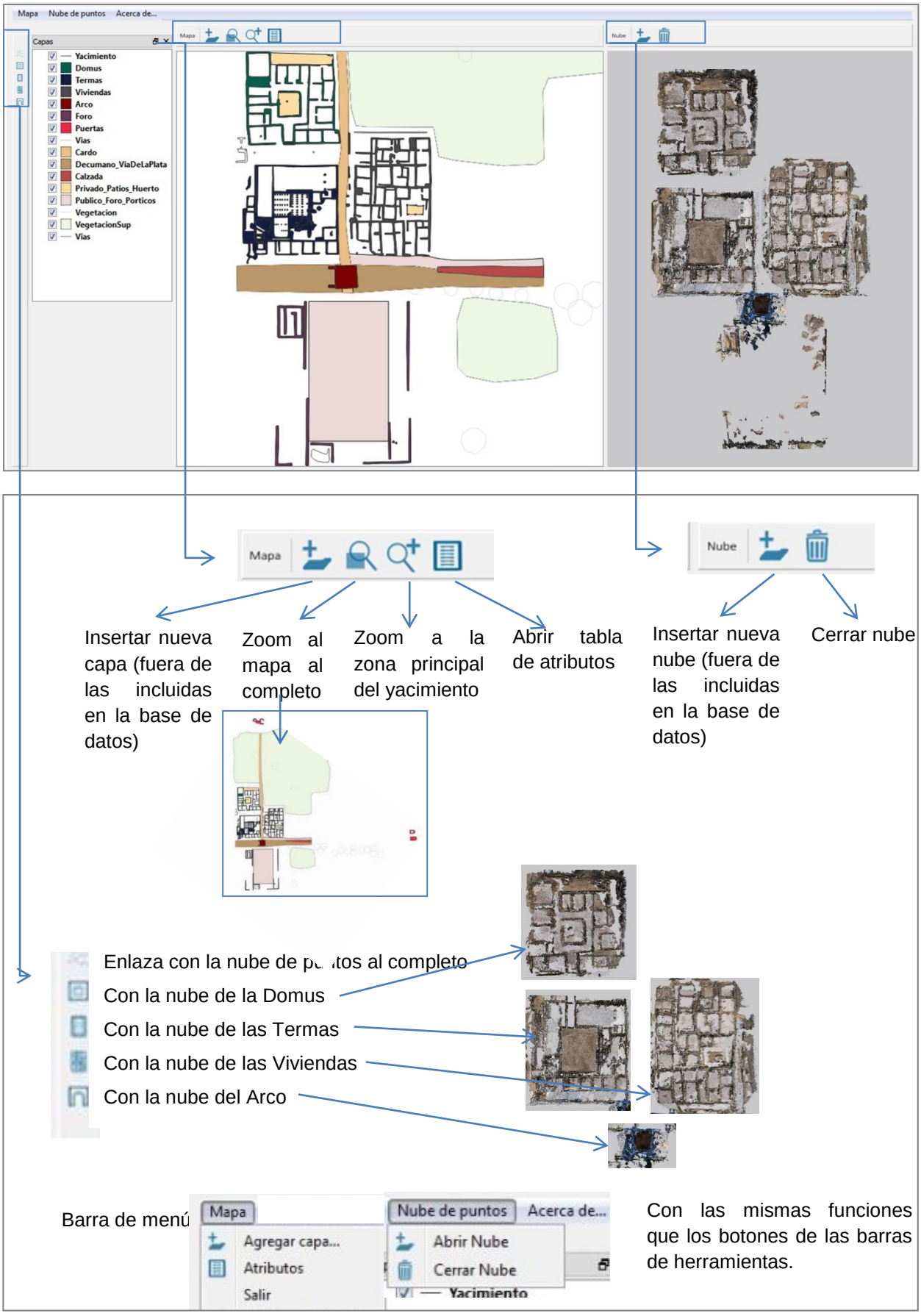


Figura 3.2.1. Captura de la aplicación y diagrama explicativo.

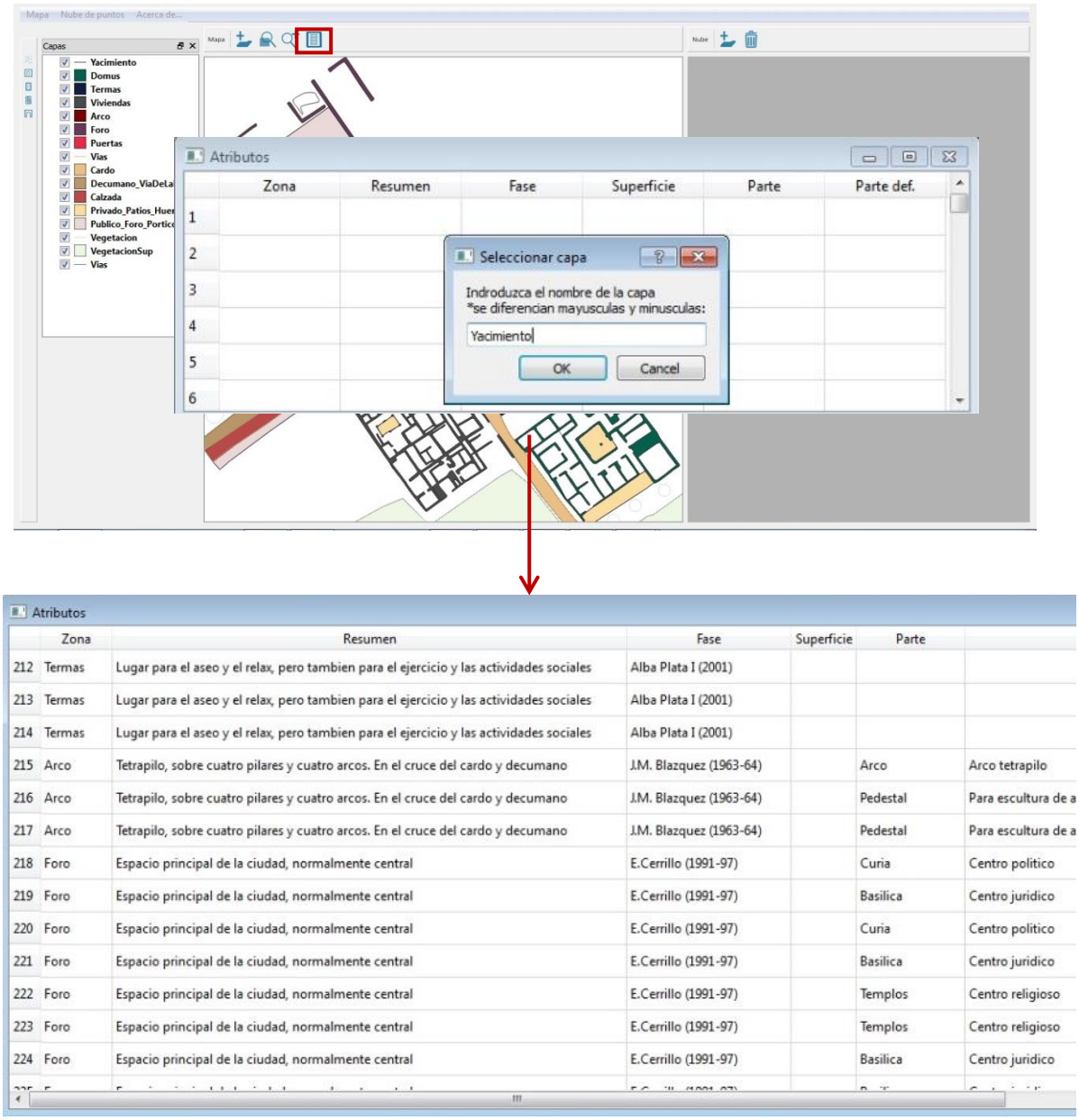


Figura 3.2.2. Apertura de atributos.

Tanto en la barra de herramientas como en la barra de menú referidas al mapa, se dispone de un botón que permite la apertura de la tabla de atributos correspondiente a cada capa. Al pulsar sobre el mismo aparece un cuadro de diálogo que solicita introducir el nombre de la capa para la que queremos abrir la tabla. Al pulsar “Ok” aparece una nueva ventana en la que se nos muestra toda la información disponible.

Esto completa la visualización de la documentación, que como se verá en el apartado siguiente se estructura como una base de datos, en la que se recogen los datos de nubes de puntos, vectoriales y atributos.

La organización de la información, junto con la creación de la aplicación detallada en este apartado, permiten la consulta de la documentación obtenida mediante los trabajos de campo y el procesado.

A continuación un ejemplo de la visualización conjunta de los datos referidos al arco:

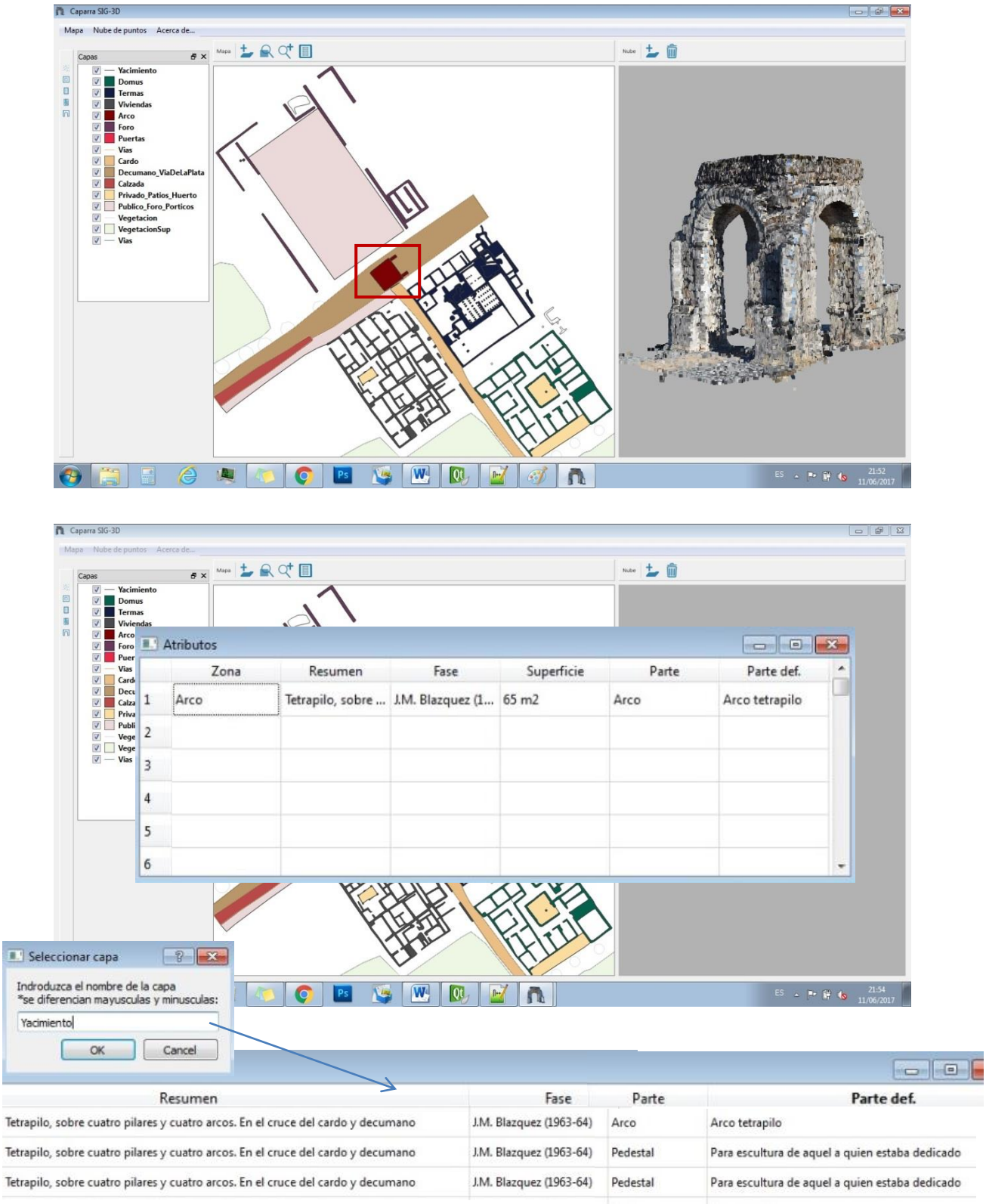


Figura 3.2.3. Visualización de los datos referidos al arco mediante la aplicación desarrollada.

3.3. GENERACIÓN DE LA BASE DE DATOS

Vectorizados todos los datos obtenidos, se reúnen una serie de archivos en formato .dxf, que corresponden con las distintas capas en las que se estructura la información, referidas a las distintas partes del yacimiento. Se va a emplear como software para tratamiento de estos datos, el programa QGIS versión Brighton 2.6, por tratarse de software libre, además de que sus librerías ya se han empleado para el desarrollo de la aplicación, por lo que ya está instalado y configurado. Se han valorado otras opciones, como ArcGIS for Desktop 10.3, que ofrece más funcionalidades, aunque se trata de un software de pago, pero finalmente se ha descartado pues se ha considerado que QGIS proporcionaba las herramientas necesarias para los fines que se buscaban. En cuanto a la versión, se opta por una ya consolidada y no la última pues proporciona mayor estabilidad, sobre todo en el ámbito de desarrollo con las librerías mencionadas.

Figura 3.3.1. Captura inicio QGIS Brighton.

Por tanto, se cargan todas las capas y se procede a la revisión y rectificación de sus entidades, así como a la creación de los atributos correspondientes. Se guarda en formato .shp. También se configuran las opciones de estilo y etiquetas, configuración que se incluirá también en la aplicación.

Y por último se georreferencian los datos vectorizados mediante la ayuda de cartografía existente, que viene dada en el sistema de referencia WGS 84, enlazada como wms desde servidores del IGN.

1. Se comienza con la apertura de las capas:

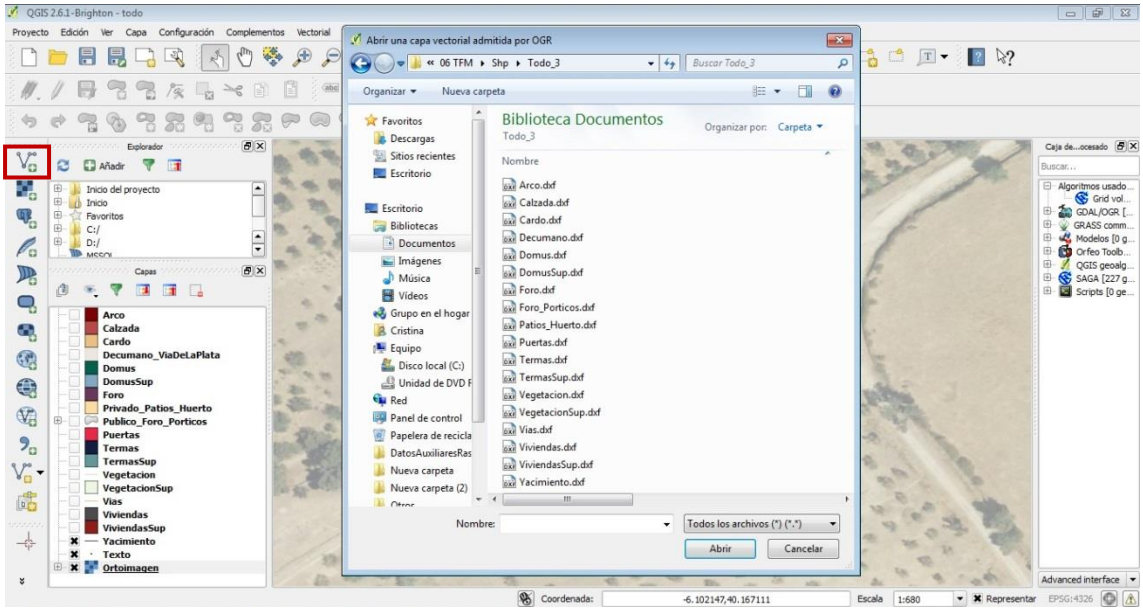


Figura 3.3.2. Apertura de capa vectorial en formato .dxf.

2. Se corrigen defectos en los archivos vectoriales, como entidades superpuestas, elementos que no existen… Activamos la edición para la capa sobre la que queremos trabajar y añadimos, movemos, simplificamos…objetos espaciales (Figura 3.3.3).

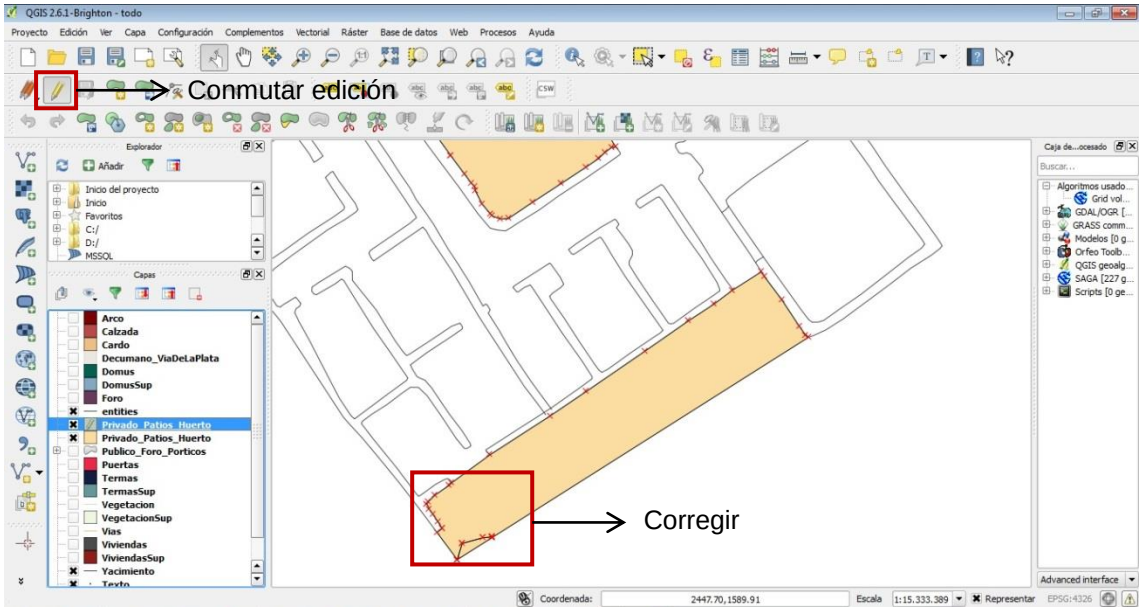


Figura 3.3.3. Revisión y rectificación de entidades.

3. Se establecen el color de capas, las etiquetas a mostrar, etc. de forma que se visualice la información de la mejor forma posible. Se traspasarán estas opciones a la propia aplicación. Este no es un paso imprescindible en la configuración de los datos, pero si se considera necesario para una visualización más cómoda.

El ajuste de estilo y etiquetas se lleva a cabo en la ventana “Propiedades de capa”. En el botón derecho, sobre la capa a modificar, Propiedades. Podemos cambiar las características y el color de cada símbolo, y podemos etiquetar la capa con uno de los campos de la tabla de atributos.

4. A continuación, se procede a la creación de los atributos correspondientes. Con la edición de cada capa activada, se van creando las columnas que se estiman necesarias, estableciendo sus opciones de configuración, si se trata de texto, números, etc. Y luego se van rellenando los datos correspondientes a cada entidad.

En la siguiente figura podemos ver como se crean seis campos diferentes: Zona (hace referencia a la parte del yacimiento a la que corresponde la entidad: domus, termas, viviendas, arco o foro), Resumen (breve descripción de la funcionalidad de cada zona), Fase (año de excavación y fase a la que corresponde), Superficie (m2 de superficie del elemento, en capas poligonales), Parte (cada elemento se subdivide a su vez en distintas partes con sus funcionalidades propias), Parte def. (breve descripción del uso de cada parte).

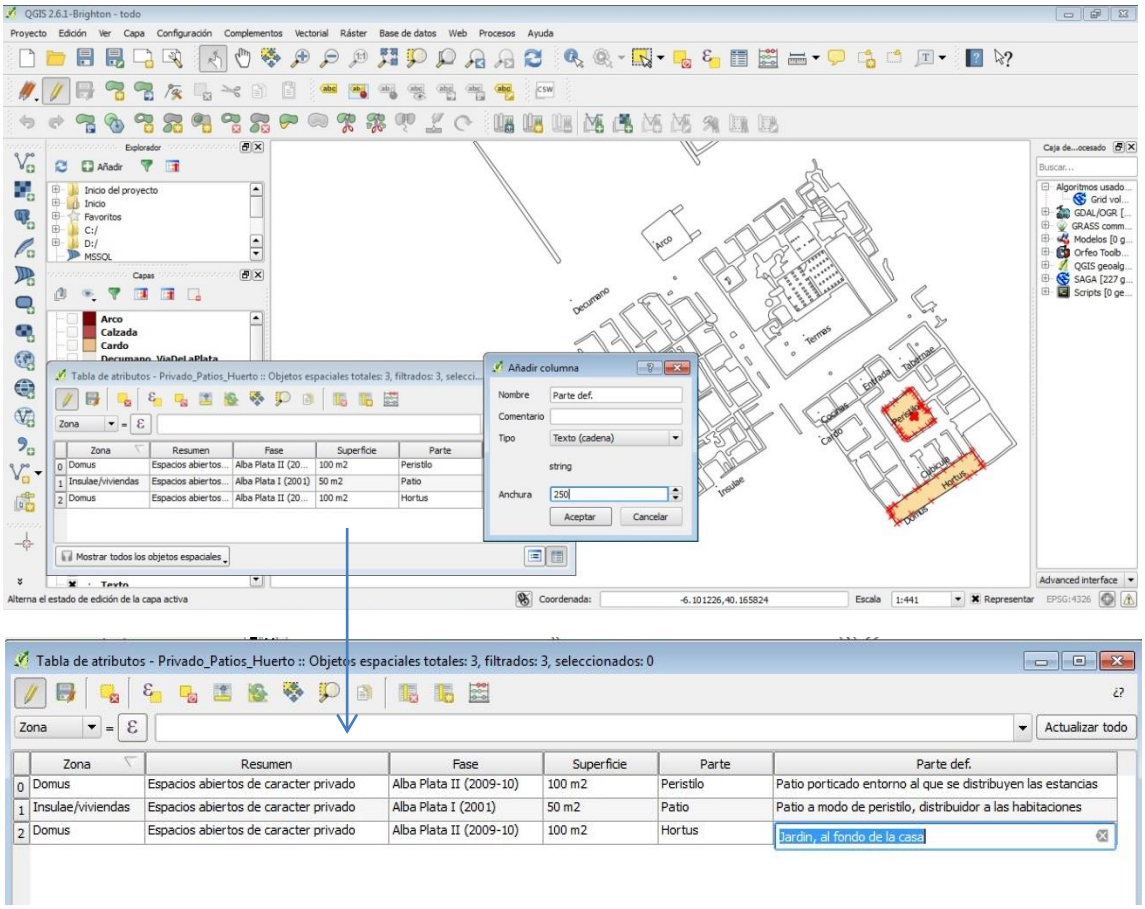


Figura 3.3.4. Creación de atributos.

5. Se busca un servicio wms adecuado y se inserta una ortofoto. En QGIS pulsamos sobre el botón Añadir capa desde un servidor WMS, conectamos con el enlace obtenido en el Geoportal IDEE y nos aparecen las capas disponibles.

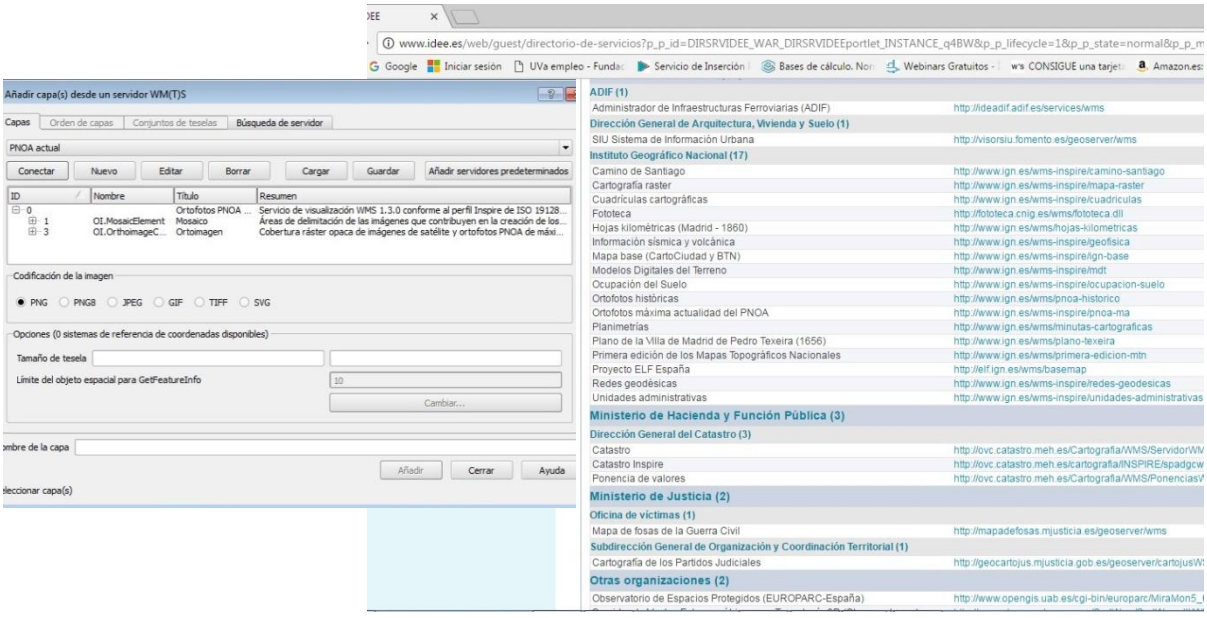


Figura 3.3.5. Búsqueda de servicio WMS en el Geoportal IDEE e incorporación a QGIS.

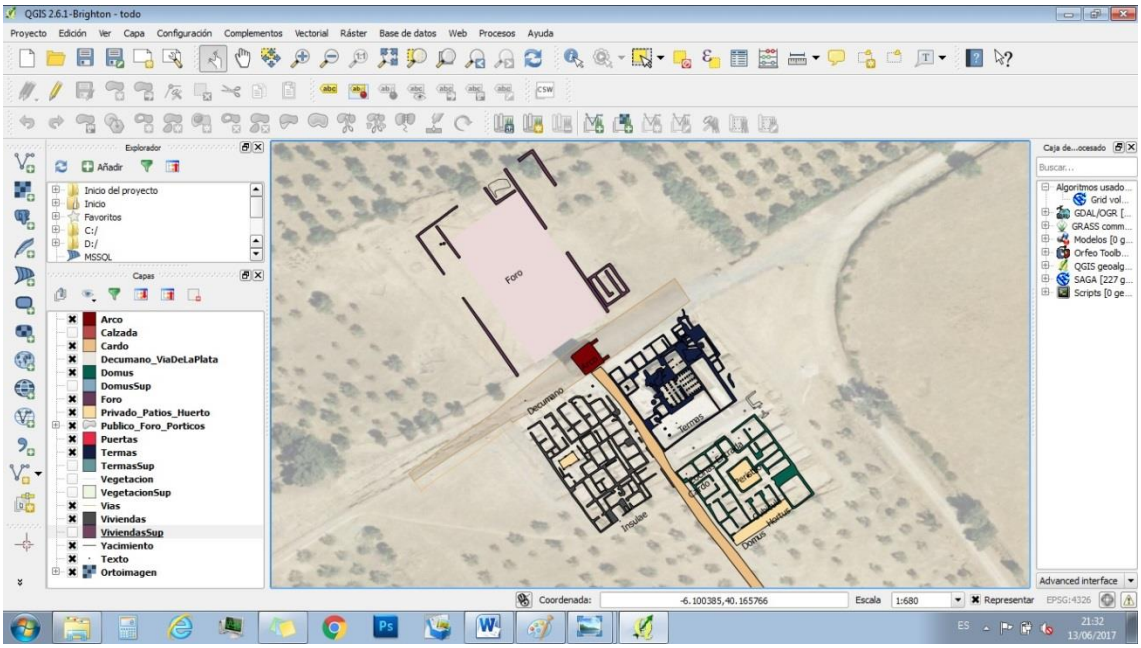


Figura 3.3.6. Vista de QGIS con todas las capas incorporadas y procesadas.

A partir de aquí se pueden elaborar los mapas que se estimen oportunos, en función de diferentes temáticas, mapas que se recogen en el apartado Productos obtenidos.

3.4 PRODUCTOS OBTENIDOS

Como ya se ha dicho, la organización de la información, junto con la creación de la aplicación, permiten la consulta de la documentación obtenida mediante los trabajos de campo y el procesado. Por tanto, como productos tenemos los datos vectoriales y atributos, y la aplicación para visualizarlos junto con las nubes de puntos resultantes del procesado fotogramétrico.

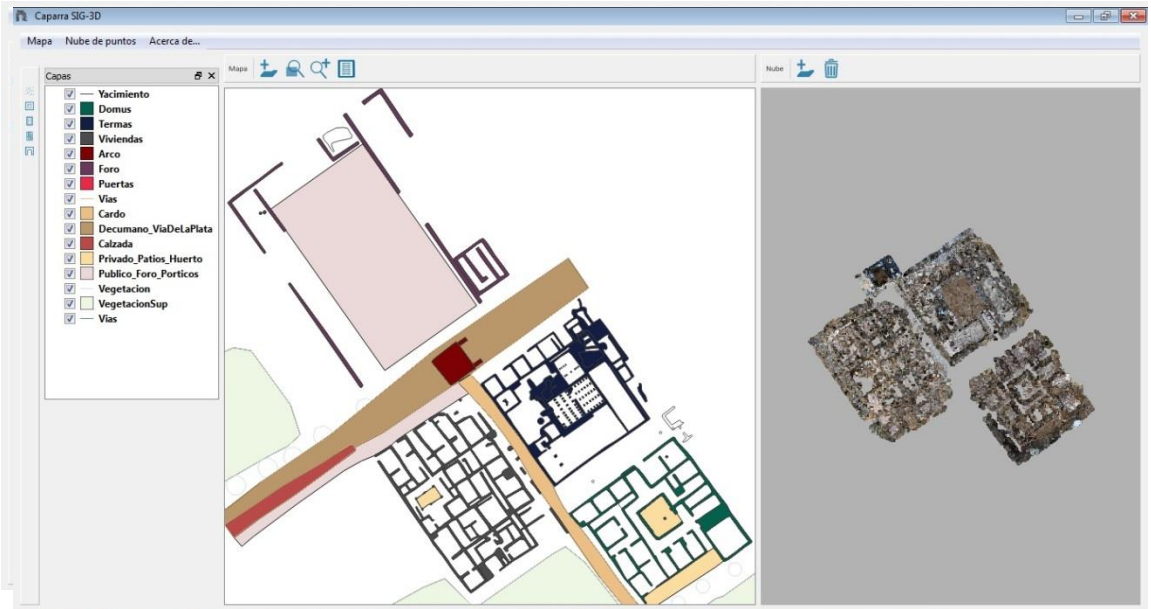


Figura 3.4.1. Vista de la aplicación, con los datos recogidos.

Adicionalmente, se pueden elaboran una serie de mapas temáticos, a partir de dichos datos, como los que se muestran a continuación.

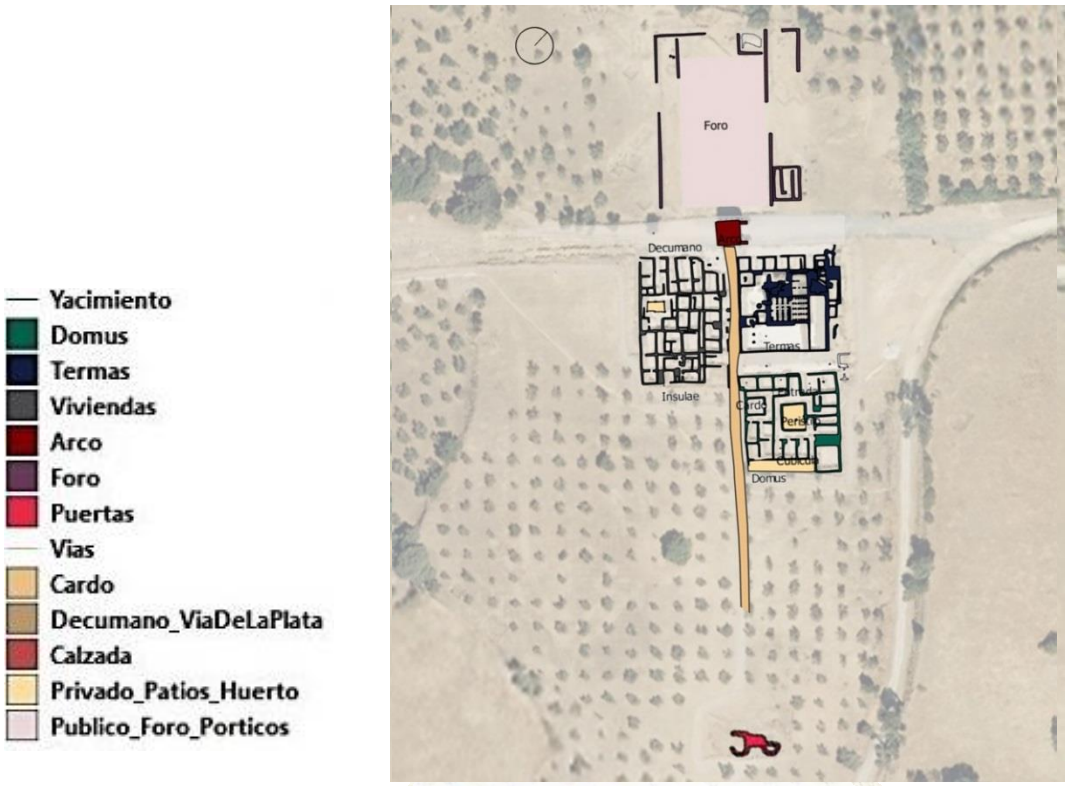


Figura 3.4.2. Mapa de la zona principal del yacimiento, con ortofoto de fondo.

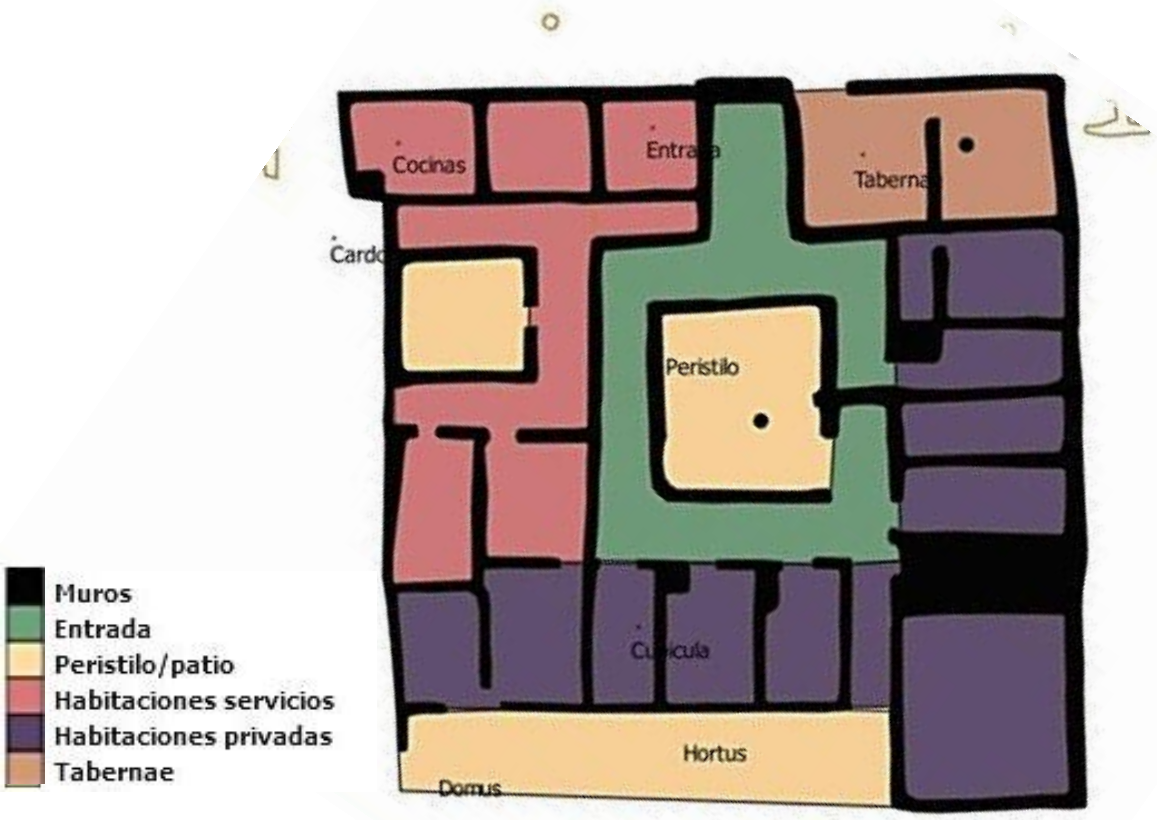


Figura 3.5.3. Mapa con la distribución de la domus.

4. CONCLUSIONES

- Discusión de resultados.

Recopilando todos los productos generados en los puntos anteriores, se han obtenido los siguientes resultados:

- Nube de puntos a escala y con color de la parte principal del yacimiento, con calidad suficiente para obtener un modelo medio, y alzados y secciones, pero no para obtener una ortofoto aérea óptima.

- Documentación 2D de plantas, en formato dxf y shp, georeferenciada.

- Modelo 3D de la interpretación del estado original del yacimiento, a partir de la información en planta y la nube de puntos.

- Organización de la documentación en shp, con sus atributos correspondientes.

- Mapas temáticos a partir de los datos en shp.

-Aplicación SIG de escritorio que combina la documentación 2D y 3D, creada con herramientas de código abierto y por tanto accesible a cualquier persona, y que ofrece la posibilidad de consultar todos los datos sin necesidad de tener instalados otros programas.

Se cumplen por tanto los objetivos que se buscaban, un levantamiento del yacimiento arqueológico por medios fotogramétricos, así como la obtención de otros productos geomáticos y su integración en una misma aplicación. Esto permite el estudio arqueológico y arquitectónico de las ruinas a un nivel superior que la mera documentación in situ, además de permitir que se comparta con el público en general con el fin de dar a conocer el monumento.

No obstante, se es consciente de las limitaciones en algunos aspectos debido al método de trabajo elegido, aunque se han primado ciertos aspectos a la hora de elegir, como se explicaba en el punto “Elección el método de trabajo”. Se consideran como ventajas de la fotogrametría frente al láser la mejor precisión en xy, la alta densidad de puntos, el sensor pancromático, el tiempo de toma de datos y el coste del equipo. Y de la fotogrametría terrestre sobre la aérea la mayor calidad en rango cercano y la disponibilidad del equipo. Combinando la fotogrametría con vehículos aéreos no tripulados se podría haber llegado a una precisión un poco superior, deseable en caso de querer obtener la ortofoto o un modelo sólido más preciso a parte del modelo de nube de puntos.

A la vista de los resultados obtenidos, se cree adecuada la elección ante la posibilidad de acercar esta forma de adquisición de datos a los trabajos en este tipo de yacimientos, pues se pueden ir tomando datos poco a poco. Si bien los resultados obtenidos pueden ser mejorables, se consideran aceptables, siempre teniendo en cuenta el fin que tienen. Y para un estudio métrico preciso de los hallazgos y una representación adecuada para su divulgación, creo son mejores que los que se podrían haber obtenido por medios tradicionales (mediciones y representación 2D y 3D “a mano”). Por tanto, la incorporación de técnicas geomáticas en estos entornos parece oportuna.

- Líneas futuras.

A continuación se exponen líneas de mejora del presente trabajo, que podrían continuar desarrollándose para obtener una documentación más completa.

En primer lugar se podría llevar a cabo una adquisición de datos por otros medios, con un vuelo de UAV con cámara sería suficiente para obtener además de lo aquí presente, una ortofoto precisa por ejemplo, o estudiar la posible mejora de la información obtenida.

También sería oportuno completar la información del yacimiento principal con los hallazgos más recientes, como son las puertas de la muralla o el anfiteatro, o con otros futuros.

La elaboración de otros mapas temáticos como los mostrados aquí atendiendo a las necesidades de divulgación o comunicación es un hecho que se da por supuesto, aquí se ha elaborado una muestra.

Respecto a la aplicación, podría mejorarse añadiendo algunas herramientas de edición, para llegar más allá del visor de datos. Y sobre todo, parece muy interesante la posibilidad de convertir la aplicación en una aplicación multiplataforma, es decir, que pudiera usarse también en móviles, sobre todo con el fin de llegar a la gente y en la línea que se vienen desarrollando las actuaciones en el entorno de la Vía de la Plata, de puesta en valor y conversión en productos turístico-cultuales. Es muy claro que hoy día todo aquello que puedes ver y consultar en tu móvil resulta si cabe aún más atractivo.

Por tanto, este trabajo se considera un inicio, un estudio de las posibilidades que las herramientas aquí utilizadas pueden tener en el ámbito del estudio y representación de elementos del patrimonio arqueológico y arquitectónico, o incluso de documentación para posibles actuaciones o intervenciones sobre él.

- Consideraciones finales.

Respecto a la elaboración de este trabajo, decir que, aunque la obtención de datos en campo ha sido más fácil y sobre todo más asequible pues se ha realizado con medios disponibles fácilmente para cualquiera, básicamente una cámara y un distanciómetro, el procesado de los datos procedentes de fotogrametría terrestre ha resultado probablemente más difícil que el procesado de datos obtenidos por un vuelo de dron en el que el alineamiento resulta más fácil, únicamente el coste computacional, o incluso que el procesado de láser. Ha supuesto más trabajo de revisión y de división del procesado, con el fin de que los alineamientos resultaran correctos. Por tanto, ha tenido más coste de trabajo en oficina. Si bien, uno de los objetivos era el comprobar el alcance de medios de bajo coste.

No obstante, aunque ha requerido más trabajo y dedicación, gracias a la formación recibida, no se han encontrado dificultades que no se pudieran solventar.

Si hay un punto en el que se ha tenido una dificultad mayor, probablemente por escasez de conocimientos en el ámbito, es en la parte informática, en el de desarrollo

de la aplicación. Sin embargo suponía un reto y también un interés personal, por la posibilidad de gestionar, sin recursos adicionales, la documentación obtenida.

Personalmente, este proyecto me ha aportado bastante en mi ámbito profesional y ha supuesto poner en práctica todos los conocimientos aprendidos, pero orientándolos a las posibilidades que yo puedo obtener de ellos. Me resulta interesante la posibilidad de elaborar Sistemas de Información en el ámbito Patrimonial, y no solo en el geográfico o urbanístico. Y también la posibilidad de utilizar medios como la fotogrametría o el láser para la documentación y levantamiento del patrimonio o la arquitectura existente, ya sea con fines meramente representativos o también de recogida de datos para intervenciones.

BIBLIOGRAFÍA

Alby, E. (2015). Point cloud vs drawing on archaeological site. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *XL-5/W7*, 7–11. https://doi.org/10.5194/isprsarchives-XL-5-W7-7-2015

Alonso Sánchez, A., Cerrillo M. de Cáceres, E., & Fernández Corrales, J. M. (1990). TRES EJEMPLOS DE POBLAMIENTO RURAL ROMANO EN TORNO A CIUDADES DE LA VIA DE LA PLATA. *Hispania Revista Espanola De Historia*, *10*(0), 73–88. Retrieved from http://campus.usal.es.ezproxy.usal.es/~revistas_trabajo/index.php/0213-2052/article/view/6426/6429

Aracena-Pizarro, D., & Daneri-Alvarado, N. (2013). Detección de puntos claves mediante SIFT paralelizado en GPU. *Ingeniare. Revista Chilena de Ingenier?a*, *21*(3), 438–447. https://doi.org/10.4067/S0718-33052013000300013

Belloso, M. L. (2004). El proyecto Alba-Plata (1998-2004): Ruta patrimonial de Extremadura, 585–597. Retrieved from http://www.dip-badajoz.es/cultura/ceex/reex_digital/reex_LXIII/2007/T. LXIII n. 2 2007 mayo-ag/RV001095.pdf

Cerrillo M. de Cáceres, E. (n.d.). Cáparra. Municipium flavium caperensis. Retrieved June 20, 2017, from http://bib.cervantesvirtual.com/portal/antigua/caparra.shtml

Cerrillo M. de Cáceres, E. (1997). LA PROSPECCIÓN SISTEMÁTICA Y EL POBLAMIENTO ROMANO EN EXTREMADURA. Retrieved from http://www.biblioarqueologia.com/doc/080507CERRILLO1997.pdf

Cerrillo M. de Cáceres, E. (1998). Forum municipii Flavii Caparensis. *Empúries: Revista de Món Clàssic I Antiguitat Tardana*, (51), 77–92. Retrieved from http://www.raco.cat.ezproxy.usal.es/index.php/Empuries/article/view/118470/288375

Dellepiane, M., Dell'Unto, N., Callieri, M., Lindgren, S., & Scopigno, R. (2013). Archeological excavation monitoring using dense stereo matching techniques. *Journal of Cultural Heritage*, *14*(3), 201–210. https://doi.org/10.1016/j.culher.2012.01.011

Documentación para QGIS 2.6. (n.d.). Retrieved June 14, 2017, from http://docs.qgis.org/2.6/es/docs/index.html

Ergincan, F., Çabuk, A., Avdan, U., & Tün, M. (2010). Advanced technologies for archaeological documentation: Patara case. *Scientific Research and Essays*, *5*(18), 2615–2629. Retrieved from http://www.academicjournals.org/SRE

Flores, P., & Braun, J. (2011). Algoritmo SIFT: fundamento teórico. Retrieved from http://iie.fing.edu.uy/investigacion/grupos/gti/timag/trabajos/2011/keypoints/FundamentoSIFT.pdf

IDE Extremadura - Visualizador. (n.d.). Retrieved June 24, 2017, from http://www.ideex.es/IDEEXVisor/

Kersten, T., Mechelke, K., & Maziull, L. (2015). 3D MODEL OF AL ZUBARAH FORTRESS IN QATAR - TERRESTRIAL LASER SCANNING VS. DENSE IMAGE MATCHING. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *XL-5/W4*, 1–8. https://doi.org/10.5194/isprsarchives-XL-5-W4-1-2015

Paris, L., Liberatore, D., & Wahbeh, W. (2012). Digital Representation of Archeological Sites. Recent Excavation at Alba Fucens. Retrieved from http://research.arc.uniroma1.it/xmlui/handle/123456789/500

Programación en C++. (n.d.). Retrieved June 21, 2017, from https://es.wikibooks.org/wiki/Programación_en_C%2B%2B

Qt 4.8. (n.d.). Retrieved June 15, 2017, from http://doc.qt.io/qt-4.8/

Ruinas romanas de Cáparra. (n.d.). Retrieved June 19, 2017, from http://www.guijodegranadilla.com/caparra.htm

Stack Overflow. (n.d.). Retrieved June 21, 2017, from https://stackoverflow.com/

Sutton, T., Hugentobler, M., Sherman, G. E., Athan, T., Contreras, G., Macho, W., … Dobias, M. (n.d.). Quantum GIS Coding and Compilation Guide. Retrieved from http://download.osgeo.org/qgis/doc/manual/qgis-1.1.0_coding-compilation_guide_en.pdf

Torres-martínez, J. A., Seddaiu, M., Rodríguez-gonzálvez, P., Hernández-lópez, D., & González-aguilera, D. (2016). A Multi-Data Source and Multi-Sensor Approach for the 3D Reconstruction and Web Visualization of a Complex Archaelogical Site : The Case Study of. *Remote Sensing*, *8*(7), 550. https://doi.org/10.3390/rs8070550

VisualizadorBasicoIDEE. (n.d.). Retrieved June 24, 2017, from http://www.idee.es/visualizador/

Von Schwerin, J., Richards-Rissetto, H., Remondino, F., Agugiaro, G., & Girardi, G. (2013). The mayaarch3d project: A 3D webgis for analyzing ancient architecture and landscapes. *Literary and Linguistic Computing*, *28*(4), 736–753. https://doi.org/10.1093/llc/fqt059

VTK/Examples. (n.d.). Retrieved June 15, 2017, from http://www.vtk.org/Wiki/VTK/Examples/Cxx

AGRADECIMIENTOS

Quiero agradecer la paciencia y la información recibida del personal del yacimiento y del Centro de Interpretación de la Ciudad Romana de Cáparra, que me han tenido por allí indagando con frecuencia.

También a mi tutor, Ángel Luis Muñoz Nieto, por orientarme en el desarrollo de mi trabajo, y darme las claves para terminarlo con éxito.

A todos los profesores del Máster de Geotecnologías, por todo lo aprendido, sin duda ha sido una experiencia muy formativa, que me ha ayudado a ampliar mi visión de las cosas más allá de mi campo, la Arquitectura.

Por último, a mi familia, a los que están y a los que ya no, y amigos, por estar siempre ahí, apoyándome, en cada proyecto que emprendo.

ANEXO I. CÓDIGO DE LA APLICACIÓN

- **Archivo *CMakeLists.txt*:**

```
# CONFIGURACIÓN
cmake_minimum_required(VERSION 2.8)
SET(CMAKE_BUILD_TYPE Release)

if(POLICY CMP0020)
  cmake_policy(SET CMP0020 NEW)
endif()

PROJECT(VisorShpXYZ)
SET(CMAKE_COLOR_MAKEFILE ON)
SET(CMAKE_MODULE_PATH        ${CMAKE_CURRENT_SOURCE_DIR}/cmake_find_rules
${CMAKE_MODULE_PATH})

# LIBRERÍAS VTK
find_package(VTK REQUIRED)
include(${VTK_USE_FILE})

if(${VTK_VERSION} VERSION_GREATER "6" AND VTK_QT_VERSION VERSION_GREATER
"4")
  set(CMAKE_AUTOMOC ON)
  find_package(Qt5Widgets REQUIRED QUIET)
else()
  find_package(Qt4 REQUIRED)
  include(${QT_USE_FILE})
endif()

# LIBRERÍAS QGIS
FIND_PACKAGE(QGIS REQUIRED)
IF (NOT QGIS_FOUND)
  MESSAGE (SEND_ERROR "QGIS dependency was not found!")
ENDIF (NOT QGIS_FOUND)
IF (WIN32)
  IF (MSVC)
    ADD_DEFINITIONS("-DGUI_EXPORT=__declspec(dllimport)")
    ADD_DEFINITIONS("-DCORE_EXPORT=__declspec(dllimport)")
  ELSE (MSVC)
    ADD_DEFINITIONS("\"-DGUI_EXPORT=__declspec(dllimport)\"")
    ADD_DEFINITIONS("\"-DCORE_EXPORT=__declspec(dllimport)\"")
  ENDIF (MSVC)
ELSE (WIN32)
  ADD_DEFINITIONS(-DGUI_EXPORT=)
  ADD_DEFINITIONS(-DCORE_EXPORT=)
ENDIF (WIN32)
```

```
IF (CMAKE_BUILD_TYPE MATCHES Release)
  ADD_DEFINITIONS(-DVisorShpXYZRELEASE=1)
ENDIF (CMAKE_BUILD_TYPE MATCHES Release)

INCLUDE_DIRECTORIES(
    ${CMAKE_CURRENT_BINARY_DIR}
    ${QT_INCLUDE_DIR}
    ${QGIS_INCLUDE_DIR}
)

# ARCHIVOS
file(GLOB UI_FILES *.ui)
file(GLOB QT_WRAP *.h)
file(GLOB CXX_FILES *.cpp)

SET (VisorShpXYZ_SRCS
    visorshpxyz.cpp
)
SET (VisorShpXYZ_MOC_HDRS
  visorshpxyz.h
)
SET (VisorShpXYZ_RCCS
 visorshpxyz.qrc
)

# LIBRERÍAS QT
QT4_ADD_RESOURCES(VisorShpXYZ_RCC_SRCS ${VisorShpXYZ_RCCS})
SET (QT_USE_QT3SUPPORT FALSE)
SET (QT_USE_QTGUI TRUE)
SET (QT_USE_QTSQL TRUE)
SET (QT_USE_QTSVG TRUE)
SET (QT_USE_QTXML TRUE)
SET (QT_USE_QTNETWORK TRUE)
FIND_PACKAGE(Qt4 REQUIRED)
INCLUDE( ${QT_USE_FILE} )

if(${VTK_VERSION} VERSION_GREATER "6" AND VTK_QT_VERSION VERSION_GREATER
"4")
  qt5_wrap_ui(UISrcs ${UI_FILES} )
  add_executable(VisorShpXYZ MACOSX_BUNDLE
    ${CXX_FILES} ${UISrcs} ${QT_WRAP})
  qt5_use_modules(VisorShpXYZ Core Gui)
  target_link_libraries(VisorShpXYZ ${VTK_LIBRARIES})
else()
  QT4_WRAP_UI(UISrcs ${UI_FILES})
  QT4_WRAP_CPP(MOCSrcs ${QT_WRAP})
```

```
 add_executable(VisorShpXYZ MACOSX_BUNDLE ${CXX_FILES} ${UISrcs} ${MOCSrcs}
${VisorShpXYZ_RCC_SRCS})
 if(VTK_LIBRARIES)
  if(${VTK_VERSION} VERSION_LESS "6")
   target_link_libraries(VisorShpXYZ ${VTK_LIBRARIES} QVTK)
  else()
   target_link_libraries(VisorShpXYZ ${VTK_LIBRARIES})
  endif()
 else()
  target_link_libraries(VisorShpXYZ vtkHybrid QVTK vtkViews ${QT_LIBRARIES})
 endif()
endif()

IF(QT_QTSQL_FOUND)
 FIND_LIBRARY(QT_QTSQL_LIBRARY NAMES QtSql QtSql4 PATHS ${QT_LIBRARY_DIR}
NO_DEFAULT_PATH)
 SET(QT_LIBRARIES ${QT_LIBRARIES} ${QT_QTSQL_LIBRARY})
ENDIF(QT_QTSQL_FOUND)

TARGET_LINK_LIBRARIES(VisorShpXYZ
 ${QT_LIBRARIES}
 ${QGIS_CORE_LIBRARY}
 ${QGIS_GUI_LIBRARY}
)

IF (MSVC)
 FIND_LIBRARY(qt qtmain)
ENDIF (MSVC)
```

- **Archivo *visorshpxyz.h*:**

```
#ifndef VISORSHP_H
#define VISORSHP_H

#include <QMainWindow>
#include <QWidget>
#include <QObject>
#include <qgsmapcanvas.h>
#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QToolBar>
#include <qgsmaptool.h>
#include <qgslayertreemapcanvasbridge.h>
#include <QVTKWidget.h>
```

```
#include <QLineEdit>

class MiVisor: public QMainWindow {
   Q_OBJECT
public:
   explicit MiVisor(QWidget *parent = 0);
   ~MiVisor();
   void mostrar();
   void mostrarno();
   QMainWindow *mymainwindow;
   QWidget *centralwidget;
   QVTKWidget *vtkwidget;
public slots:
//    void encuadre();
//    void zoomAmpliar();
//    void zoomReducir();
   void zoomCompleto();
   void zoom();
   void agregarCapa();
   void arco();
   void termas();
   void domus();
   void viviendas();
   void todo();
   void atributos();
   void abrirNube();
   void cerrarNube();
   void acercaDe();
private:
    QgsMapCanvas * mypMapCanvas;
    QVBoxLayout  * layoutV;
    QHBoxLayout * layoutH;
    QToolBar * toolbar;
    QgsMapTool * mypPanTool;
    QgsMapTool * mypZoomInTool;
    QgsMapTool * mypZoomOutTool;
    QgsLayerTreeMapCanvasBridge *bridge;
    QgsVectorLayer * mypLayer16;
};

#endif // VISORSHP_H
```

- **Archivo *visorshpxyz.cpp*:**

```
#include "visorshpxyz.h"
#include <iostream>
//
// QGIS Includes
//
#include <qgsapplication.h>
#include <qgsproviderregistry.h>
#include <qgssinglesymbolrendererv2.h>
#include <qgsmaplayerregistry.h>
#include <qgsvectorlayer.h>
#include <qgsmapcanvas.h>
#include <qgsmaptool.h>
#include <qgslayertreemapcanvasbridge.h>
#include <qgsproject.h>
#include <qgslayertreemodel.h>
#include <qgslayertreeview.h>
//
// QGIS Map tools
//
#include <qgsmaptoolpan.h>
#include <qgsmaptoolzoom.h>
//
// Qt Includes
//
#include <QApplication>
#include <QString>
#include <QWidget>
#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QMenuBar>
#include <QMenu>
#include <QToolBar>
#include <QAction>
#include <QIcon>
#include <QFileDialog>
#include <QFileInfo>
#include <QList>
#include <QDockWidget>
#include <QTableWidget>
#include <QStringList>
#include <QFile>
#include <QRegExp>
#include <QTableWidgetItem>
#include <QTextEdit>
#include <QInputDialog>
```

```
#include <QLineEdit>
//
// VTK Includes
//
//Abrir xyz
#include <vtkSmartPointer.h>
#include <vtkSimplePointsReader.h>
#include <vtkPolyDataMapper.h>
#include <vtkActor.h>
#include <vtkProperty.h>
#include <vtkRenderer.h>
#include <vtkRenderWindow.h>
#include <vtkRenderWindowInteractor.h>

//Abrir texto plano
#include "vtkVersion.h"
#include "vtkSmartPointer.h"
#include "vtkPolyData.h"
#include "vtkPoints.h"
#include "vtkCellArray.h"
#include "vtkXMLPolyDataWriter.h"
#include <iostream>

//Colorear nube
#include <vtkVersion.h>
#include <vtkSmartPointer.h>
#include <vtkPoints.h>
#include <vtkPolyData.h>
#include <vtkPointData.h>
#include <vtkCellArray.h>
#include <vtkUnsignedCharArray.h>
#include <vtkPolyDataMapper.h>
#include <vtkActor.h>
#include <vtkRenderWindow.h>
#include <vtkRenderer.h>
#include <vtkRenderWindowInteractor.h>
#include <vtkVertexGlyphFilter.h>
#include <vtkProperty.h>
// For compatibility with new VTK generic data arrays
#ifdef vtkGenericDataArray_h
#define InsertNextTupleValue InsertNextTypedTuple
#endif

//QVTKWidget
#include <QVTKWidget.h>
```

```
int main(int argc, char *argv[])
{
   // Iniciar la aplicacion

   QgsApplication app(argc, argv, true);
   QgsApplication::setPrefixPath("C:/Program Files (x86)/QGIS Brighton/apps/qgis",
true);
   QgsApplication::initQgis();

   MiVisor w;
   w.mostrar();

   // Ejecutar la aplicacion

   return app.exec();
   return EXIT_SUCCESS;
}

MiVisor::MiVisor(QWidget *parent):QMainWindow(parent){

}

MiVisor::~MiVisor(){

}

void MiVisor::mostrarno()
{

   std::string inputFilename = "test1.xyz";
//    QApplication app(argc, argv);
   QVTKWidget *widget = new QVTKWidget ;
   widget->show();
   widget->resize( 256, 256 );
   // Read the file
   vtkSmartPointer<vtkSimplePointsReader> reader =
    vtkSmartPointer<vtkSimplePointsReader>::New();
   reader->SetFileName(inputFilename.c_str());
   reader->Update();
   // Visualize
   vtkSmartPointer<vtkPolyDataMapper> mapper =
    vtkSmartPointer<vtkPolyDataMapper>::New();
   mapper->SetInputConnection(reader->GetOutputPort());

   vtkSmartPointer<vtkActor> actor =
    vtkSmartPointer<vtkActor>::New();
```

```
   actor->SetMapper(mapper);
//    actor->GetProperty()->SetPointSize(4);

   vtkSmartPointer<vtkRenderer> renderer =
     vtkSmartPointer<vtkRenderer>::New();
   renderer->AddActor(actor);
//    renderer->SetBackground(.3, .6, .3); // Background color green
   renderer->SetBackground(.3, .3, .3); // Background color green
   widget->GetRenderWindow()->AddRenderer( renderer );
//    widget.show();
//    return app.exec();
//    return EXIT_SUCCESS;


}

void MiVisor::mostrar(){

   // CAPA 1

   QString myPluginsDir = "C:/Program Files (x86)/QGIS Brighton/apps/qgis/lib";
   QString myLayerPath = "C:/Qt/Projects/VisorShpXYZ-build/shp/Vias";
   QString myLayerBaseName = "Vias";
   QString myProviderName = "ogr";

   // Instanciar Provider Registry

   QgsProviderRegistry::instance(myPluginsDir);

   // Crear una capa al iniciar

   QgsVectorLayer * mypLayer = new QgsVectorLayer(myLayerPath,
myLayerBaseName, myProviderName);
   QgsSingleSymbolRendererV2 *mypRenderer = new
QgsSingleSymbolRendererV2(QgsSymbolV2::defaultSymbol(mypLayer-
>geometryType()));
   QList <QgsMapCanvasLayer> myLayerSet;
   mypLayer->setRendererV2(mypRenderer);
   if (mypLayer->isValid()){
   qDebug("Layer is valid");}
   else{
   qDebug("Layer is NOT valid");}
   QgsSymbolV2 * symbol = mypRenderer->symbol();
   symbol->setColor(QColor(85,148,135));

   // Añadir la capa al Layer Registry y al Layer Set
```

```
//    QgsMapLayerRegistry::instance()->addMapLayer(mypLayer, true);
//    myLayerSet.append(QgsMapCanvasLayer(mypLayer, true));

   // CAPA 2

   QString myLayerPath2 = "C:/Qt/Projects/VisorShpXYZ-build/shp/VegetacionSup";
   QString myLayerBaseName2 = "VegetacionSup";

   QgsVectorLayer * mypLayer2 = new QgsVectorLayer(myLayerPath2,
myLayerBaseName2, myProviderName);
   QgsSingleSymbolRendererV2 *mypRenderer2 = new
QgsSingleSymbolRendererV2(QgsSymbolV2::defaultSymbol(mypLayer2-
>geometryType()));
   QList <QgsMapCanvasLayer> myLayerSet2;
   mypLayer2->setRendererV2(mypRenderer2);
   QgsSymbolV2 * symbol2 = mypRenderer2->symbol();
   symbol2->setColor(QColor(237,247,227));

   QgsMapLayerRegistry::instance()->addMapLayer(mypLayer2, true);
   myLayerSet.append(QgsMapCanvasLayer(mypLayer2, true));

   // CAPA 3

   QString myLayerPath3 = "C:/Qt/Projects/VisorShpXYZ-build/shp/Vegetacion";
   QString myLayerBaseName3 = "Vegetacion";

   QgsVectorLayer * mypLayer3 = new QgsVectorLayer(myLayerPath3,
myLayerBaseName3, myProviderName);
   QgsSingleSymbolRendererV2 *mypRenderer3 = new
QgsSingleSymbolRendererV2(QgsSymbolV2::defaultSymbol(mypLayer3-
>geometryType()));
   QList <QgsMapCanvasLayer> myLayerSet3;
   mypLayer3->setRendererV2(mypRenderer3);
   QgsSymbolV2 * symbol3 = mypRenderer3->symbol();
   symbol3->setColor(QColor(208,225,222));

   QgsMapLayerRegistry::instance()->addMapLayer(mypLayer3, true);
   myLayerSet.append(QgsMapCanvasLayer(mypLayer3, true));

   // CAPA 4

   QString myLayerPath4 = "C:/Qt/Projects/VisorShpXYZ-
build/shp/Publico_Foro_Porticos";
   QString myLayerBaseName4 = "Publico_Foro_Porticos";

QgsVectorLayer * mypLayer4 = new QgsVectorLayer(myLayerPath4,
myLayerBaseName4, myProviderName);
```

```
QgsSingleSymbolRendererV2 *mypRenderer4 = new
QgsSingleSymbolRendererV2(QgsSymbolV2::defaultSymbol(mypLayer4-
>geometryType()));
   QList <QgsMapCanvasLayer> myLayerSet4;
   mypLayer4->setRendererV2(mypRenderer4);
   QgsSymbolV2 * symbol4 = mypRenderer4->symbol();
   symbol4->setColor(QColor(235,217,217));

   QgsMapLayerRegistry::instance()->addMapLayer(mypLayer4, true);
   myLayerSet.append(QgsMapCanvasLayer(mypLayer4, true));

   // CAPA 5

   QString myLayerPath5 = "C:/Qt/Projects/VisorShpXYZ-
build/shp/Privado_Patios_Jardin";
   QString myLayerBaseName5 = "Privado_Patios_Jardin";

   QgsVectorLayer * mypLayer5 = new QgsVectorLayer(myLayerPath5,
myLayerBaseName5, myProviderName);
   QgsSingleSymbolRendererV2 *mypRenderer5 = new
QgsSingleSymbolRendererV2(QgsSymbolV2::defaultSymbol(mypLayer5-
>geometryType()));
   QList <QgsMapCanvasLayer> myLayerSet5;
   mypLayer5->setRendererV2(mypRenderer5);
   QgsSymbolV2 * symbol5 = mypRenderer5->symbol();
   symbol5->setColor(QColor(251,221,161));

   QgsMapLayerRegistry::instance()->addMapLayer(mypLayer5, true);
   myLayerSet.append(QgsMapCanvasLayer(mypLayer5, true));

   // CAPA 6

   QString myLayerPath6 = "C:/Qt/Projects/VisorShpXYZ-build/shp/Calzada";
   QString myLayerBaseName6 = "Calzada";

   QgsVectorLayer * mypLayer6 = new QgsVectorLayer(myLayerPath6,
myLayerBaseName6, myProviderName);
   QgsSingleSymbolRendererV2 *mypRenderer6 = new
QgsSingleSymbolRendererV2(QgsSymbolV2::defaultSymbol(mypLayer6-
>geometryType()));
   QList <QgsMapCanvasLayer> myLayerSet6;
   mypLayer6->setRendererV2(mypRenderer6);
   QgsSymbolV2 * symbol6 = mypRenderer6->symbol();
   symbol6->setColor(QColor(186,74,74));

   QgsMapLayerRegistry::instance()->addMapLayer(mypLayer6, true);
   myLayerSet.append(QgsMapCanvasLayer(mypLayer6, true));
```

```
    // CAPA 7

    QString myLayerPath7 = "C:/Qt/Projects/VisorShpXYZ-
build/shp/Decumano_ViaDeLaPlata";
    QString myLayerBaseName7 = "Decumano_ViaDeLaPlata";

    QgsVectorLayer * mypLayer7 = new QgsVectorLayer(myLayerPath7,
myLayerBaseName7, myProviderName);
    QgsSingleSymbolRendererV2 *mypRenderer7 = new
QgsSingleSymbolRendererV2(QgsSymbolV2::defaultSymbol(mypLayer7-
>geometryType()));
    QList <QgsMapCanvasLayer> myLayerSet7;
    mypLayer7->setRendererV2(mypRenderer7);
    QgsSymbolV2 * symbol7 = mypRenderer7->symbol();
    symbol7->setColor(QColor(186,151,106));

    QgsMapLayerRegistry::instance()->addMapLayer(mypLayer7, true);
    myLayerSet.append(QgsMapCanvasLayer(mypLayer7, true));

    // CAPA 8

    QString myLayerPath8 = "C:/Qt/Projects/VisorShpXYZ-build/shp/Cardo";
    QString myLayerBaseName8 = "Cardo";

    QgsVectorLayer * mypLayer8 = new QgsVectorLayer(myLayerPath8,
myLayerBaseName8, myProviderName);
    QgsSingleSymbolRendererV2 *mypRenderer8 = new
QgsSingleSymbolRendererV2(QgsSymbolV2::defaultSymbol(mypLayer8-
>geometryType()));
    QList <QgsMapCanvasLayer> myLayerSet8;
    mypLayer8->setRendererV2(mypRenderer8);
    QgsSymbolV2 * symbol8 = mypRenderer8->symbol();
    symbol8->setColor(QColor(235,191,134));

    QgsMapLayerRegistry::instance()->addMapLayer(mypLayer8, true);
    myLayerSet.append(QgsMapCanvasLayer(mypLayer8, true));

    // CAPA 9

    QString myLayerPath9 = "C:/Qt/Projects/VisorShpXYZ-build/shp/Vias";
    QString myLayerBaseName9 = "Vias";

    QgsVectorLayer * mypLayer9 = new QgsVectorLayer(myLayerPath9,
myLayerBaseName9, myProviderName);
```

```
   QgsSingleSymbolRendererV2 *mypRenderer9 = new
QgsSingleSymbolRendererV2(QgsSymbolV2::defaultSymbol(mypLayer9-
>geometryType()));
   QList <QgsMapCanvasLayer> myLayerSet9;
   mypLayer9->setRendererV2(mypRenderer9);
   QgsSymbolV2 * symbol9 = mypRenderer9->symbol();
   symbol9->setColor(QColor(235,191,134)); //(QColor(66,31,7));

   QgsMapLayerRegistry::instance()->addMapLayer(mypLayer9, true);
   myLayerSet.append(QgsMapCanvasLayer(mypLayer9, true));

   // CAPA 10

   QString myLayerPath10 = "C:/Qt/Projects/VisorShpXYZ-build/shp/Puertas";
   QString myLayerBaseName10 = "Puertas";

   QgsVectorLayer * mypLayer10 = new QgsVectorLayer(myLayerPath10,
myLayerBaseName10, myProviderName);
   QgsSingleSymbolRendererV2 *mypRenderer10 = new
QgsSingleSymbolRendererV2(QgsSymbolV2::defaultSymbol(mypLayer10-
>geometryType()));
   QList <QgsMapCanvasLayer> myLayerSet10;
   mypLayer10->setRendererV2(mypRenderer10);
   QgsSymbolV2 * symbol10 = mypRenderer10->symbol();
   symbol10->setColor(QColor(234,41,73));

   QgsMapLayerRegistry::instance()->addMapLayer(mypLayer10, true);
   myLayerSet.append(QgsMapCanvasLayer(mypLayer10, true));

   // CAPA 11

   QString myLayerPath11 = "C:/Qt/Projects/VisorShpXYZ-build/shp/Foro";
   QString myLayerBaseName11 = "Foro";

   QgsVectorLayer * mypLayer11 = new QgsVectorLayer(myLayerPath11,
myLayerBaseName11, myProviderName);
   QgsSingleSymbolRendererV2 *mypRenderer11 = new
QgsSingleSymbolRendererV2(QgsSymbolV2::defaultSymbol(mypLayer11-
>geometryType()));
   QList <QgsMapCanvasLayer> myLayerSet11;
   mypLayer11->setRendererV2(mypRenderer11);
   QgsSymbolV2 * symbol11 = mypRenderer11->symbol();
   symbol11->setColor(QColor(104,57,90));

   QgsMapLayerRegistry::instance()->addMapLayer(mypLayer11, true);
   myLayerSet.append(QgsMapCanvasLayer(mypLayer11, true));
```

```
    // CAPA 12

    QString myLayerPath12 = "C:/Qt/Projects/VisorShpXYZ-build/shp/Arco";
    QString myLayerBaseName12 = "Arco";

    QgsVectorLayer * mypLayer12 = new QgsVectorLayer(myLayerPath12,
myLayerBaseName12, myProviderName);
    QgsSingleSymbolRendererV2 *mypRenderer12 = new
QgsSingleSymbolRendererV2(QgsSymbolV2::defaultSymbol(mypLayer12-
>geometryType()));
    QList <QgsMapCanvasLayer> myLayerSet12;
    mypLayer12->setRendererV2(mypRenderer12);
    QgsSymbolV2 * symbol12 = mypRenderer12->symbol();
    symbol12->setColor(QColor(124,3,3));

    QgsMapLayerRegistry::instance()->addMapLayer(mypLayer12, true);
    myLayerSet.append(QgsMapCanvasLayer(mypLayer12, true));

    // CAPA 13

    QString myLayerPath13 = "C:/Qt/Projects/VisorShpXYZ-build/shp/Viviendas";
    QString myLayerBaseName13 = "Viviendas";

    QgsVectorLayer * mypLayer13 = new QgsVectorLayer(myLayerPath13,
myLayerBaseName13, myProviderName);
    QgsSingleSymbolRendererV2 *mypRenderer13 = new
QgsSingleSymbolRendererV2(QgsSymbolV2::defaultSymbol(mypLayer13-
>geometryType()));
    QList <QgsMapCanvasLayer> myLayerSet13;
    mypLayer13->setRendererV2(mypRenderer13);
    QgsSymbolV2 * symbol13 = mypRenderer13->symbol();
    symbol13->setColor(QColor(75,74,76));

    QgsMapLayerRegistry::instance()->addMapLayer(mypLayer13, true);
    myLayerSet.append(QgsMapCanvasLayer(mypLayer13, true));

    // CAPA 14

    QString myLayerPath14 = "C:/Qt/Projects/VisorShpXYZ-build/shp/Termas";
    QString myLayerBaseName14 = "Termas";

    QgsVectorLayer * mypLayer14 = new QgsVectorLayer(myLayerPath14,
myLayerBaseName14, myProviderName);
    QgsSingleSymbolRendererV2 *mypRenderer14 = new
QgsSingleSymbolRendererV2(QgsSymbolV2::defaultSymbol(mypLayer14-
>geometryType()));
    QList <QgsMapCanvasLayer> myLayerSet14;
```

```
    mypLayer14->setRendererV2(mypRenderer14);
    QgsSymbolV2 * symbol14 = mypRenderer14->symbol();
    symbol14->setColor(QColor(20,30,67));

    QgsMapLayerRegistry::instance()->addMapLayer(mypLayer14, true);
    myLayerSet.append(QgsMapCanvasLayer(mypLayer14, true));

    // CAPA 15

    QString myLayerPath15 = "C:/Qt/Projects/VisorShpXYZ-build/shp/Domus";
    QString myLayerBaseName15 = "Domus";

    QgsVectorLayer * mypLayer15 = new QgsVectorLayer(myLayerPath15,
myLayerBaseName15, myProviderName);
    QgsSingleSymbolRendererV2 *mypRenderer15 = new
QgsSingleSymbolRendererV2(QgsSymbolV2::defaultSymbol(mypLayer15-
>geometryType()));
    QList <QgsMapCanvasLayer> myLayerSet15;
    mypLayer15->setRendererV2(mypRenderer15);
    QgsSymbolV2 * symbol15 = mypRenderer15->symbol();
    symbol15->setColor(QColor(5,97,78));

    QgsMapLayerRegistry::instance()->addMapLayer(mypLayer15, true);
    myLayerSet.append(QgsMapCanvasLayer(mypLayer15, true));

    // CAPA 16

    QString myLayerPath16 = "C:/Qt/Projects/VisorShpXYZ-build/shp/Yacimiento";
    QString myLayerBaseName16 = "Yacimiento";

    mypLayer16 = new QgsVectorLayer(myLayerPath16, myLayerBaseName16,
myProviderName);
    QgsSingleSymbolRendererV2 *mypRenderer16 = new
QgsSingleSymbolRendererV2(QgsSymbolV2::defaultSymbol(mypLayer16-
>geometryType()));
    QList <QgsMapCanvasLayer> myLayerSet16;
    mypLayer16->setRendererV2(mypRenderer16);
    QgsSymbolV2 * symbol16 = mypRenderer16->symbol();
    symbol16->setColor(QColor(70,70,70)); //(QColor(129,126,133));

    QgsMapLayerRegistry::instance()->addMapLayer(mypLayer16, true);
    myLayerSet.append(QgsMapCanvasLayer(mypLayer16, true));

    // Crear el Map Canvas

    mypMapCanvas = new QgsMapCanvas(0, 0);
    mypMapCanvas->setExtent(mypLayer->extent()); //
```

```
    mypMapCanvas->enableAntiAliasing(true);
    mypMapCanvas->setCanvasColor(QColor(255, 255, 255));
//    mypMapCanvas->setFixedWidth(500);
    mypMapCanvas->freeze(false);

    // Mostrar en el Map Canvas el Layer Set (conjunto de capas)

    mypMapCanvas->setLayerSet(myLayerSet); //
    mypMapCanvas->setLayerSet(myLayerSet2); //
    mypMapCanvas->setLayerSet(myLayerSet3);
    mypMapCanvas->setLayerSet(myLayerSet4);
    mypMapCanvas->setLayerSet(myLayerSet5);
    mypMapCanvas->setLayerSet(myLayerSet6);
    mypMapCanvas->setLayerSet(myLayerSet7);
    mypMapCanvas->setLayerSet(myLayerSet8);
    mypMapCanvas->setLayerSet(myLayerSet9);
    mypMapCanvas->setLayerSet(myLayerSet10);
    mypMapCanvas->setLayerSet(myLayerSet11);
    mypMapCanvas->setLayerSet(myLayerSet12);
    mypMapCanvas->setLayerSet(myLayerSet13);
    mypMapCanvas->setLayerSet(myLayerSet14);
    mypMapCanvas->setLayerSet(myLayerSet15);
    mypMapCanvas->setLayerSet(myLayerSet16);
//    mypMapCanvas->setExtent(mypLayer8->extent());
    mypMapCanvas->setExtent(mypLayer16->extent());

    mypMapCanvas->setVisible(true);
    mypMapCanvas->refresh();
    mypMapCanvas->show();

    // Mostrar en una ventana el Map Canvas y el menu

    QVBoxLayout* layoutV = new QVBoxLayout();
    QMenuBar* menu = new QMenuBar;
    layoutV->addWidget(menu);
    QMenu *menu_mapa = menu->addMenu("Mapa");
    QMenu *menu_nube = menu->addMenu("Nube de puntos");
    QMenu *menu_acerca = menu->addMenu("Acerca de...");
    QToolBar * toolbar = new QToolBar;
    QToolBar * toolbar2 = new QToolBar;
//    toolbar->setIconSize(QSize(70,20));
//    layoutV->addWidget(toolbar);

    mymainwindow = new QMainWindow();
    centralwidget = new QWidget(mymainwindow);
    vtkwidget = new QVTKWidget(mymainwindow);
//    vtkwidget->setFixedWidth(600);
```

```
    mymainwindow->setWindowTitle("Caparra SIG-3D");
    mymainwindow->setWindowState(Qt::WindowMaximized);
    mymainwindow->setWindowIcon(QIcon(":/iconos/Icono2.png"));
//    mymainwindow->setFont(QFont("Courier", 8, QFont::Bold, true));
//    vtkwidget->GetRenderWindow()->AddRenderer(imgview->GetRenderer());
//    vtkwidget->setFixedSize(512,512);
//    QWidget* window = new QWidget;
//    window->setLayout(layoutV);
//    window->show();
//    window->setWindowTitle("Visor de Shapefiles");

    QHBoxLayout* layoutH = new QHBoxLayout();
    layoutV->addLayout(layoutH);
    QToolBar * toolbarV = new QToolBar;
//    layoutH->addWidget(toolbarV);
    toolbarV->setOrientation(Qt::Vertical);
    toolbarV->setIconSize(QSize(12,12));
    QDockWidget *legend = new QDockWidget("Capas", mymainwindow); //
//    layoutH->addWidget(legend);
    QHBoxLayout* layoutH1 = new QHBoxLayout();
    layoutH->addLayout(layoutH1);
    layoutH1->setContentsMargins(0,15,0,0);
    QVBoxLayout* layoutV1 = new QVBoxLayout();
    layoutH1->addWidget(toolbarV);
    layoutV1->addWidget(legend);
    layoutH1->addLayout(layoutV1);
//    layoutH->addWidget(mypMapCanvas);
    //layoutH->addWidget(new QgsMapCanvas);
//    mymainwindow->setGeometry(280,120,800,500); //

    QVBoxLayout* layoutV2 = new QVBoxLayout();
    layoutH->addLayout(layoutV2);
    layoutV2->addWidget(toolbar);
    layoutV2->addWidget(mypMapCanvas);
    QVBoxLayout* layoutV3 = new QVBoxLayout();
    layoutH->addLayout(layoutV3);
    layoutV3->addWidget(toolbar2);

    vtkwidget->show();
    layoutV3->addWidget(vtkwidget); // layoutH

    centralwidget->setLayout(layoutV);
    mymainwindow->setCentralWidget(centralwidget);
    mymainwindow->show();

    // Slots menu
```

```
   QAction  *actionAgregarCapa  =  new  QAction(QIcon(":/iconos/AgregarCapa.png"),
"Agregar capa...", mymainwindow); //
   QAction *actionSalir = new QAction("Salir", mymainwindow); //

   menu_mapa->addAction(actionAgregarCapa);
//    menu_mapa->addAction(actionSalir);

   connect(actionAgregarCapa, SIGNAL (triggered()),this, SLOT (agregarCapa()));
   connect(actionSalir, SIGNAL (triggered()),qApp, SLOT (quit()));

   // ToolBar

//    QAction *actionZoomAmpliar = new
QAction(QIcon(":/iconos/ZoomAmpliar.png"), "Zoom Ampliar", mymainwindow);
//    QAction *actionZoomReducir = new QAction(QIcon(":/iconos/ZoomReducir.png"),
"Zoom Reducir", mymainwindow);
//    QAction *actionEncuadre = new QAction(QIcon(":/iconos/Encuadre.png"),
"Encuadre", mymainwindow);
   QAction *actionZoomCompleto = new QAction
(QIcon(":/iconos/ZoomCompleto.png"), "Zoom Completo", mymainwindow);
   QAction *actionZoom = new QAction (QIcon(":/iconos/ZoomAmpliar.png"),
"Zoom", mymainwindow);
   QAction *actionMapa = new QAction (QIcon(":/iconos/Mapa.png"), "",
mymainwindow);
   QAction *actionNube = new QAction (QIcon(":/iconos/Nube.png"), "",
mymainwindow);
   QAction *actionVacio = new QAction (QIcon(":/iconos/Vacio.png"), "",
mymainwindow);
   QAction *actionArco = new QAction (QIcon(":/iconos/Arco.png"), "Arco",
mymainwindow);
   QAction *actionTermas = new QAction (QIcon(":/iconos/Termas.png"), "Termas",
mymainwindow);
   QAction *actionDomus = new QAction (QIcon(":/iconos/Domus.png"), "Domus",
mymainwindow);
   QAction *actionViviendas = new QAction (QIcon(":/iconos/Viviendas.png"),
"Viviendas", mymainwindow);
   QAction *actionTodo = new QAction (QIcon(":/iconos/Todo.png"), "Todo",
mymainwindow);
   QAction *actionAtributos = new QAction (QIcon(":/iconos/Atributos.png"),
"Atributos", mymainwindow);
   QAction *actionAbrirNube = new QAction (QIcon(":/iconos/AgregarCapa.png"),
"Abrir Nube", mymainwindow);
   QAction *actionCerrarNube = new QAction (QIcon(":/iconos/Cerrar.png"), "Cerrar
Nube", mymainwindow);
   QAction *actionAcercaDe = new QAction (QIcon(":/iconos/Vacio.png"), "Acerca
De...", mymainwindow);
```

```
//    connect(actionZoomAmpliar, SIGNAL(triggered()), this, SLOT(zoomAmpliar()));
//    connect(actionZoomReducir, SIGNAL(triggered()), this, SLOT(zoomReducir()));
//    connect(actionEncuadre, SIGNAL(triggered()), this, SLOT(encuadre()));
    connect(actionZoomCompleto, SIGNAL(triggered()), this, SLOT(zoomCompleto()));
    connect(actionZoom, SIGNAL(triggered()), this, SLOT(zoom()));
    connect(actionArco, SIGNAL(triggered()), this, SLOT(arco()));
    connect(actionTermas, SIGNAL(triggered()), this, SLOT(termas()));
    connect(actionViviendas, SIGNAL(triggered()), this, SLOT(viviendas()));
    connect(actionTodo, SIGNAL(triggered()), this, SLOT(todo()));
    connect(actionAtributos, SIGNAL(triggered()), this, SLOT(atributos()));
    connect(actionDomus, SIGNAL(triggered()), this, SLOT(domus()));
    connect(actionAbrirNube, SIGNAL(triggered()), this, SLOT(abrirNube()));
    connect(actionCerrarNube, SIGNAL(triggered()), this, SLOT(cerrarNube()));
    connect(actionAcercaDe, SIGNAL(triggered()), this, SLOT(acercaDe()));

    toolbar->addAction(actionMapa);
    toolbar->addSeparator();
    toolbar->addAction(actionAgregarCapa);
//    toolbar->addAction(actionZoomAmpliar);
//    toolbar->addAction(actionZoomReducir);
//    toolbar->addAction(actionEncuadre);
    toolbar->addAction(actionZoomCompleto);
    toolbar->addAction(actionZoom);
    toolbar->addAction(actionAtributos);

    toolbar2->addAction(actionNube);
    toolbar2->addSeparator();
    toolbar2->addAction(actionAbrirNube);
    toolbar2->addAction(actionCerrarNube);

    toolbarV->addAction(actionVacio);
    toolbarV->addAction(actionTodo);
    toolbarV->addAction(actionDomus);
    toolbarV->addAction(actionTermas);
    toolbarV->addAction(actionViviendas);
    toolbarV->addAction(actionArco);

//    mypPanTool = new QgsMapToolPan(mypMapCanvas);
//    mypPanTool->setAction(actionEncuadre);
//    mypZoomInTool = new QgsMapToolZoom(mypMapCanvas, FALSE); // false = in
//    mypZoomInTool->setAction(actionZoomAmpliar);
//    mypZoomOutTool = new QgsMapToolZoom(mypMapCanvas, TRUE ); //true = out
//    mypZoomOutTool->setAction(actionZoomReducir);

    menu_mapa->addAction(actionAtributos);
    menu_mapa->addAction(actionSalir);
    menu_nube->addAction(actionAbrirNube);
```

```
    menu_nube->addAction(actionCerrarNube);
    menu_acerca->addAction(actionAcercaDe);

    // Arbol de capas

    bridge = new QgsLayerTreeMapCanvasBridge(QgsProject::instance()-
>layerTreeRoot(), mypMapCanvas); //sincronizar con el mapa
    QgsLayerTreeModel *model = new QgsLayerTreeModel(QgsProject::instance()-
>layerTreeRoot()); //crear arbol
    model->setFlag(QgsLayerTreeModel::AllowNodeReorder, true);
    model->setFlag(QgsLayerTreeModel::AllowNodeChangeVisibility, true);
    model->setFlag(QgsLayerTreeModel::ShowLegend, true);
    QgsLayerTreeView *view = new QgsLayerTreeView(); //crear vista
    view->setModel(model);

    // Mostrar el arbol de capas en el DockWidget

    legend->setObjectName("layers");
    legend->setFixedWidth(210);
    legend->setFixedHeight(400);
    layoutV1->setAlignment(legend, Qt::AlignTop);
//    layoutV1->setContentsMargins(0,15,0,0);
    legend->setWidget(view);

//    // Mostrar nube completa
//    // Read the file
//      vtkSmartPointer<vtkSimplePointsReader> reader =
//        vtkSmartPointer<vtkSimplePointsReader>::New();
//      reader->SetFileName("nubes/todoxyz.xyz");
//      reader->Update();
//      vtkPolyData* pointsPolydata = reader->GetOutput();
//      for(vtkIdType i = 0; i < pointsPolydata->GetNumberOfPoints(); i++)
//          {
//          double p[3];
//          pointsPolydata->GetPoint(i,p);
//          }

//      vtkSmartPointer<vtkSimplePointsReader> readerRGB =
//        vtkSmartPointer<vtkSimplePointsReader>::New();
//      readerRGB->SetFileName("nubes/todorgb.xyz");
//      readerRGB->Update();
//      vtkPolyData* pointsPolydataRGB = readerRGB->GetOutput();
//      double e;
//      for(vtkIdType i = 0; i < pointsPolydataRGB->GetNumberOfPoints(); i++)
//          {
//          e = pointsPolydataRGB->GetNumberOfPoints();
//          double p[3];
```

```
//         pointsPolydataRGB->GetPoint(i,p);
//         }
//     cout << e << endl;

//     vtkSmartPointer<vtkVertexGlyphFilter> vertexFilter =
//       vtkSmartPointer<vtkVertexGlyphFilter>::New();
//    #if VTK_MAJOR_VERSION <= 5
//     vertexFilter->SetInputConnection(pointsPolydata->GetProducerPort());
//    #else
//     vertexFilter->SetInputData(pointsPolydata);
//    #endif
//     vertexFilter->Update();

//     vtkSmartPointer<vtkPolyData> polydata =
//       vtkSmartPointer<vtkPolyData>::New();
//     polydata->ShallowCopy(vertexFilter->GetOutput());

//     // Setup colors

//     vtkSmartPointer<vtkUnsignedCharArray> colors =
//       vtkSmartPointer<vtkUnsignedCharArray>::New();
//     colors->SetNumberOfComponents(3);
//     colors->SetName ("Colors");
//     double c[3];
//     unsigned char f[3];
//     for(vtkIdType i = 0; i < pointsPolydataRGB->GetNumberOfPoints(); i++)
//         {
//         pointsPolydataRGB->GetPoint(i,c);
//         f[0] = (unsigned char)c[0];
//         f[1] = (unsigned char)c[1];
//         f[2] = (unsigned char)c[2];
//         colors->InsertNextTupleValue(f);
//         }
//     polydata->GetPointData()->SetScalars(colors);

//     // Visualization
//     vtkSmartPointer<vtkPolyDataMapper> mapper =
//       vtkSmartPointer<vtkPolyDataMapper>::New();
//    #if VTK_MAJOR_VERSION <= 5
//     mapper->SetInputConnection(polydata->GetProducerPort());
//    #else
//     mapper->SetInputData(polydata);
//    #endif

//     vtkSmartPointer<vtkActor> actor =
//       vtkSmartPointer<vtkActor>::New();
//     actor->SetMapper(mapper);
```

```
//     actor->GetProperty()->SetPointSize(5);

    vtkSmartPointer<vtkRenderer> renderer =
     vtkSmartPointer<vtkRenderer>::New();

//     renderer->AddActor(actor);
    renderer->SetBackground(.7, .7, .7); // Background color

    vtkwidget->GetRenderWindow()->AddRenderer( renderer );
    vtkwidget->update();

}

//void MiVisor::zoomAmpliar()
//{
//  mypMapCanvas->setMapTool(mypZoomInTool);
//}

//void MiVisor::zoomReducir()
//{
//  mypMapCanvas->setMapTool(mypZoomOutTool);
//}

//void MiVisor::encuadre()
//{
//  mypMapCanvas->setMapTool(mypPanTool);
//}

void MiVisor::zoomCompleto()
{
  mypMapCanvas->zoomToFullExtent();
}

void MiVisor::zoom()
{
  mypMapCanvas->setExtent(mypLayer16->extent());
  mypMapCanvas->refresh();
}

void MiVisor::agregarCapa()
{
   // Crear una capa

   QString myFileName = QFileDialog::getOpenFileName(0,"Abrir","","(*.shp)");
   QFileInfo myRasterFileInfo(myFileName);
   qDebug() << "myFileName =" << myFileName;
```

```
   QgsVectorLayer * mypLayer =  new QgsVectorLayer(myRasterFileInfo.filePath(),
myRasterFileInfo.completeBaseName(),"ogr");
   QgsSingleSymbolRendererV2 *mypRenderer = new
QgsSingleSymbolRendererV2(QgsSymbolV2::defaultSymbol(mypLayer-
>geometryType()));
   QList <QgsMapCanvasLayer> myLayerSet;
   mypLayer->setRendererV2(mypRenderer);
   if (mypLayer->isValid()){
   qDebug("Layer is valid");}
   else{
   qDebug("Layer is NOT valid");}

   // Añadir la capa al Layer Registry

   QgsMapLayerRegistry::instance()->addMapLayer(mypLayer, true);

   // Añadir la capa al Layer Set y mostrar

   myLayerSet.append(QgsMapCanvasLayer(mypLayer, true));

   mypMapCanvas->setExtent(mypLayer->extent());
   mypMapCanvas->setLayerSet(myLayerSet);
}

void MiVisor::arco()
{
   // Read the file
    vtkSmartPointer<vtkSimplePointsReader> reader =
     vtkSmartPointer<vtkSimplePointsReader>::New();
    reader->SetFileName("nubes/arcoxyz.xyz");
    reader->Update();
    vtkPolyData* pointsPolydata = reader->GetOutput();
    for(vtkIdType i = 0; i < pointsPolydata->GetNumberOfPoints(); i++)
        {
        double p[3];
        pointsPolydata->GetPoint(i,p);
        }

    vtkSmartPointer<vtkSimplePointsReader> readerRGB =
     vtkSmartPointer<vtkSimplePointsReader>::New();
    readerRGB->SetFileName("nubes/arcorgb.xyz");
    readerRGB->Update();
    vtkPolyData* pointsPolydataRGB = readerRGB->GetOutput();
    double e;
    for(vtkIdType i = 0; i < pointsPolydataRGB->GetNumberOfPoints(); i++)
        {
        e = pointsPolydataRGB->GetNumberOfPoints();
```

```
      double p[3];
      pointsPolydataRGB->GetPoint(i,p);
      }
  cout << e << endl;

  vtkSmartPointer<vtkVertexGlyphFilter> vertexFilter =
    vtkSmartPointer<vtkVertexGlyphFilter>::New();
#if VTK_MAJOR_VERSION <= 5
  vertexFilter->SetInputConnection(pointsPolydata->GetProducerPort());
#else
  vertexFilter->SetInputData(pointsPolydata);
#endif
  vertexFilter->Update();

  vtkSmartPointer<vtkPolyData> polydata =
    vtkSmartPointer<vtkPolyData>::New();
  polydata->ShallowCopy(vertexFilter->GetOutput());

  // Setup colors

  vtkSmartPointer<vtkUnsignedCharArray> colors =
    vtkSmartPointer<vtkUnsignedCharArray>::New();
  colors->SetNumberOfComponents(3);
  colors->SetName ("Colors");
  double c[3];
  unsigned char f[3];
  for(vtkIdType i = 0; i < pointsPolydataRGB->GetNumberOfPoints(); i++)
      {
      pointsPolydataRGB->GetPoint(i,c);
      f[0] = (unsigned char)c[0];
      f[1] = (unsigned char)c[1];
      f[2] = (unsigned char)c[2];
      colors->InsertNextTupleValue(f);
      }
  polydata->GetPointData()->SetScalars(colors);

  // Visualization
  vtkSmartPointer<vtkPolyDataMapper> mapper =
    vtkSmartPointer<vtkPolyDataMapper>::New();
#if VTK_MAJOR_VERSION <= 5
  mapper->SetInputConnection(polydata->GetProducerPort());
#else
  mapper->SetInputData(polydata);
#endif

  vtkSmartPointer<vtkActor> actor =
    vtkSmartPointer<vtkActor>::New();
```

```
    actor->SetMapper(mapper);
    actor->GetProperty()->SetPointSize(5);

    vtkSmartPointer<vtkRenderer> renderer =
      vtkSmartPointer<vtkRenderer>::New();
//    vtkSmartPointer<vtkRenderWindow> renderWindow =
//      vtkSmartPointer<vtkRenderWindow>::New();
//    renderWindow->AddRenderer(renderer);
//    vtkSmartPointer<vtkRenderWindowInteractor> renderWindowInteractor =
//      vtkSmartPointer<vtkRenderWindowInteractor>::New();
//    renderWindowInteractor->SetRenderWindow(renderWindow);

    renderer->AddActor(actor);
    renderer->SetBackground(.7, .7, .7); // Background color

//    renderWindow->Render();
//    renderWindowInteractor->Start();

    vtkwidget->GetRenderWindow()->AddRenderer( renderer );
    vtkwidget->update();

//    return EXIT_SUCCESS;
}

void MiVisor::termas()
{
  // Read the file
    vtkSmartPointer<vtkSimplePointsReader> reader =
      vtkSmartPointer<vtkSimplePointsReader>::New();
    reader->SetFileName("nubes/termasxyz.xyz");
    reader->Update();
    vtkPolyData* pointsPolydata = reader->GetOutput();
    for(vtkIdType i = 0; i < pointsPolydata->GetNumberOfPoints(); i++)
        {
        double p[3];
        pointsPolydata->GetPoint(i,p);
        }

    vtkSmartPointer<vtkSimplePointsReader> readerRGB =
      vtkSmartPointer<vtkSimplePointsReader>::New();
    readerRGB->SetFileName("nubes/termasrgb.xyz");
    readerRGB->Update();
    vtkPolyData* pointsPolydataRGB = readerRGB->GetOutput();
    double e;
    for(vtkIdType i = 0; i < pointsPolydataRGB->GetNumberOfPoints(); i++)
        {
        e = pointsPolydataRGB->GetNumberOfPoints();
```

```
      double p[3];
      pointsPolydataRGB->GetPoint(i,p);
      }
  cout << e << endl;

  vtkSmartPointer<vtkVertexGlyphFilter> vertexFilter =
    vtkSmartPointer<vtkVertexGlyphFilter>::New();
#if VTK_MAJOR_VERSION <= 5
  vertexFilter->SetInputConnection(pointsPolydata->GetProducerPort());
#else
  vertexFilter->SetInputData(pointsPolydata);
#endif
  vertexFilter->Update();

  vtkSmartPointer<vtkPolyData> polydata =
    vtkSmartPointer<vtkPolyData>::New();
  polydata->ShallowCopy(vertexFilter->GetOutput());

  // Setup colors

  vtkSmartPointer<vtkUnsignedCharArray> colors =
    vtkSmartPointer<vtkUnsignedCharArray>::New();
  colors->SetNumberOfComponents(3);
  colors->SetName ("Colors");
  double c[3];
  unsigned char f[3];
  for(vtkIdType i = 0; i < pointsPolydataRGB->GetNumberOfPoints(); i++)
      {
      pointsPolydataRGB->GetPoint(i,c);
      f[0] = (unsigned char)c[0];
      f[1] = (unsigned char)c[1];
      f[2] = (unsigned char)c[2];
      colors->InsertNextTupleValue(f);
      }
  polydata->GetPointData()->SetScalars(colors);

  // Visualization
  vtkSmartPointer<vtkPolyDataMapper> mapper =
    vtkSmartPointer<vtkPolyDataMapper>::New();
#if VTK_MAJOR_VERSION <= 5
  mapper->SetInputConnection(polydata->GetProducerPort());
#else
  mapper->SetInputData(polydata);
#endif

  vtkSmartPointer<vtkActor> actor =
    vtkSmartPointer<vtkActor>::New();
```

```
    actor->SetMapper(mapper);
    actor->GetProperty()->SetPointSize(3);

    vtkSmartPointer<vtkRenderer> renderer =
     vtkSmartPointer<vtkRenderer>::New();

    renderer->AddActor(actor);
    renderer->SetBackground(.7, .7, .7); // Background color

    vtkwidget->GetRenderWindow()->AddRenderer( renderer );
    vtkwidget->update();

}

void MiVisor::domus()
{
  // Read the file
    vtkSmartPointer<vtkSimplePointsReader> reader =
     vtkSmartPointer<vtkSimplePointsReader>::New();
    reader->SetFileName("nubes/domusxyz.xyz");
    reader->Update();
    vtkPolyData* pointsPolydata = reader->GetOutput();
    for(vtkIdType i = 0; i < pointsPolydata->GetNumberOfPoints(); i++)
        {
        double p[3];
        pointsPolydata->GetPoint(i,p);
        }

    vtkSmartPointer<vtkSimplePointsReader> readerRGB =
     vtkSmartPointer<vtkSimplePointsReader>::New();
    readerRGB->SetFileName("nubes/domusrgb.xyz");
    readerRGB->Update();
    vtkPolyData* pointsPolydataRGB = readerRGB->GetOutput();
    double e;
    for(vtkIdType i = 0; i < pointsPolydataRGB->GetNumberOfPoints(); i++)
        {
        e = pointsPolydataRGB->GetNumberOfPoints();
        double p[3];
        pointsPolydataRGB->GetPoint(i,p);
        }
    cout << e << endl;

    vtkSmartPointer<vtkVertexGlyphFilter> vertexFilter =
     vtkSmartPointer<vtkVertexGlyphFilter>::New();
  #if VTK_MAJOR_VERSION <= 5
    vertexFilter->SetInputConnection(pointsPolydata->GetProducerPort());
  #else
```

```
  vertexFilter->SetInputData(pointsPolydata);
#endif
  vertexFilter->Update();

  vtkSmartPointer<vtkPolyData> polydata =
    vtkSmartPointer<vtkPolyData>::New();
  polydata->ShallowCopy(vertexFilter->GetOutput());

  // Setup colors

  vtkSmartPointer<vtkUnsignedCharArray> colors =
    vtkSmartPointer<vtkUnsignedCharArray>::New();
  colors->SetNumberOfComponents(3);
  colors->SetName ("Colors");
  double c[3];
  unsigned char f[3];
  for(vtkIdType i = 0; i < pointsPolydataRGB->GetNumberOfPoints(); i++)
      {
      pointsPolydataRGB->GetPoint(i,c);
      f[0] = (unsigned char)c[0];
      f[1] = (unsigned char)c[1];
      f[2] = (unsigned char)c[2];
      colors->InsertNextTupleValue(f);
      }
  polydata->GetPointData()->SetScalars(colors);

  // Visualization
  vtkSmartPointer<vtkPolyDataMapper> mapper =
    vtkSmartPointer<vtkPolyDataMapper>::New();
#if VTK_MAJOR_VERSION <= 5
  mapper->SetInputConnection(polydata->GetProducerPort());
#else
  mapper->SetInputData(polydata);
#endif

  vtkSmartPointer<vtkActor> actor =
    vtkSmartPointer<vtkActor>::New();
  actor->SetMapper(mapper);
  actor->GetProperty()->SetPointSize(3);

  vtkSmartPointer<vtkRenderer> renderer =
    vtkSmartPointer<vtkRenderer>::New();

  renderer->AddActor(actor);
  renderer->SetBackground(.7, .7, .7); // Background color

  vtkwidget->GetRenderWindow()->AddRenderer( renderer );
```

```
    vtkwidget->update();

}

void MiVisor::viviendas()
{
   // Read the file
    vtkSmartPointer<vtkSimplePointsReader> reader =
     vtkSmartPointer<vtkSimplePointsReader>::New();
    reader->SetFileName("nubes/viviendasxyz.xyz");
    reader->Update();
    vtkPolyData* pointsPolydata = reader->GetOutput();
    for(vtkIdType i = 0; i < pointsPolydata->GetNumberOfPoints(); i++)
        {
        double p[3];
        pointsPolydata->GetPoint(i,p);
        }

    vtkSmartPointer<vtkSimplePointsReader> readerRGB =
     vtkSmartPointer<vtkSimplePointsReader>::New();
    readerRGB->SetFileName("nubes/viviendasrgb.xyz");
    readerRGB->Update();
    vtkPolyData* pointsPolydataRGB = readerRGB->GetOutput();
    double e;
    for(vtkIdType i = 0; i < pointsPolydataRGB->GetNumberOfPoints(); i++)
        {
        e = pointsPolydataRGB->GetNumberOfPoints();
        double p[3];
        pointsPolydataRGB->GetPoint(i,p);
        }
    cout << e << endl;

    vtkSmartPointer<vtkVertexGlyphFilter> vertexFilter =
     vtkSmartPointer<vtkVertexGlyphFilter>::New();
  #if VTK_MAJOR_VERSION <= 5
    vertexFilter->SetInputConnection(pointsPolydata->GetProducerPort());
  #else
    vertexFilter->SetInputData(pointsPolydata);
  #endif
    vertexFilter->Update();

    vtkSmartPointer<vtkPolyData> polydata =
     vtkSmartPointer<vtkPolyData>::New();
    polydata->ShallowCopy(vertexFilter->GetOutput());

    // Setup colors
```

```
    vtkSmartPointer<vtkUnsignedCharArray> colors =
      vtkSmartPointer<vtkUnsignedCharArray>::New();
    colors->SetNumberOfComponents(3);
    colors->SetName ("Colors");
    double c[3];
    unsigned char f[3];
    for(vtkIdType i = 0; i < pointsPolydataRGB->GetNumberOfPoints(); i++)
        {
        pointsPolydataRGB->GetPoint(i,c);
        f[0] = (unsigned char)c[0];
        f[1] = (unsigned char)c[1];
        f[2] = (unsigned char)c[2];
        colors->InsertNextTupleValue(f);
        }
    polydata->GetPointData()->SetScalars(colors);

    // Visualization
    vtkSmartPointer<vtkPolyDataMapper> mapper =
      vtkSmartPointer<vtkPolyDataMapper>::New();
  #if VTK_MAJOR_VERSION <= 5
    mapper->SetInputConnection(polydata->GetProducerPort());
  #else
    mapper->SetInputData(polydata);
  #endif

    vtkSmartPointer<vtkActor> actor =
      vtkSmartPointer<vtkActor>::New();
    actor->SetMapper(mapper);
    actor->GetProperty()->SetPointSize(3);

    vtkSmartPointer<vtkRenderer> renderer =
      vtkSmartPointer<vtkRenderer>::New();

    renderer->AddActor(actor);
    renderer->SetBackground(.7, .7, .7); // Background color

    vtkwidget->GetRenderWindow()->AddRenderer( renderer );
    vtkwidget->update();

}

void MiVisor::todo()
{
  // Read the file
    vtkSmartPointer<vtkSimplePointsReader> reader =
      vtkSmartPointer<vtkSimplePointsReader>::New();
    reader->SetFileName("nubes/todoxyz.xyz");
```

```
  reader->Update();
  vtkPolyData* pointsPolydata = reader->GetOutput();
  for(vtkIdType i = 0; i < pointsPolydata->GetNumberOfPoints(); i++)
      {
      double p[3];
      pointsPolydata->GetPoint(i,p);
      }

  vtkSmartPointer<vtkSimplePointsReader> readerRGB =
    vtkSmartPointer<vtkSimplePointsReader>::New();
  readerRGB->SetFileName("nubes/todorgb.xyz");
  readerRGB->Update();
  vtkPolyData* pointsPolydataRGB = readerRGB->GetOutput();
  double e;
  for(vtkIdType i = 0; i < pointsPolydataRGB->GetNumberOfPoints(); i++)
      {
      e = pointsPolydataRGB->GetNumberOfPoints();
      double p[3];
      pointsPolydataRGB->GetPoint(i,p);
      }
  cout << e << endl;

  vtkSmartPointer<vtkVertexGlyphFilter> vertexFilter =
    vtkSmartPointer<vtkVertexGlyphFilter>::New();
#if VTK_MAJOR_VERSION <= 5
  vertexFilter->SetInputConnection(pointsPolydata->GetProducerPort());
#else
  vertexFilter->SetInputData(pointsPolydata);
#endif
  vertexFilter->Update();

  vtkSmartPointer<vtkPolyData> polydata =
    vtkSmartPointer<vtkPolyData>::New();
  polydata->ShallowCopy(vertexFilter->GetOutput());

  // Setup colors

  vtkSmartPointer<vtkUnsignedCharArray> colors =
    vtkSmartPointer<vtkUnsignedCharArray>::New();
  colors->SetNumberOfComponents(3);
  colors->SetName ("Colors");
  double c[3];
  unsigned char f[3];
  for(vtkIdType i = 0; i < pointsPolydataRGB->GetNumberOfPoints(); i++)
      {
      pointsPolydataRGB->GetPoint(i,c);
      f[0] = (unsigned char)c[0];
```

```
        f[1] = (unsigned char)c[1];
        f[2] = (unsigned char)c[2];
        colors->InsertNextTupleValue(f);
        }
    polydata->GetPointData()->SetScalars(colors);

    // Visualization
    vtkSmartPointer<vtkPolyDataMapper> mapper =
      vtkSmartPointer<vtkPolyDataMapper>::New();
  #if VTK_MAJOR_VERSION <= 5
    mapper->SetInputConnection(polydata->GetProducerPort());
  #else
    mapper->SetInputData(polydata);
  #endif

    vtkSmartPointer<vtkActor> actor =
      vtkSmartPointer<vtkActor>::New();
    actor->SetMapper(mapper);
    actor->GetProperty()->SetPointSize(3);

    vtkSmartPointer<vtkRenderer> renderer =
      vtkSmartPointer<vtkRenderer>::New();

    renderer->AddActor(actor);
    renderer->SetBackground(.7, .7, .7); // Background color

    vtkwidget->GetRenderWindow()->AddRenderer( renderer );
    vtkwidget->update();

}

void MiVisor::atributos()
{
  QTableWidget *tableWidget = new QTableWidget;
  tableWidget->show();
  QStringList myString;
  tableWidget->clear();
  tableWidget->setColumnCount(mypLayer16->pendingFields().count());
  tableWidget->setRowCount(mypLayer16->pendingFeatureCount());
     for (int i=0;i<mypLayer16->pendingFields().count();i++)
     {
         myString<<mypLayer16->pendingFields().field(i).name();
     }
  tableWidget->setHorizontalHeaderLabels(myString);
//   tableWidget->setWindowState(Qt::WindowMaximized);
  tableWidget->setMinimumSize(QSize(650,200));
  tableWidget->setWindowTitle("Atributos");
```

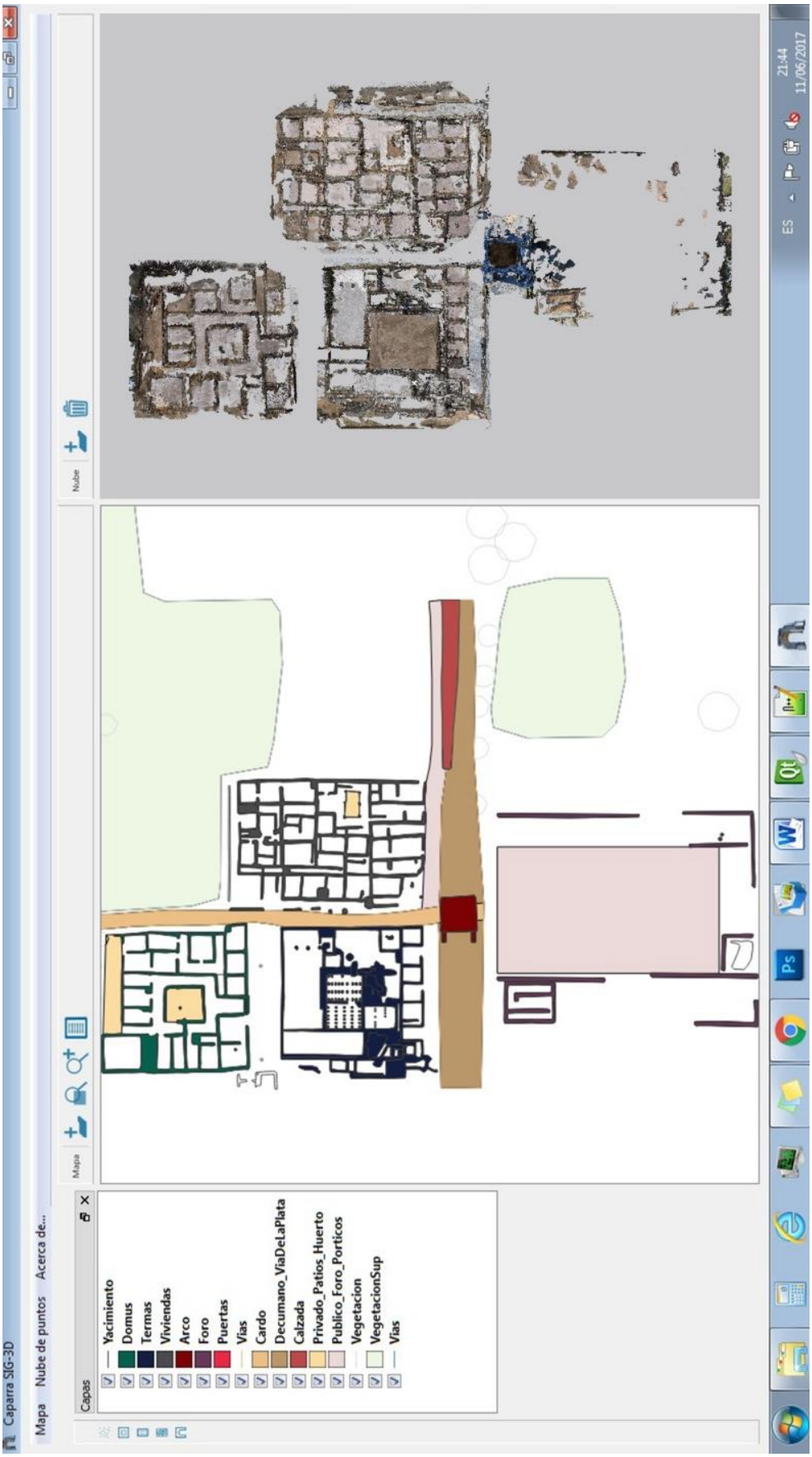
```
   QString string1 = "C:/Qt/Projects/VisorShpXYZ-build/shp/";
   bool ok;
   QString string2 = QInputDialog::getText(this, tr("Seleccionar capa"),
            tr("Indroduzca el nombre de la capa \n*se diferencian mayusculas y
minusculas:"),
            QLineEdit::Normal, tr(""), &ok);
   string1.append(string2);
   string1.append("/");
   string1.append(string2);
   string1.append(".txt");
   qDebug() << string1 << endl;
   QFile file(string1);

   int fcount=mypLayer16->featureCount();
   int hcount=mypLayer16->pendingFields().count();
   int i=0;
   int j;
//    QString filename = QFileDialog::getOpenFileName();
//    QFile file(filename);
       if (!file.open(QIODevice::ReadOnly | QIODevice::Text))
       return;

       for(i=-1;i<fcount;i++){
           QString content = file.readLine();
           QRegExp tagExp("\t");
           QStringList firstList = content.split(tagExp);
           for(j=0;j<firstList.count();j++){
//                cout<<firstList[j].toStdString()<<endl;
               QTableWidgetItem *item = new QTableWidgetItem(firstList[j]);
               tableWidget->setItem(i,j,item);
           }
           content.clear();
           firstList.clear();
       }

      file.close();

}

void MiVisor::abrirNube()
{
   QString myFileName = QFileDialog::getOpenFileName(0,"Abrir","","(*.xyz)");
   QString str1 = myFileName;
   QByteArray ba = str1.toLatin1();
   const char *c_str2 = ba.data();
   printf("str2: %s", c_str2);
```

```
  QString myFileName2 = QFileDialog::getOpenFileName(0,"Abrir archivo
rgb","","(*.xyz)");
  QString str3 = myFileName2;
  QByteArray ba3 = str3.toLatin1();
  const char *c_str4 = ba3.data();
  printf("str2: %s", c_str4);

  // Read the file
   vtkSmartPointer<vtkSimplePointsReader> reader =
    vtkSmartPointer<vtkSimplePointsReader>::New();
   reader->SetFileName(c_str2);
   reader->Update();
   vtkPolyData* pointsPolydata = reader->GetOutput();
   for(vtkIdType i = 0; i < pointsPolydata->GetNumberOfPoints(); i++)
      {
      double p[3];
      pointsPolydata->GetPoint(i,p);
      }

   vtkSmartPointer<vtkSimplePointsReader> readerRGB =
    vtkSmartPointer<vtkSimplePointsReader>::New();
   readerRGB->SetFileName(c_str4);
   readerRGB->Update();
   vtkPolyData* pointsPolydataRGB = readerRGB->GetOutput();
   double e;
   for(vtkIdType i = 0; i < pointsPolydataRGB->GetNumberOfPoints(); i++)
      {
      e = pointsPolydataRGB->GetNumberOfPoints();
      double p[3];
      pointsPolydataRGB->GetPoint(i,p);
      }
   cout << e << endl;

   vtkSmartPointer<vtkVertexGlyphFilter> vertexFilter =
    vtkSmartPointer<vtkVertexGlyphFilter>::New();
  #if VTK_MAJOR_VERSION <= 5
   vertexFilter->SetInputConnection(pointsPolydata->GetProducerPort());
  #else
   vertexFilter->SetInputData(pointsPolydata);
  #endif
   vertexFilter->Update();

   vtkSmartPointer<vtkPolyData> polydata =
    vtkSmartPointer<vtkPolyData>::New();
   polydata->ShallowCopy(vertexFilter->GetOutput());
```

```
    // Setup colors

    vtkSmartPointer<vtkUnsignedCharArray> colors =
      vtkSmartPointer<vtkUnsignedCharArray>::New();
    colors->SetNumberOfComponents(3);
    colors->SetName ("Colors");
    double c[3];
    unsigned char f[3];
    for(vtkIdType i = 0; i < pointsPolydataRGB->GetNumberOfPoints(); i++)
        {
        pointsPolydataRGB->GetPoint(i,c);
        f[0] = (unsigned char)c[0];
        f[1] = (unsigned char)c[1];
        f[2] = (unsigned char)c[2];
        colors->InsertNextTupleValue(f);
        }
    polydata->GetPointData()->SetScalars(colors);

    // Visualization
    vtkSmartPointer<vtkPolyDataMapper> mapper =
      vtkSmartPointer<vtkPolyDataMapper>::New();
  #if VTK_MAJOR_VERSION <= 5
    mapper->SetInputConnection(polydata->GetProducerPort());
  #else
    mapper->SetInputData(polydata);
  #endif

    vtkSmartPointer<vtkActor> actor =
      vtkSmartPointer<vtkActor>::New();
    actor->SetMapper(mapper);
    actor->GetProperty()->SetPointSize(3);

    vtkSmartPointer<vtkRenderer> renderer =
      vtkSmartPointer<vtkRenderer>::New();

    renderer->AddActor(actor);
    renderer->SetBackground(.7, .7, .7); // Background color

    vtkwidget->GetRenderWindow()->AddRenderer( renderer );
    vtkwidget->update();

}

void MiVisor::cerrarNube()
{
  vtkSmartPointer<vtkRenderer> renderer =
    vtkSmartPointer<vtkRenderer>::New();
```

```
    renderer->SetBackground(.7, .7, .7); // Background color

    vtkwidget->GetRenderWindow()->AddRenderer( renderer );
    vtkwidget->update();
}

void MiVisor::acercaDe()
{
    QWidget *acerca = new QWidget;
    acerca->setWindowTitle("Acerca de Caparra SIG-3D");
    QVBoxLayout *box = new QVBoxLayout;
    acerca->setLayout(box);
    QTextEdit *text = new QTextEdit;
    text->setText("TFM MASTER EN GEOTECNOLOGIAS CARTOGRAFICAS EN INGENIERIA
Y ARQUITECTURA. UNIVERSIDAD DE SALAMANCA. "
                  "Objetivo: Documentacion y levantamiento fotogrametrico del yacimiento
arqueologico de Caparra. "
                  "Desarrollo de una aplicacion SIG 3D e inclusion de los datos obtenidos. "
                  "Autor: CRISTINA TEJEDA SANCHEZ");
    text->setAlignment(Qt::AlignJustify);
    text->setFixedWidth(300);
    box->addWidget(text);
    acerca->show();
}
```

ANEXO II. INTERFAZ GRÁFICA DE LA APLICACIÓN



ANEXO III. NUBE DE PUNTOS DENSA

ANEXO IV. ALZADO, SECCIÓN Y PLANTA DEL ARCO

ANEXO V. POSIBLE RECONSTRUCCIÓN 3D. VISTAS DESDE EL INTERIOR DE LAS TERMAS Y EL FORO