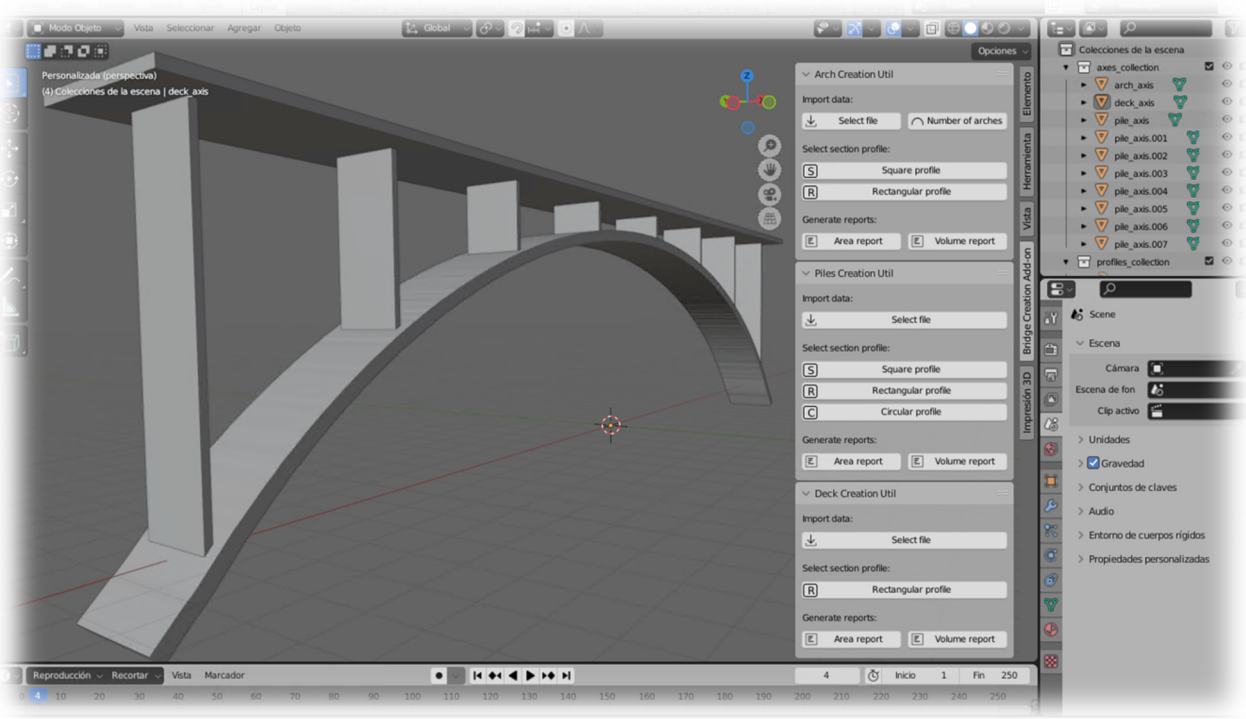
Universidad de La Laguna

PROCESADO GEOMÉTRICO DE TIPOS ESTRUCTURALES USANDO TECNOLOGÍA BIM



Escuela de Doctorado y Estudios de Posgrado

Máster en Gestión e Innovación Tecnológica en la Construcción

Trabajo Fin de Máster

Autora: Dña. María Gara Acosta González

Tutora: Dra. Norena Natalia Martín Dorta

Co-tutores: Dr. Alejandro Mateo Hernández Díaz
Dña. Ana Pérez García

Septiembre 2022

Dña. Norena Natalia Martín Dorta, con DNI 78674114S, profesora del Área de Expresión Gráfica en la Ingeniería, del Departamento de Técnicas y Proyectos en Ingeniería y Arquitectura de la Universidad de La Laguna, responsable del BIMLab ULL y miembro del FabLab ULL, D. Alejandro Mateo Hernández Díaz, con DNI 78559539A, profesor en excedencia del Área de Mecánica de Medios Continuos y Teoría de las Estructuras, del Departamento de Técnicas y Proyectos en Ingeniería y Arquitectura de la Universidad de La Laguna y Jefe de Estudios e Hidrología de la Dirección General de Aguas del Gobierno de Canarias y Dña. Ana Pérez García, con DNI 79065310G, personal investigador en formación, del Departamento de Técnicas y Proyectos en Ingeniería y Arquitectura de la Universidad de La Laguna,

HACEN CONSTAR

Que la presente memoria titulada:

"Procesado geométrico de tipos estructurales usando tecnología BIM"

Ha sido realizada bajo sus direcciones por María Gara Acosta González, con DNI 42186363Q.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 7 de septiembre de 2022.

Fdo. Norena Natalia Martín Dorta

Fdo. Alejandro Mateo Hernández Díaz

Fdo. Ana Pérez García

Agradecimientos

A mi tutora del Trabajo Fin de Máster, Norena Natalia Martín Dorta, por su responsabilidad y dedicación durante todos estos meses de realización del trabajo, incluso durante su periodo de vacaciones. Su constancia y apoyo continuo y su admirable diligencia han sido valores clave para la elaboración final del mismo.

A mi co-tutor del Trabajo Fin de Máster, Alejandro Mateo Hernández Díaz, por entender la enseñanza como un compromiso social que va más allá del conocimiento que se pueda compartir. Por su creatividad y entusiasmo y, sobre todo, por los extraordinarios valores humanos demostrados durante la realización de este trabajo. Sin sus profesionales juicios y consejos la culminación de este trabajo no hubiera sido posible. Excelente profesional y mejor persona.

A Ana Pérez García, por su aportación en la programación del código que ha permitido la materialización de las ideas que constituyen la razón de ser de este trabajo.

A mi gran amigo Gabriel Martín de Lorenzo Cáceres, ingeniero de caminos, canales y puertos, ideólogo y director de algunas de las obras que se han tomado como referencia para la elaboración de este trabajo.

A mis queridos padres y mi querido hermano Pedro, a quienes dedico este trabajo y agradezco todo su cariño y apoyo incondicional tanto en lo personal como en lo laboral durante este intenso pero apasionante año académico. Gracias por la paciencia y comprensión mostradas. Son las personas que más admiro y quiero.

Resumen

La tecnología Building Information Modeling (en lo sucesivo, BIM) supone la evolución de los sistemas tradicionales de diseño de un proyecto basados en *el plano*.

Uno de los campos de la ingeniería civil en los que la incidencia de dicha tecnología es más significativa es el de la ingeniería estructural, al permitir el uso de estándares abiertos (formato IFC, Industry Foundation Classes) y proporcionar un enfoque universal al diseño colaborativo y abierto que permite el intercambio de datos entre distintos softwares BIM.

Blender es un programa de código abierto basado en la animación, renderizado y modelado en 3D, que además ofrece, como posibilidad excepcional, la programación de código en Python a fin de explorar nuevas funcionalidades del programa en materia de modelado geométrico.

A fin de explorar y validar las anteriores potencialidades, así como su aplicación concreta a sistemas de ingeniería civil soportados sobre estructuras relativamente complejas desde el punto de vista de su definición geométrica - tales como puentes arco, presas arco o túneles-, el presente trabajo se ha focalizado en el modelado geométrico de estructuras con curvatura (a excepción de la torsión). En este sentido, se ha adoptado el *arco tridimensional* como tipo estructural esencial de referencia.

A partir de la definición geométrica de la directriz de un arco de referencia se desarrolla un código que permite el procesado automático de estructuras curvas, así como la obtención de los atributos y propiedades asociados a esas geometrías de forma igualmente automatizada. Por otra parte, y a fin de ilustrar el proceso de ensamblaje y automatización de una estructura completa con las características anteriormente mencionadas, se ha desarrollado el modelado geométrico completo de un *puente arco*.

En el presente Trabajo Fin de Máster se valida la aplicabilidad del software BLENDER en el modelado geométrico de estructuras tipo arco de uso frecuente en sistemas de ingeniería civil.

Palabras clave: BIM, cálculo de estructuras, IFC, Python, blender, ingeniería civil, puente arco.

Abstract

Building Information Modeling technology (from now on, BIM) is the evolution of traditional plan-based project design systems.

One of the fields of civil engineering where the impact of this technology is most significant is structural engineering, as it enables the use of open standards (IFC format, Industry Foundation Classes) and provides a universal approach to open and collaborative design that allows the exchange of data between different BIM software.

Blender is an open source programme based on animation, rendering and 3D modelling, which also offers, as an exceptional possibility, the programming of code in Python in order to explore new functionalities of the programme in terms of geometric modelling.

In order to explore and validate the above potentialities, as well as their concrete application to civil engineering systems supported on relatively complex structures from the point of view of their geometric definition -such as arch bridges, arch dams or tunnels-, the present work has focused on the geometric modelling of structures with curvature (with the exception of torsion). In this sense, the three-dimensional arch has been adopted as the essential structural type of reference.

Based on the geometric definition of the directrix of a reference arc, a code is developed that allows the automatic processing of curved structures, as well as obtaining the attributes and properties associated with these geometries in an equally automated way. On the other hand, and in order to illustrate the assembly and automation process of a complete structure with the aforementioned characteristics, the complete geometric modelling of an arch bridge has been developed.

This Master's thesis validates the applicability of BLENDER software in the geometric modelling of arch-type structures frequently used in civil engineering systems.

Keywords: BIM, structural design, IFC, Python, blender, civil engineering, arch bridge.

ÍNDICE

Pág.

1. INTRODUCCIÓN

1.1. Antecedentes

El proceso de cálculo de todo sistema de ingeniería civil comienza con la definición del dominio geométrico del mismo, como aquella región del espacio o volumen sobre el que posteriormente se aplican unos esfuerzos que permiten registrar unas tensiones como consecuencia de los mismos.

Un modelo es una representación de algunas de las características de una entidad concreta o abstracta [1], mientras que un modelo geométrico describe componentes con propiedades geométricas inherentes. Entre sus características destacan su estructura espacial, la conectividad entre elementos y las propiedades asociadas a componentes espaciales.

El modelo geométrico debe contener toda la información necesaria para representar el tipo estructural que se pretende diseñar, permitiendo realizar todas las operaciones requeridas sobre el modelo: edición, visualización, cálculo de propiedades y simulaciones. El tipo estructural que se pretende diseñar condiciona los datos que debe contener el modelo geométrico. Dichos modelos geométricos así definidos describen la geometría de los sistemas estructurales de la forma más precisa posible, con el objeto de que los ingenieros de estructuras puedan trabajar de forma eficaz con ellos.

Los problemas estructurales no se consideran enteramente definidos sino se introducen ciertas condiciones en los contornos. Estas condiciones de contorno se definen en función de las fuerzas de reacción o de los desplazamientos. De esta forma, un apoyo simple permite el movimiento en la dirección del eje x y el giro en el plano x-y, y la reacción es una fuerza perpendicular al eje x. Para el caso de un apoyo doble, el desplazamiento está impedido en el eje x y en el eje y, y solo se permite el giro. Las reacciones aparecen en las direcciones de los ejes x e y. En el empotramiento el giro también está coaccionado.

El dominio geométrico del sistema estructural se acompañará, a su vez, de las propiedades del material constitutivo del mismo. Las propiedades mecánicas determinan el comportamiento de un determinado material ante las fuerzas que se le aplican y definen la relación entre su respuesta a una carga y la deformación que sufre.

1.2. Objetivos

Este trabajo tiene por objeto abordar la automatización del procesado de la información geométrica en el contexto de los tipos estructurales de aplicación frecuente en ingeniería civil, aprovechando las capacidades que ofrece la tecnología BIM y el software Blender usando el lenguaje de alto nivel Python.

Para ello, se procede a la programación de un código que permite leer un conjunto de datos de entrada desde un archivo externo en formato .CSV, que los transforma en una geometría representada a través de una interfaz gráfica. A esa geometría generada se le asignará, además, una clase IFC a través de la librería ifcOpenShell.

En contexto BIM, hasta la fecha este procedimiento consistía en la definición de la geometría de un sistema estructural a través de un software BIM para, posteriormente, y mediante el uso de otros programas, obtener una serie de variables o características de interés del mismo, tales como el volumen de la geometría generada o el coste de su encofrado, lo que permitiría realizar cálculos estructurales, de plazos o, incluso, de procesos constructivos.

En este trabajo, el procedimiento propuesto permite la integración en un software opensource, a través de un código programado con Python, no solo de la definición de geometrías complejas de los sistemas estructurales más representativos sino además de la obtención de determinadas características y atributos de los mismos, de forma totalmente automatizada. La efectividad de este procedimiento permite que la obtención de dichos atributos sea más ágil al prescindir de otros programas necesarios a tal fin. Los diferentes procesos así implementados se validan mediante su aplicación en el diseño y planificación de unidades concretas integradas en proyectos de ingeniería estructural, con el posterior objeto de incorporar este proceso en el ámbito de la optimización estructural.

Para el desarrollo de esta Trabajo Fin de Máster se ha seleccionado “el arco” como tipo estructural esencial para demostrar la aplicabilidad del uso del software Blender en los sistemas más representativos de la Ingeniería Civil, tales como los puentes arco, los túneles y las presas arco.

En la siguiente imagen se muestra el puente arco de hormigón de Sandö, que cruza el río Ångermanälven en el municipio Kramfors, en la provincia de Ångermanland, en el norte de Suecia. Tiene una luz libre de 264 m y se encuentra a 42 m sobre el agua. El puente fue inaugurado en 1943 y hasta 1964, era el mayor arco de hormigón en el mundo.

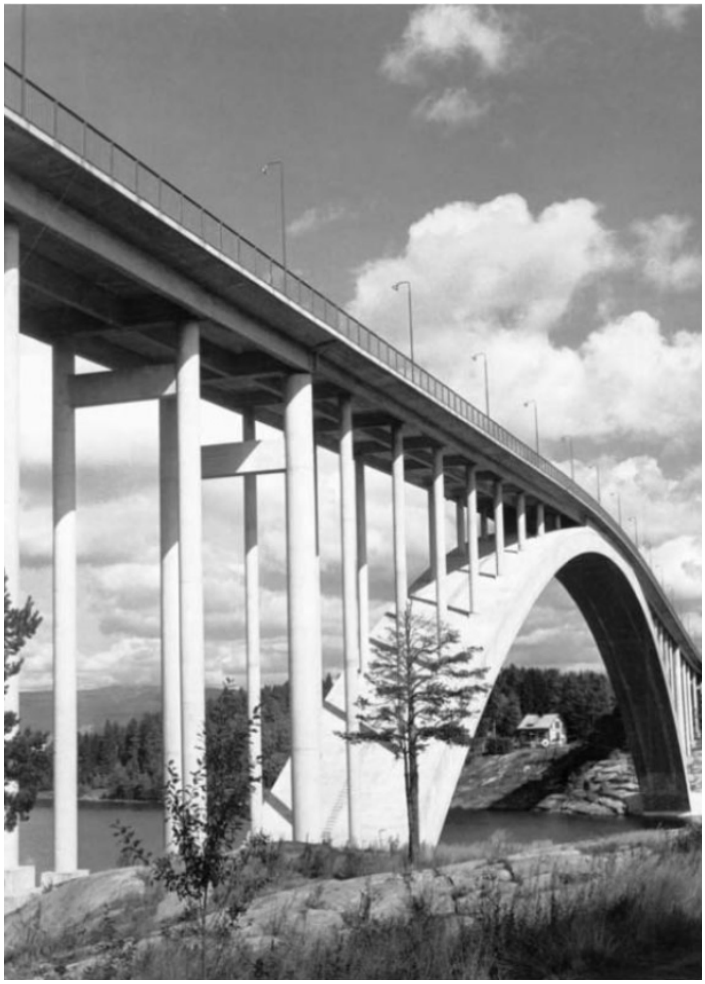


Figura 1. Puente de Sandö. Fotografía: K. W. Gullers. Razón y ser de los tipos estructurales. Eduardo Torroja Miret, 2007

En la siguiente Figura se muestra una imagen de un tramo de carretera de interés regional que forma parte del anillo insular de la isla de Tenerife, concretamente el tramo comprendido entre los municipios de El Tanque e Icod de los Vinos.

Este tramo tiene una longitud aproximada de unos 12 kilómetros de nueva construcción y fue ejecutado por la empresa constructora Ferrovial Agromán. Con el objeto de minimizar y evitar la afección a zonas ecológicamente protegidas, resultó necesario diseñar elementos singulares que contribuyeran a reducir el impacto sobre el territorio y el medio ambiente, tales como falsos túneles que evitan la formación de grandes superficies de talud en los desmontes generados.

Figura 2. Falsos túneles en el tramo de carretera comprendido entre El Tanque e Icod de los Vinos. Recuperado de https://blog.ferrovial.com/es/2014/11/construccion-y-proteccion-del-entorno-natural-de-la-mano-en-canarias/

En la Figura que se muestra a continuación puede observarse una imagen de la presa de Aldeadávila, todo un hito de la ingeniería civil española.

Está construida en el curso medio del río Duero, entre España y Portugal, a 7 km de la localidad de Aldeadávila de la Ribera, en la provincia de Salamanca. La presa es un arco de gravedad de hormigón de 139,40 m de altura. Dispone de un aliviadero de superficie con 8 compuertas de segmento de 14 m x 8,30 m. Además, posee un túnel aliviadero con 2 compuertas tipo segmento de 12,50 m x 9,70 m. Constituye la central hidroeléctrica más importante de España a nivel de potencia instalada y de producción.

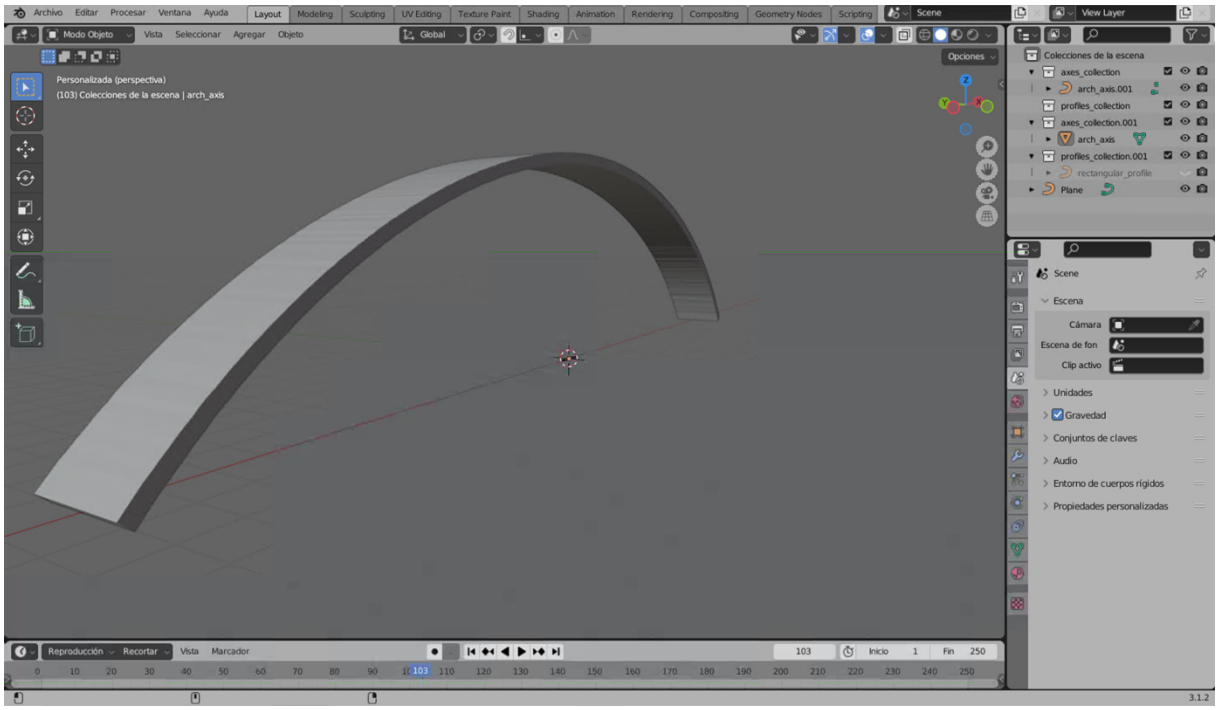
Fue construida ente los años 1958 y 1965 y constituye una de las presas más emblemáticas de la ingeniería de presas, tanto a nivel español como a nivel mundial.

Figura 3. Presa de Aldeadávila. Recuperado de https://www.iagua.es/data/infraestructuras/presas/aldeadavila

1.3. Metodología

La directriz del arco se define mediante un conjunto de coordenadas (x, z) que se incorporan en un archivo externo .CSV que el código programado es capaz de leer y procesar con el objeto de transformarlos en la geometría que se muestra en la Figura 4.

Para ello, la consola del programa permite seleccionar entre varias opciones, tales como el tipo de sección transversal del arco (cuadrada, rectangular o circular), sus características geométricas (ancho, canto de su sección, diámetro), el número de arcos a representar y la equidistancia entre ellos.



(a)

```
Read prefs: C:\Users\Usuario2\AppData\Roaming\Blender Foundation\Blender\3.1\config\userpref.blend
Read blend: C:\Users\Usuario2\Desktop\Gara_TFM\scriptVerticeCreationFromCSV_6.blend
Indique el tipo de sección del arco ("sq" para cuadrada; "r" para rectangular; "c" para circular): r
Indique el valor del ancho unitario de la sección: 2
Indique el valor del canto de la sección: .3
Indique el número de arcos que desea dibujar: 1
Info: Removidos: 0 vértices, 0 bordes, 0 caras
Info: 8 vértices removidos
Info: Removidos: 0 vértices, 0 bordes, 0 caras
Info: Modified: -8 vertices, -8 edges, +0 faces
Volumen de un arco: 19.93 m3
Volumen total: 19.93 m3
```

(b)

Figura 4. Extracto de código con la generación automática de un arco de sección transversal rectangular, de 2,00 m de ancho y 0,30 m de canto y con un volumen igual a 19,93 m^3 (a) y consola de Python para dicha generación (b). Captura de pantalla. Fuente: elaboración propia

Dado que el arco trabaja fundamentalmente a compresión, los materiales pétreos se prestan bien a su construcción. Para el caso de arcos más expuestos a flexiones, se emplea el hormigón armado e incluso el acero para salvar grandes luces aprovechando las altas resistencias de estos materiales, tanto a la tracción como a la compresión. De esta forma, con arcos de hormigón armado se han llegado a los 264 metros de luz, mientras que con arcos de acero se han sobrepasado los 500 metros.

Por otro lado, el arco genera empujes horizontales sobre los apoyos, tal y como se ha mencionado con anterioridad. Si el arco está empotrado, los empujes son algo menores ya que la flexión contribuye al trabajo resistente, pero la solución estructural se encarece. Por tanto, el arco requiere siempre una buena cimentación o unos buenos contrarrestos para soportar estos empujes. Los arcos triarticulados presentan la ventaja de no producir esfuerzos, ni por variaciones térmicas o higroscópicas, ni por deformación lenta. Los arcos biarticulados son más deformables que el empotrado y pueden reducir las variaciones térmicas dentro de los límites aceptables.

En cuanto a la sección clásica de los arcos de hormigón armado, es necesario recurrir a secciones en cajón rectangular, cuando las luces son excepcionalmente importantes. La sección de los arcos de fábrica es la rectangular, donde el ancho viene impuesto por el peligro de pandeo. En arcos de acero se requiere casi siempre ir a secciones compuestas para aumentar la rigidez al pandeo sin exagerar inútilmente el peso del material utilizado.

La siguiente Tabla muestra un resumen de las características geométricas y de los materiales constitutivos del arco objeto de estudio:

Tabla 1. Resumen de las características geométricas y de los materiales constitutivos del arco tipo

Luz (m)	Flecha (m)	Ancho (m)	Canto (m)	Peso específico (N/m3)	Módulo de elasticidad (N/m2)	Coeficiente de Poisson
30,0	6,00	1,00	0,15	2.400	27.664,49E6	0,3

2. BIM EN INGENIERÍA CIVIL

Se procede a continuación a realizar una revisión de la literatura más relevante a través de una búsqueda bibliográfica en relación con la aplicabilidad de la tecnología BIM en el ámbito de la ingeniería civil, y más concretamente en el cálculo de las estructuras, clasificando la literatura encontrada en tres grandes grupos: II.1) hitos históricos que fundamentan el desarrollo de una nueva metodología de trabajo a través de BIM, II.2) integración del análisis estructural en la ingeniería civil con la tecnología BIM y, por último, II.3) la aplicabilidad de Blender en la ingeniería civil.

2.1. BIM en la Ingeniería Civil

Entre los hitos más importantes que han definido el desarrollo de la tecnología BIM desde sus orígenes, destacan a continuación los siguientes tomando como base de referencia a [2]:

En el año 1957 Patrick J. Hanratty, conocido como el padre del CAM/CAD, desarrolló el primer programa comercial con tecnología CAM (Computer Aided Manufacturing) denominado PRONTO (Program for Numerical Tooling Operations). Se basaba en un lenguaje de computadoras denominado NC (Numerical Control), desarrollado principalmente por la empresa General Electric. La principal innovación fue la automatización de tareas repetitivas por medio de la tecnología, programando los movimientos de las herramientas de mecanizado de piezas.

En el año 1962 Douglas Engelbart, conocido como el inventor del ratón para los ordenadores, presenta su visión del arquitecto del futuro [3], adoptando un enfoque nuevo y sistemático para mejorar la eficacia intelectual del ser humano, aprovechando la asistencia directa en línea e integrada con nuevos conceptos y métodos que ofrece la computadora. Este marco conceptual que analiza el sistema compuesto por el individuo y las herramientas genera el primer acercamiento al concepto que hoy en día conocemos como BIM.

En el año 1963 Ivan Sutherland, conocido por ser considerado el padre de la computación gráfica, desarrolló el primer software de CAD (Computer Aided Design) denominado Sketchpad. Este programa permitía al usuario interactuar en una interfaz gráfica al utilizar un ordenador para escribir sobre una pantalla usando un lápiz óptico.

Las principales innovaciones fueron la posibilidad de dibujar sobre un monitor puntos, segmentos de líneas y arcos, y duplicar estos objetos manteniendo una relación con sus originales (concepto fundamental para los sistemas BIM).

En el año 1964 nuevamente Patrick J. Hanratty desarrolla DAC (Design Augmented by Computer). La aportación más relevante de este programa fue un sistema de tiempo compartido, lo cual implicaba que cuando un usuario no interactuaba con el equipo, el uso de ese recurso era cedido a otro usuario. Con el paso del tiempo, este concepto se convirtió en uno de los pilares de los sistemas BIM avanzados.

En al año 1972 se desarrolla un programa para análisis de modelos 3D, denominado Syntha Vision de MAGi (Mathematical Applications Group, Inc.). Este programa dio paso al desarrollo del CSG (Constructive Solid Geometry), que posteriormente sería utilizado por los sistemas CAD para la generación de modelos tridimensionales.

En el año 1975 Charles M. Eastman, quien es considerado el padre de BIM, publica un artículo de investigación para la Carnegie Mellon University [4]. Este artículo describe un prototipo funcional denominado BDS (Building Description System) que desarrolla ideas para trabajar con diseños paramétricos. No solo plantea el uso de un modelo tridimensional que a su vez es una base de datos integrada, sino que además define la estructura de datos que un determinado elemento debe tener para cumplir con el enfoque del BDS.

En el año 1977 la compañía Avions Marcel Dassault desarrolla el primer software de CAD que se enfoca en el modelado 3D de superficies complejas, denominado CATIA (Computer Aided Three Dimensional Interactive Application).

En el año 1979 General Electric y el National Bureau of Standars implementan IGES (Initial Graphic Exchange Standar), como respuesta a la necesidad inminente de emplear alguna forma de estandarización en el empleo de los sistemas CAD. IGES contribuyó en la interoperabilidad entre varios programas para el intercambio de superficies 3D complejas, y hoy en día continúa siendo un formato de transferencia de datos en las herramientas de software CAD.

En el año 1982 John Walker, junto con otros doce cofundadores, fundan la compañía Autodesk, y crean la versión 1 de AutoCAD. Ese mismo año Gábor Bojár funda la compañía Graphisoft y comienza a desarrollar **ArchiCAD**, el cual se convirtió en el primer software de CAD de la categoría BIM disponible en un ordenador personal. En 1984 se lanza la primera versión de ArchiCAD, bajo el nombre Radar CH, únicamente para ordenadores Apple Lisa.

En el año 1984 la empresa Allplan del Grupo Nemetscheck, fundada por Georg Memetschek, desarrolla **Allplan**, convirtiéndose en el segundo software de CAD de la categoría BIM disponible en un ordenador personal.

En el año 1985 Richard Diehl funda la compañía Graphsoft (que más tarde cambia su nombre a Diehl Graphsoft debido a su similitud con la empresa Graphisoft), y desarrolla el software denominado **MiniCAD** diseñado para Apple Macintosh, convirtiéndose en el tercer software de CAD de la categoría BIM disponible en un ordenador personal. La compañía experimenta una última actualización de nombre con el cambio a Windows, cuando se convirtió en Vectorworks.

En el año 1986 Robert Aish publica un artículo [5] en el que se usa por primera vez el térmico "Building Modeling" vinculado al concepto que hoy en día conocemos como BIM.

En el año 1993 Graphisoft desarrolla la primera versión de ArchiCAD para Windows, convirtiéndose en el primer software CAD-BIM multiplataforma.

En el año 1994 Autodesk promovió la formación de un consorcio para la construcción de un conjunto de clases que estandarizaran el desarrollo de aplicaciones de software. Inicialmente, el consorcio tomó el nombre de International Alliance for Interoperability (IAI) y a partir de su trabajo se establecieron las bases para la definición de las Clases Fundamentales para la Industria (en lo sucesivo Industry Foundation Classes, IFC) de la Arquitectura, la Ingeniería y la Construcción (AEC). En la actualidad, la alianza se llama buildingSMART y tiene por objetivo mejorar el intercambio de información entre los programas usados en el sector de la construcción.

En el año 1996 Diehl Graphsoft desarrolla la versión 6 de MiniCAD, disponible tanto para Windows como para Mac, convirtiéndose en el segundo software CAD-BIM multiplataforma. Comienza a funcionar el Consorcio Industrial IAI que asesora el desarrollo de aplicaciones integradas.

En el año 1998 Ton Roosendaal fundó la compañía Not a Number (NaN), como una derivada del estudio de animación holandés NeoGeo, con el objeto de comercializar y desarrollar la herramienta de modelado y animación 3D que conocemos como **Blender** [6].

En el año 2000 la compañía Charles River Software, fundada por Leonid Raiz e Irwin Jungreis en el año 1997, cambia su nombre a Revit Technology Corporation y lanza al mercado la primera versión del software **Revit**.

En el año 2002, Autodesk adquiere la compañía lo cual permitió más investigación, desarrollo y mejora del software, ofreciendo un verdadero programa tipo BIM con una base de datos relacional que actualiza a la vez el modelo y la información de todos los componentes del proyecto. En este mismo año se crea el primer proyecto BIM integrado en Finlandia.

En el año 2006 se lleva a cabo el primer proyecto IPD (Integrated Project Delivery) en Estado Unidos.

En el año 2007 se crean las primeras guías que hay que seguir para llevar a cabo un proyecto BIM. En este sentido, la GSA (General Services Administration) del gobierno de los Estados Unidos, creó la serie de guías BIM con el propósito de proporcionar orientación y requisitos a los diferentes equipos de proyectos. En Finlandia, la empresa gubernamental Senate Properties, responsable de la gestión de edificios públicos, estableció que todos los proyectos nuevos o de renovación por encima de 1.000.000 € debían ejecutarse utilizando tecnología BIM y se redactaron a su vez las primeras guías para la implementación de esta metodología. La confederación de constructores finlandeses exigió que todos los paquetes de software de diseño tuvieran la certificación IFC.

A partir del año 2010, Autodesk introduce la interfaz gráfica Ribbon en Revit, la cual sustituye las clásicas barras de menús por pestañas grandes, donde se almacenan las distintas herramientas del programa. Con la introducción de la interfaz Ribbon, se añaden nuevos iconos, cinta de opciones y la ventana de propiedades para editar los parámetros del proyecto. El Gobierno de Reino Unido anuncia los requisitos para la implantación.

En el año 2011, el gobierno de Reino Unido publicó un informe [7] en relación con su intención de requerir la implantación de BIM en todos los proyectos de construcción a partir del año 2016.

En el año 2012 Finlandia publica los requerimientos BIM comunes a nivel nacional. A partir de este año, los edificios públicos daneses deben ser desarrollados con metodología BIM.

En el año 2013 en Qatar, Arabia Saudí y Kuwait la tecnología BIM se convierte en un requisito del cliente.

En el año 2015, países como España adoptan hojas de ruta para la implantación de la metodología BIM.

En el año 2016 Reino Unido hace obligatoria la implantación de la metodología BIM en los proyectos de obras públicas. El artículo 22.4 de la Directiva vigente en materia de contratación pública [8] establece que los Estados miembros podrán exigir en sus procedimientos de contratación de obras y servicios el uso de herramientas de modelado digital de la información de la construcción (BIM) o herramientas similares, a más tardar el 18 de abril de 2016. En este sentido, en Reino Unido, Países Bajos, Dinamarca, Finlandia y Noruega ya requieren el uso de BIM en proyectos de construcción financiados con fondos públicos.

En Francia, en el año 2017 el ministro de territorio, igualdad y vivienda, Cècile Duflot, declaró que se requeriría la adaptación gradual de la tecnología BIM.

En España, en el año 2018 se impone el uso obligatorio de BIM en proyectos de Licitaciones Públicas de Edificación.

A nivel nacional, se ha creado la Comisión BIM, que es un órgano colegiado cuya misión principal es impulsar y garantizar la coordinación de todos los agentes implicados (administraciones, ingenierías, constructoras, universidades, profesionales, etc.) en la implantación de la metodología BIM en la contratación pública en España. De acuerdo con esta comisión, se propone una implantación progresiva de esta metodología de trabajo que se adapte a las capacidades tanto de las diferentes administraciones (entidades contratantes) como del sector en su conjunto, del cual forman parte los licitadores. Estos requisitos vendrán definidos dentro del ámbito que supone plantear unos objetivos a alcanzar a través del empleo de BIM, en función de los cuales se establecerán los usos BIM por fases.

En relación con la incorporación de la metodología BIM en los pliegos de licitación de obras públicas, se establecen tres niveles en función de la progresividad de su inclusión en los requerimientos en las licitaciones:

- BIM introducido como mejora (inclusión voluntaria de la metodología BIM como una mejora).
- BIM como criterio cualitativo.
- BIM introducido como criterio de solvencia técnica (servicios y obras ejecutadas, personal técnico, recursos materiales y técnicos).

Para el caso de la Comunidad Autónoma de Canarias, y dado que está previsto que en un futuro sea obligatorio el uso de esta metodología en todas las licitaciones públicas de obra civil, en las actuales licitaciones se propone valorar como criterio cualitativo la aplicación de la metodología BIM, lo que supone una segunda fase de implantación de su uso en obra civil de carreteras en Canarias, promoviendo así la progresiva introducción de las empresas del sector y de la propia administración pública en esta metodología, con el objetivo de aprovechar la ventajas que ofrece y estar preparados cuando su uso sea preceptivo.

La siguiente Figura muestra el Timeline de la historia de BIM, desde el momento de su ideación y concepción hasta la actualidad donde la obligatoriedad por parte del gobierno marcará el paradigma de la situación BIM en España.

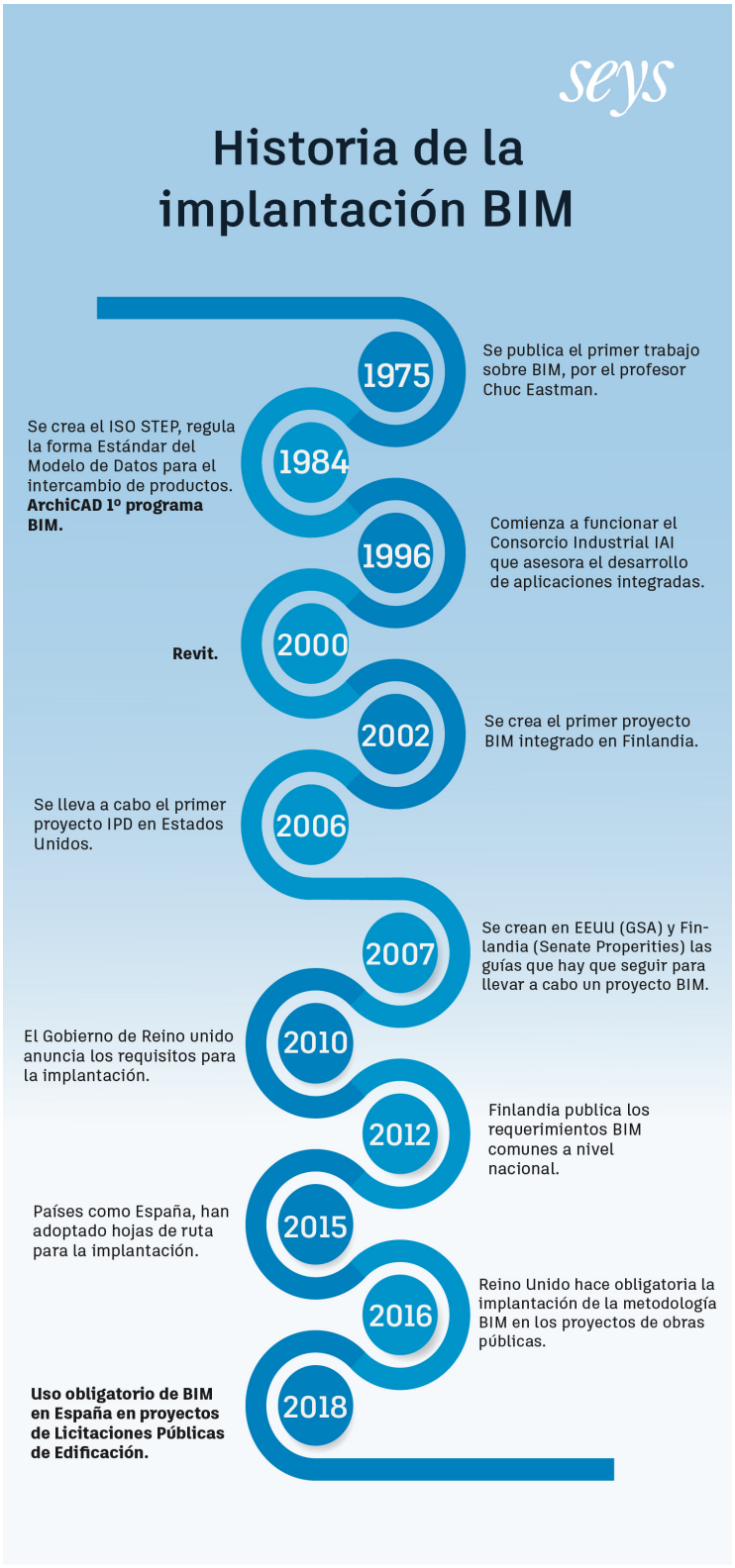


Figura 5. Timeline de la historia de BIM. Seystic, 2020. Recuperado de https://seystic.com/bim-la-historia-del-buildinginformation-modelling/

2.2. BIM en el cálculo de las estructuras

BIM es una metodología que se lleva a la práctica mediante un determinado software y, de un tiempo a esta parte, está revolucionando el mundo de los sistemas de ingeniería civil, especialmente el de los puentes.

El puente colgante es el que permite la máxima ligereza y el mínimo peso muerto cuando se trata de grandes luces. En este sentido, el Golden Gate Bridge, con sus 1.280 metros de luz entre ejes de pilas, es el mayor vano salvado a día de la fecha.

Aún sin necesidad de llegar a estos extremos, con este ejemplo pretende ponerse de manifiesto la complejidad y seriedad que representan el diseño y la construcción de un puente, tan estructuralmente dominante que no se presta a ser tratado como ningún otro sistema estructural.

En base a lo anterior, el concepto Bridge Information Modeling (BRIM), que podría definirse como una metodología de trabajo colaborativo que proviene de BIM, pero aplicada exclusivamente a los puentes [9], pone a disposición de los ingenieros de estructuras una importante plataforma metodológica que permite la integración en el procedimiento de optimización y cálculo estructural de la tecnología BIM, automatizando todo el proceso conjunto, y controlando de forma más efectiva otras variables de interés tales como el costo total de la construcción de la estructura, sus plazos de ejecución o, incluso, la conservación y mantenimiento de la misma.

En este sentido, en el ámbito de las inspecciones de las obras de paso se define la conservación como el *conjunto de operaciones y trabajos necesarios para que una estructura se mantenga con las características funcionales, resistentes e incluso estéticas con las que fue proyectada y construida* [10].

A través de la inspección de una estructura se obtienen los datos necesarios para conocer en un instante dado el estado de la misma, la cual está sometida a las solicitaciones para las cuales se ha diseñado inicialmente, a las inclemencias ambientales, en ocasiones difíciles de evaluar y considerar y a las acciones accidentales muchas veces imprevisibles. En las inspecciones se observan aspectos tales como la presencia de baches, roderas, rotura de losas de transición, estado de los sumideros de drenaje, eflorescencias, fisuras, choques de vehículos, presencia de vegetación, desplomes, erosiones en la cimentación, etc., tal y como puede apreciarse en las figuras 6 y 7 que se muestran a continuación:

Figura 6. Implementación en BIM de las patologías asociadas a una bóveda triarticulada. Guía de inspecciones básicas de obras de paso. Red de Carreteras del Estado. Ministerio de Fomento, 2009

Código..... Carretera :..... P.K. Denominación :.....

FICHA DE DAÑOS **INSPECCIÓN BÁSICA**

ELEMENTOS ESTRUCTURALES 1: VANOS (Tablero sobre pilas/estribos; arco; bóveda; Marco; Tubo)

DAÑO	NO	SI	ESTADO			OBSERVACIONES	FOTOGRAFÍAS
			A	NR	UR		
Humedades/eflorescencias							
Vegetación/acumulación de materiales							
Degradación superficial/coqueras/nidos de grava/juntas degradadas (fábricas)							
Golpes/desconchones/roturas/pérdida/desplazamiento de piezas							
Fisuras/grietas		X			X	Existen dos tipos de fisuras, unas en la embocadura [foto 52] y otras en las propias bóvedas [fotog 53]	52, 53
Armaduras vistas/corroidas/rotas							
Corrosión de elementos/soldaduras							
Pérdida de tornillos/roblones							
Deformaciones/desplazamientos excesivos/abombamientos							

Figura 7. Implementación en BIM de ficha de daños en una bóveda triarticulada. Guía de inspecciones básicas de obras de paso. Rede de Carreteras del Estado. Ministerio de Fomento, 2009

Con el objeto de asegurar el mantenimiento de la vialidad y prevenir problemas de futuro, garantizando de este modo un cierto nivel de calidad y servicio de las obras de paso, se hace necesario desarrollar una metodología que permita realizar un mantenimiento bien organizado, ágil, eficaz y adaptado a los recursos de la administración competente a tal fin.

Dado que los datos recopilados a través de las inspecciones se vuelcan habitualmente en hojas de cálculo y fichas de inspección que resultan incompatibles con la tecnología BIM, el trabajo realizado por Shuyuan Xu [11] propone un método basado en el estándar de intercambio de información (IFC) con el objeto de evitar las prácticas actuales en relación con el almacenamiento de dichos datos.

IFC es un formato de datos que permite el intercambio de información sin la pérdida de los datos que contiene. La información recopilada a través de las inspecciones puede modelarse en BIM, lo cual permitirá evaluar de forma automática la gravedad y/o el alcance de los defectos de conformidad con la normativa técnica de aplicación, así como predecir la posible evolución del deterioro de la estructura, decidir la frecuencia de las inspecciones a realizar, determinar las necesidades de mantenimiento e, incluso, vincular la respuesta estructural a los defectos del puente a través de una correcta interpretación del modelo integral de la obra de paso y de la información que proporcionan los datos sobre su estado de conservación.

La Figura 8 muestra cómo se evalúa la interoperabilidad del archivo IFC bridge BIM:

Por un lado, se realizó una prueba cíclica de exportación-importación en el mismo software, Blender, exportando en primer lugar el modelo original del puente, actualizando el archivo IFC bridge BIM con información relacionada con la inspección para, finalmente, importar dicho archivo IFC actualizado para visualizarlo.

Por otro lado, se llevó a cabo una prueba similar en todos los visores BIM y herramientas de autoría de BIM importando el IFC a otros programas como Solibri y FreeCAD. Se pudo verificar que, en cualquiera de los casos, la información del defecto (una grieta) se conservó en todas las herramientas de software analizadas.

La capacidad de mantener la integridad de la información relacionada con la inspección, en este caso particular, durante el intercambio encaja en el proceso open BIM.

De este modo, las distintas administraciones competentes en materia de explotación y mantenimiento de las infraestructuras, podrían acceder al modelo BIM del puente desde diferentes programas y plataformas sin que se pierda información o ésta sea ambigua.

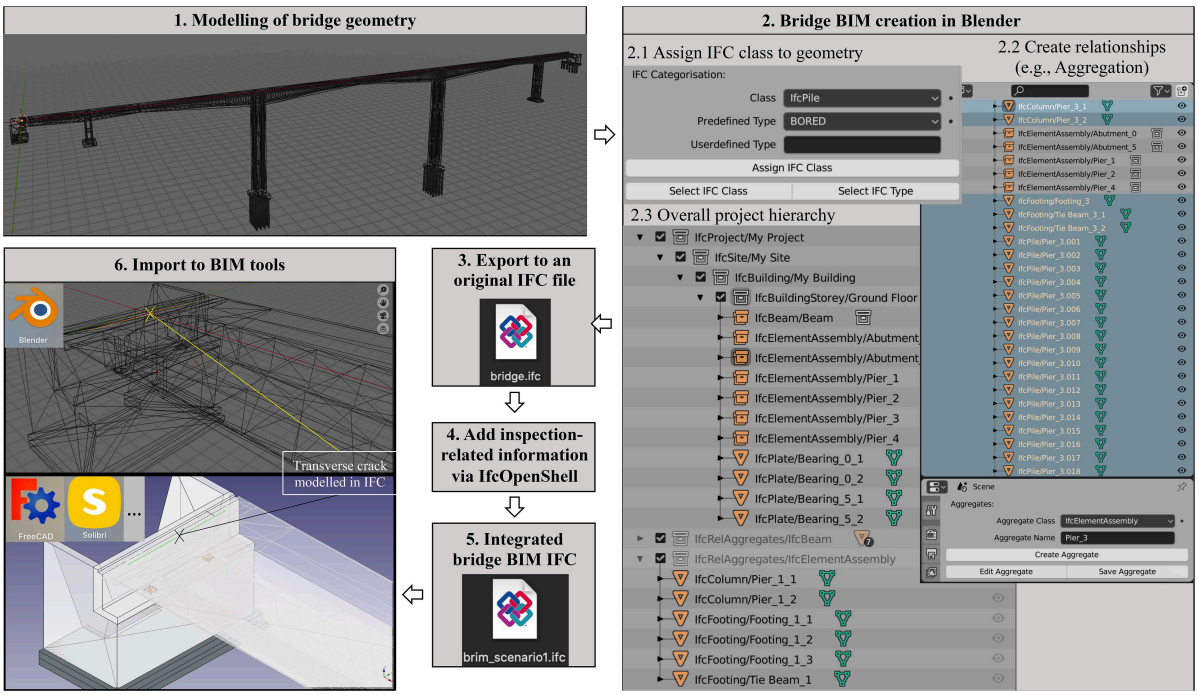


Figura 8. Diagrama de flujo para verificar la interoperabilidad. A Parameter-Driven Method for Modeling Bridge Defects through IFC. Shuyuan Xu, Jun Wang, Xiangyu Wang, Peng Wu, Wenchi Shou and Chao Liu, 2020

Por otro lado, en el diseño estructural de edificios de gran altura [12], las exigencias estéticas y las consideraciones estructurales requieren la automatización de todo el proceso conjunto, de tal forma que el modelo arquitectónico y el estructural esté basado en la colaboración estrecha entre todas las disciplinas que en ellos participan y se actualicen de forma automática ante cualquier modificación de los mismos.

En el diseño convencional de edificios de gran altura, los arquitectos y los ingenieros trabajan de forma independiente, de tal manera que el arquitecto proporciona los requisitos funcionales y estéticos, mientras que el ingeniero se encarga de los criterios de seguridad estructural y de los costes de ejecución. El resultado de este procedimiento de diseño se traduce en la pérdida de información, duplicidad de datos, inexactitud, demoras, sobrecostos excesivos y calidad insuficiente del proyecto.

En este sentido, en el trabajo realizado por Tofigh Hamidavi [12] se desarrolla una metodología de trabajo colaborativa capaz de generar de forma automática un diseño estructural optimizado basado en un modelo arquitectónico, y en el que se facilita la colaboración dinámica entre arquitectos e ingenieros durante todo el proceso de diseño, tal y como se observa en la figura 9.

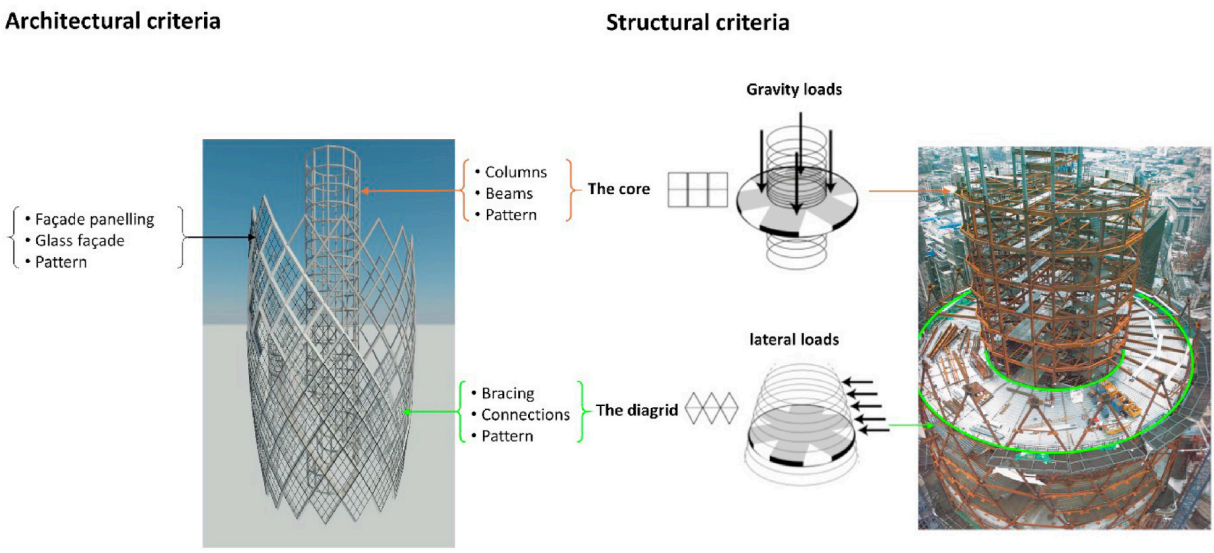


Figura 9. Diseño integrado arquitectónico-estructural. Towards intelligent structural design of buildings: A BIM-based solution. Tofigh Hamidavi, Sepehr Abrishami, M. Reza Hosseini, 2020

La tecnología BIM permite mejorar el proceso de diseño estructural utilizando las ventajas de la automatización, ya que facilita a los diseñadores la generación y optimización de diferentes modelos estructurales a través de un proceso totalmente automático y paramétrico, garantizando de este modo la sinergia entre arquitectos e ingenieros de estructuras en una plataforma basada en la tecnología BIM, mejorando la interoperabilidad de ambas disciplinas.

Un ejemplo de este procedimiento consiste en la sincronización de los datos paramétricos del modelo arquitectónico en la plataforma de diseño estructural, diseñando diferentes modelos estructurales alternativos. Dado que los modelos estructurales así generados se basan en las condiciones de contorno del modelo arquitectónico, cualquier modificación de dicho modelo actualizará los modelos estructurales de forma automática.

Otros autores han analizado los beneficios que la tecnología BIM proporciona en el diseño estructural, a través de un modelado sistemático, de una plataforma de visualización interactiva y de las interfaces de intercambio de datos estandarizados [13].

El uso de la metodología BIM en ingeniería civil y, más concretamente, en procedimientos de análisis estructural permite que un proyecto sea diseñado directamente en 3D y de forma completa en todo su conjunto. De esta forma, tanto la redacción inicial del proyecto como su posible modificación en caso de ser necesario, se podrá realizar de una manera más eficiente y rápida. A su vez, dado que el software BIM emplea elementos específicos de la ingeniería civil, tales como muros, carreteras, terraplenes, etc. los cuales llevan asociados sus características inherentes (altura, espesor, longitud, densidad, precio), cualquier operación posterior sobre las mediciones, el presupuesto o el control de plazos se ve altamente facilitada.

BIM permite la representación integral de la geometría 3D de los sistemas de ingeniería civil y de arquitectura del sector de la construcción y, más concretamente, en el ámbito de las estructuras, permitiendo de este modo un adecuado control geométrico y topográfico de dichos sistemas durante su construcción y/o explotación, la monitorización de la salud estructural en viaductos u otras obras de paso, el funcionamiento y/o explotación y fallos de sistemas estructurales tales como las obras marítimas, el monitoreo de deformaciones en túneles ferroviarios, etc.

Cualquier conflicto o incompatibilidad entre las distintas partes que constituyen el proyecto, es fácilmente detectable mediante la herramienta BIM en fase previa a la construcción.

La metodología de trabajo BIM tiene en cuenta, además, el mantenimiento y explotación de la infraestructura proyectada y construida, ya que incluye el periodo de vida útil de los diferentes materiales y equipos incorporados.

Según se recoge en el estudio de Hung-Lin Chi et al. [13], las principales ventajas de implementar BIM consisten entre otras en la reducción de los costes de diseño y órdenes de cambio, la mayor coordinación entre los diferentes profesionales y equipos implicados, el aumento de la productividad, la reducción de errores de diseño, etc. Las tendencias a futuro en relación con el desarrollo del diseño estructural integrado en BIM buscan un diseño estructural parametrizado con respecto a la funcionalidad estructural, la sostenibilidad y la seguridad. Se requiere un desarrollo innovador de herramientas eficaces de apoyo a la toma de decisiones basadas en bases de datos BIM y trabajos cooperativos fortalecidos con capacidades de intercambio de datos de BIM.

En la siguiente Figura se muestra un diagrama de flujo multifuncional para un diseño estructural basado en tecnología BIM:

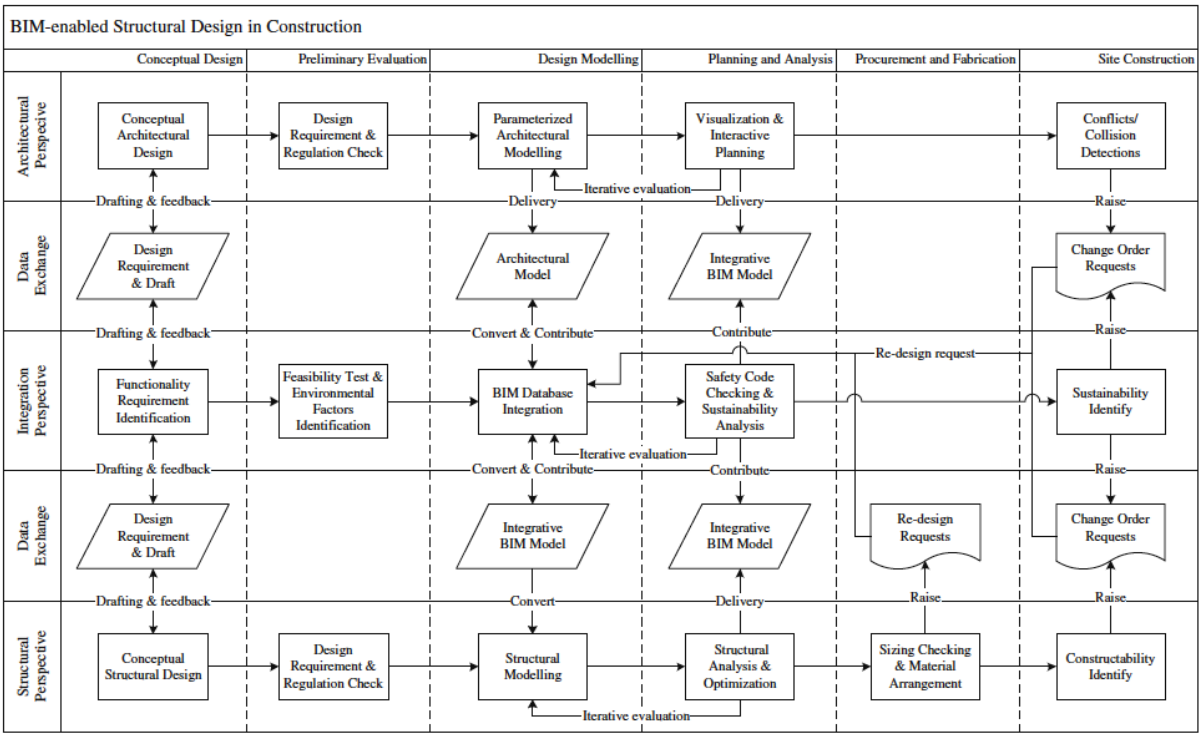


Figura 10. Diagrama de flujo multifuncional para un diseño estructural basado en tecnología BIM. BIM-Enabled Structural Design: Impacts and Future Developments in Structural Modelling, Analysis and Optimisation Processes. Hung-Lin Chi, Xiangyu Wang, Yi Jiao, 2014

Otro estudio relacionado con la aplicabilidad de la metodología BIM en el campo de la ingeniería civil analiza el impacto económico que suponen los problemas relacionados con la interoperabilidad en el ámbito de dicha metodología [14]. Para ello, se implementa un procedimiento de diseño de obras lineales basado en la asociación de modelos BIM creados con diferentes herramientas de software que se vinculan mediante formatos de archivo específicos con el objeto de resolver los problemas de interoperabilidad entre estas aplicaciones, tal y como puede observarse en la figura 11.

Concretamente, en el estudio de Aranda et al. [14] se analiza la mejora de la interoperabilidad entre el software ISTRAM, de diseño de obras lineales, y el software CivilEstudio, de cálculo estructural, mediante el establecimiento de una estructura de archivos común para el intercambio de información. BuildingSMART promueve el formato de archivo IFC con el objeto de mejorar el intercambio de información entre los programas usados en el sector de la construcción, de tal forma que el conjunto de clases así generado estandariza el desarrollo de aplicaciones de software, de forma que resulten interoperables.

Los problemas de interoperabilidad tienen un impacto económico importante en el ámbito de la ingeniería civil, debido fundamentalmente a la falta de medios que amparan la integración de los distintos programas utilizados en el diseño. El procedimiento implementado en este estudio [14] permite una optimización de los recursos tanto humanos como materiales, así como la obtención de un mejor diseño limitando las posibilidades de deficiencias y errores en el mismo.

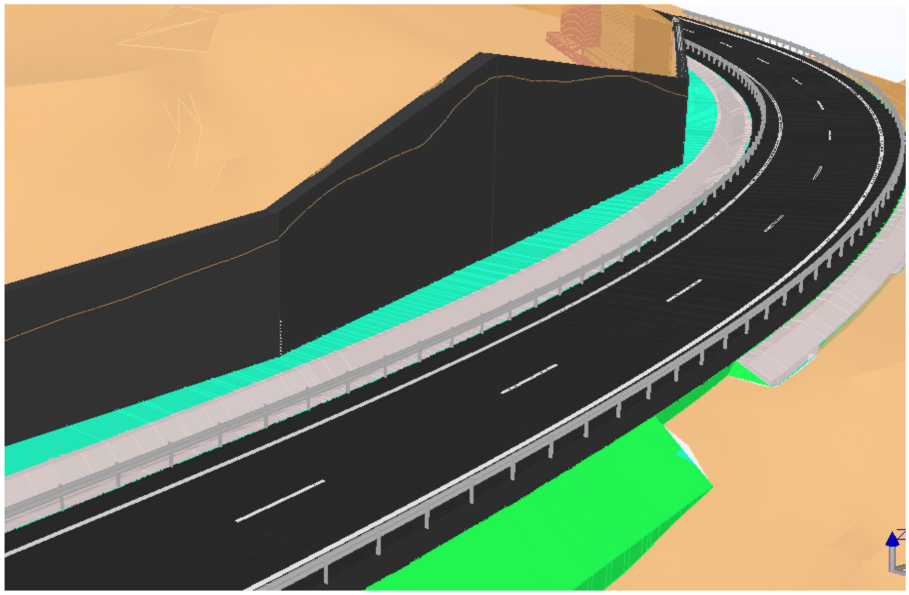


Figura 11. Asociación de la topografía, el diseño de la obra lineal (Istram) y el diseño del muro de contención (CivilEstudio) en un único modelo. Sustainability and Interoperability: An Economic Study on BIM Implementation by a Small Civil Engineering Firm. José Ángel Aranda, Norena Martín-Dorta, Ferrán Naya, Julián Conesa-Pastor and Manuel Contero, 2020

Por otro lado, las diversas tipologías de puentes dificultan el uso de las bases de datos BIM existentes, dado que existen importantes variaciones geométricas entre los diferentes tipos estructurales tales como puentes arco, atirantados, colgantes, etc. En este sentido, el estudio de Girardet [15] desarrolla un modelo utilizando un algoritmo paramétrico para modelar los elementos constitutivos de una estructura en un software de diseño y para generar y analizar el modelo en un software de análisis estructural.

Actualmente, los proyectos de ingeniería civil son cada vez de mayor envergadura, por lo que las empresas deben desarrollar nuevas herramientas para generar documentación gráfica, así como para gestionar los cambios y la actualización de información durante la fase de diseño. Los softwares tradicionales de dibujo, tales como AutoCAD, están siendo sustituidos por la tecnología BIM, la cual proporciona una gran ventaja en términos de integración de datos. Sin embargo, a pesar de que esta metodología se encuentra entre las más avanzadas

para el modelado de elementos estructurales complejos, todavía está limitada cuando se trata de manipular geometrías curvas en 3D.

En el ámbito de los tipos estructurales de ingeniería civil, y más particularmente de los puentes, uno de los desafíos que enfrentan los modelos IFC para este tipo de proyectos está relacionado con el mapeo de datos, que permite definir a nivel de un lenguaje de programación, o para niveles de desarrollo de modelos, la correspondencia entre dos modelos de datos. La generación e intercambio de modelos de sistemas estructurales tipo puentes basados en modelos IFC continúa siendo complejo, especialmente para el caso de geometrías curvas 3D.

Según se recoge en este trabajo [15], a día de la fecha todavía existe la necesidad de desarrollar un software que sea capaz de manipular los distintos elementos que constituyen un puente de geometría curva 3D, toda vez que posibilite la actualización del modelo a voluntad del usuario así como su aplicación en varias estructuras diferentes, incluyendo los parámetros asociados a los objetos.

En otro orden de cosas, los proyectos de construcción de gran envergadura generalmente involucran a varios consultores especialistas en el cálculo de estructuras que utilizan una amplia variedad de programas de análisis estructural. Estas aplicaciones y tecnologías tienen una interoperabilidad inadecuada, por lo que resulta fundamental analizar en profundidad los problemas de interoperabilidad en el ámbito de la optimización estructural. En este sentido, el estudio de Zhen-Zhong Hu et al. [16] combina un modelo de información IFC con una serie de algoritmos para mejorar la interoperabilidad entre modelos arquitectónicos y estructurales, y entre múltiples modelos de análisis estructural.

Este estudio [16] pretende superar las inconsistencias en las estructuras de datos, lógicas de representación y sintaxis utilizadas en diferentes herramientas de software. El trabajo [16] incluyó diferentes escenarios: la conversión bidireccional entre cuatro herramientas de análisis estructural; la comparación de la conversión propuesta con la conversión a través de enlaces directos entre las herramientas involucradas; la exportación directa desde una herramienta arquitectónica basada en IFC a través de la interfaz de programación de aplicaciones (en lo sucesivo Application Programming Interface, API) y, por último, la conversión y visualización de los resultados del análisis estructural.

La conversión entre los cuatro programas de análisis estructural objeto de estudio (ETABS, SAP2000, ANSYS y MIDAS) se analizó con éxito para todas las rutas de conversión posibles entre las cuatro aplicaciones, obteniendo como resultado una conversión precisa.

A su vez, Goran Sibenik en su estudio [17] remarca que la transferencia fluida de modelos 3D entre distintas herramientas de análisis y diseño sin inconsistencias significativas en los datos sigue siendo un gran desafío en la práctica. Con el objeto de reducir plazos y lograr una mayor calidad de diseño, se precisa de una alta interoperabilidad en términos de intercambio de datos, por lo que [17] propone y recomienda mejoras para limitar los problemas existentes en la transferencia de datos entre el diseño arquitectónico y el análisis estructural. Estas propuestas de mejora proporcionan una hoja de ruta para el desarrollo del intercambio de datos.

Las figuras 12 y 13 muestran las capturas de pantalla de los modelos IFC exportados, así como algunas de las inconsistencias detectadas durante el proceso de la importación:

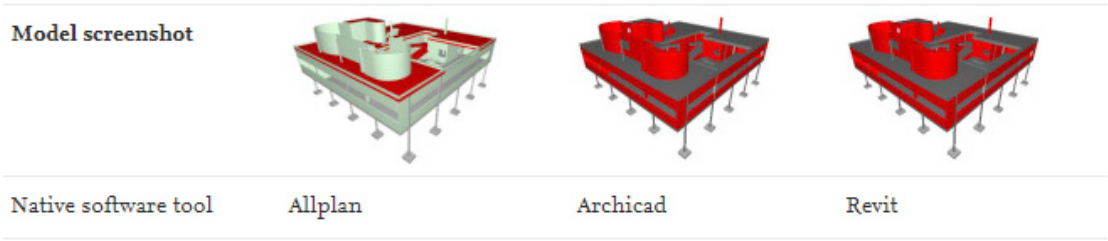


Figura 12. Captura de pantalla de los modelos exportados a IFC. Assessment of model-based data exchange between architectural design and structural analysis. Goran Sibenik, Iva Kovacic, 2020

Model screenshot			
Interpretation	Punctual interpretation	Linear interpretation	Planar interpretation (a)
Software tool combination	Revit export/RFEM import	Revit export/RFEM import	Allplan export/RFEM import
Inconsistency	Foundations interpreted as planar and linear elements instead as joints	Columns are missing	Wall interpreted as slab, offset for half of the height

Figura 13. Captura de pantalla de las inconsistencias detectadas durante el proceso de importación. Assessment of model-based data exchange between architectural design and structural analysis. Goran Sibenik, Iva Kovacic, 2020

El referido estudio [17] concluye que el intercambio de datos basado en IFC es el más difundido entre los diferentes softwares que se emplean habitualmente en la arquitectura, la ingeniería y la construcción.

A su vez, en el estudio [18] realizado por André Borrmann se analiza el proyecto IFC-Bridge de buildingSMART International, el cual ha desarrollado una extensión del estándar de intercambio de datos independiente del proveedor Industry Foundation Classes (IFC).

Dicho estudio [18] destaca la importancia de un proceso de desarrollo bien definido y la participación de un grupo de expertos internacionales, teniendo en cuenta las tipologías de puentes más extendidas y la aplicación de los escenarios de intercambio de datos que más beneficios aportan. Describe la ampliación del estándar IFC, haciendo especial énfasis en los principios generales de la norma, así como en la minimización del número de nuevas entidades. Con el objeto de identificar deficiencias, el estudio [18] se validó mediante la implementación de un prototipo en dos aplicaciones IFC, realizando varias pruebas para verificar que el intercambio de datos entre estas dos aplicaciones fuera correcto. Uno de los softwares empleados para la verificación y visualización del IFC es TUM Open Infra Platform (figura 14).

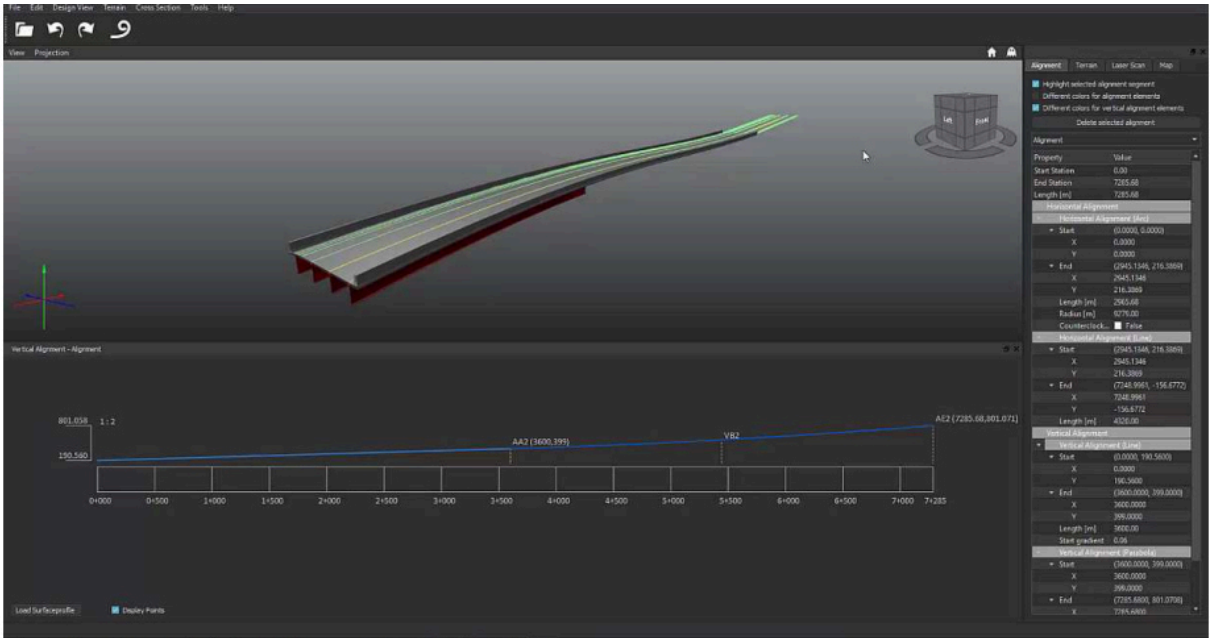


Figura 14. TUM Open Infra Platform visualiza la geometría de los tableros de los puentes creada mediante un barrido basado en la alineación. The IFC-Bridge project - Extending the IFC standard to enable high-quality exchange of bridge information models. André Borrmann, Sergej Muhič, Juha Hyvärinen, Tim Chipman, Stefan Jaud, Christophe Castaing, Claude Dumoulin, Thomas Liebich, Laura Mol, 2019

En el estudio [18] se demuestra que es posible crear una extensión bien definida del estándar IFC en un plazo limitado. Los procedimientos estandarizados de buildingSMART International ayudan a obtener un proyecto de alta calidad, garantizando la validez técnica del mismo, así como su aplicabilidad en el ámbito de destino.

Para mayor abundamiento, el autor CS Shim [19] consideraba ya desde hace una década que los modelos BIM 3D para puentes mejoraban la calidad del diseño en términos de precisión, de construcción y de colaboración, proponiendo un esquema de información extensible para puentes con el objeto de permitir la interoperabilidad entre diferentes procesos de diseño y construcción.

Desde la fase de redacción de un proyecto de construcción de puentes hasta la fase de su construcción, los modelos BIM de puentes se orientaron en la mejora del desarrollo de esta metodología, realizando maquetas digitales (en lo sucesivo Digital Mock-Up, DMU), planos de taller, modelado paramétrico, simulación 4D y 5D, con el objeto de constituir una guía de diseño de modelos BIM 3D.

Un modelo BIM de puentes bien definido puede mejorar considerablemente el proceso de revisión del diseño y la comunicación con los equipos que llevan a cabo su construcción.

La tecnología BIM ayuda a los ingenieros de estructuras a mejorar la productividad y a reducir el riesgo durante la construcción, ya que la colaboración resulta trascendental durante dicha fase y requiere un proceso de innovación.

En este estudio [19] se propone un esquema de información extensible para puentes que permita la interoperabilidad entre diferentes procesos de diseño y construcción.

Tal y como puede observarse en la Figura 15, los proyectos de construcción de puentes de gran envergadura adoptaron la tecnología BIM principalmente en la fase inicial de la construcción. Mediante la generación de modelos 3D, los ingenieros de estructuras pueden verificar su materialización física por DMU y generar planos de taller precisos.

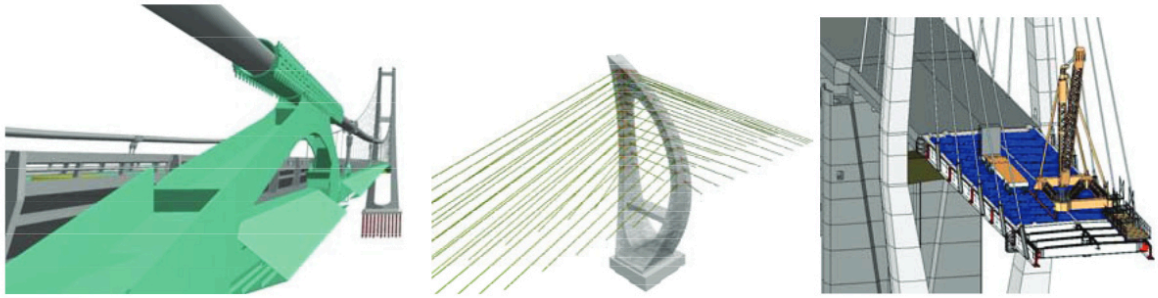


Figura 15. Aplicabilidad de tecnología BIM en puentes de gran envergadura. Application of 3D Bridge Information Modeling to Design and Construction of Bridges. CS Shim, NR Yun, HH Song, 2011

A su vez, los modelos BIM de puentes que contienen datos procedentes de inspecciones agregan un valor añadido a los sistemas de información altamente sofisticados y especializados para la gestión de estructuras existentes en la actualidad y denominados Sistemas de Gestión de Puentes (BMS, Bridge Management Systems). El estudio [20] realizado por Dušan Isailovića se orienta en la detección de daños basado en nubes de puntos, así como en la incorporación de los resultados de dicha inspección en un modelo BIM a través de un modelo IFC as-built.

Los modelos IFC se basan en la inclusión de grupos de daños clasificados y reconstruidos nuevamente en el IFC *as-built*, generando así un modelo IFC preciso que cumple con los requisitos de inspección de BMS.

IfcSurfaceFeature puede representar de manera precisa y significativa la geometría del daño. En la siguiente Figura se muestra la forma en la que se asocia un determinado daño en la estructura (IfcSurfaceFeature) al elemento dañado. En color verde se representan los objetos IFC que describen el desconchado de hormigón de la viga, la cual se representa en color gris. Los objetos IFC que definen la relación entre el desconchamiento y la viga se representan en color rojo.

IfcSurfaceFeature está conectado con el elemento dañado del puente *IfcElement* (en este caso, *IfcBeam*), a través de la relación *IfcRelVoidsElement.* Esta relación asegura un cálculo automático del resultado de la diferencia entre las representaciones geométricas de esos dos objetos cada vez que el modelo se representa en el visor IFC, restando el volumen faltante debido al daño del volumen original de los elementos del puente.

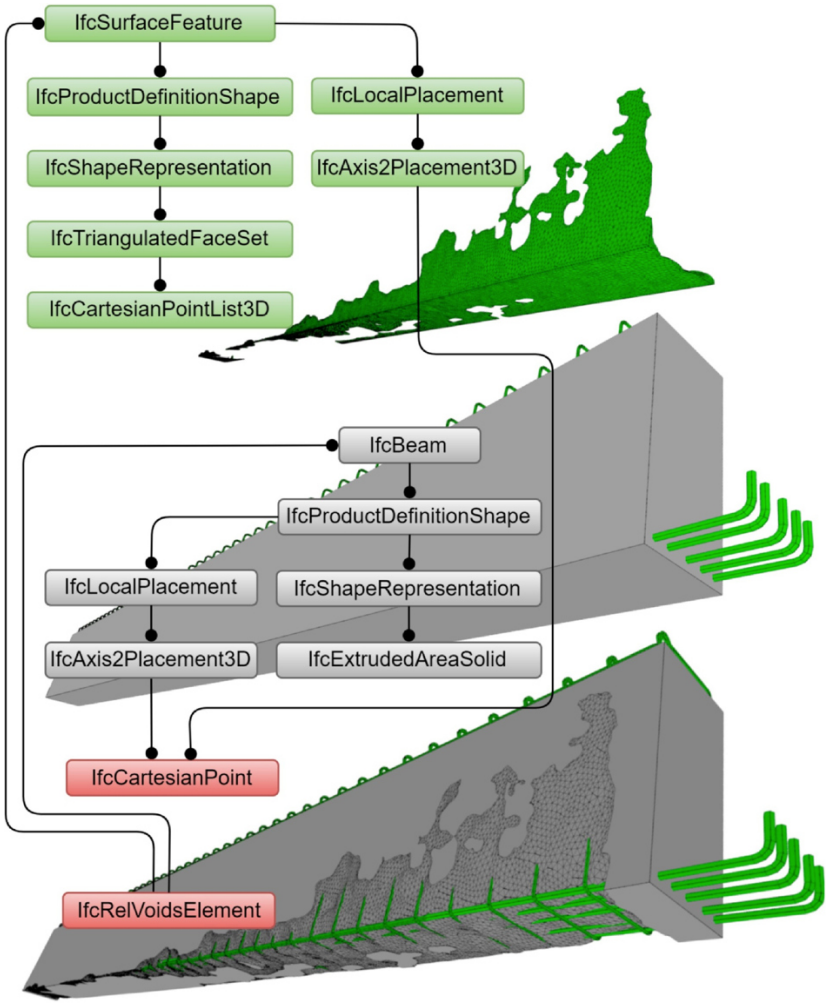


Figura 16. Estructura IFC para la representación geométrica de daños. Bridge damage: Detection, IFC-based semantic enrichment and visualization. Dušan Isailović, Vladeta Stojanovic, Matthias Trapp, Rico Richter, Rade Hajdin, Jürgen Döllner, 2020

El estudio [20] propone el puente sobre el río Gročica, ubicado en la ciudad de Belgrado, como caso objeto de análisis. A través de fotogrametría aérea se obtuvo la nube de puntos del puente. El BRIM diseñado se modeló a través del software Autodesk Revit y se exportó a formato IFC, con el objeto de evaluar a continuación los métodos para la detección de daños y el enriquecimiento del modelo BRIM. La siguiente figura muestra una representación de la nube de puntos 3D basada en fotogrametría y modelo BRIM.

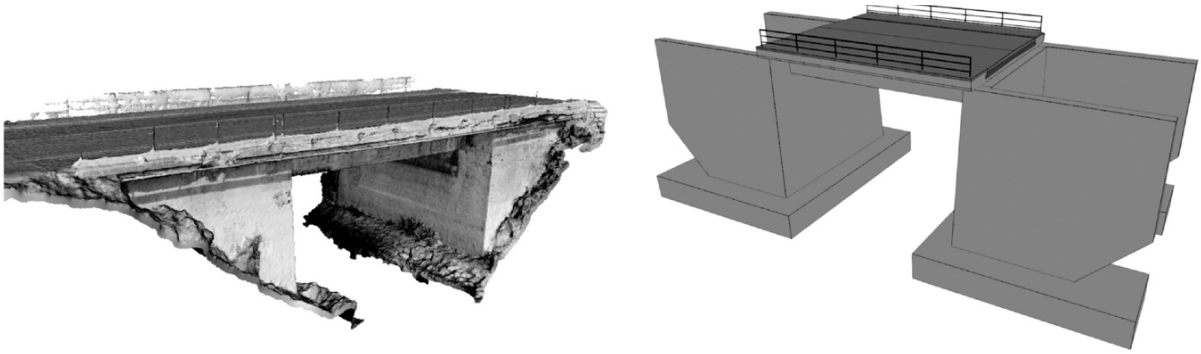


Figura 17. Representación de la nube de puntos 3D basada en fotogrametría y modelo BRIM según el diseño. Bridge damage: Detection, IFC-based semantic enrichment and visualization. Dušan Isailović, Vladeta Stojanovic, Matthias Trapp, Rico Richter, Rade Hajdin, Jürgen Döllner, 2020

2.3. Aplicabilidad del software Blender en la Ingeniería Civil

El trabajo de Puech Oriol [21] analiza las posibilidades del uso del software Blender en la ingeniería de caminos, canales y puertos como herramienta de diseño 3D a través de la programación de utilidades en Python. Para ello, se crean cinco herramientas que demuestran la viabilidad del uso del programa Blender en la ingeniería civil: (1) modelos tridimensionales del terreno, (2) perfiles longitudinales de carreteras, (3) tuberías normalizadas para distintos materiales, (4) rocas naturales y (5) bloques de hormigón artificial para diques.

Los códigos desarrollados se pueden ejecutar de forma directa en la consola de Python de Blender, quedando de manifiesto las distintas capacidades de la herramienta:

- utilizar distintas fuentes de datos,
- la capacidad de realizar modelos 3D y 2D indistintamente,
- la versatilidad en el desarrollo de los modelos,
- la capacidad de automatización de los procesos,
- la posibilidad de obtener resultados toforrealistas aplicando materiales a los objetos,
- la capacidad de generar elementos de geometría conocida mediante la parametrización de éstos y
- generar modelos de geometría imperfecta y aleatoria mediante generación procedimental.

Se muestra a continuación una figura que representa los resultados obtenidos a través del código creado breakwater blocks:

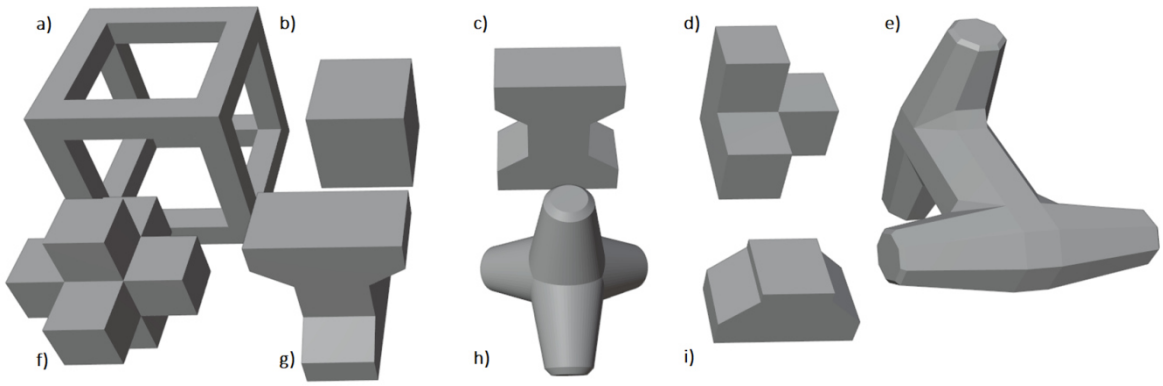


Figura 18. Bloques rompeolas generados mediante el código desarrollado: a) Cob, b) Cubo, c) Akmon tipo cruz plana, d) Tripod, e) Dolos, f) Hexaleg, g) Akmon tipo estándar, h) Tetrápodo con resolución 67 e i) mitad de Akmon. Uso de Blender a través de la programación en la ingeniería de caminos, canales y puertos. Miguel Puech Oriol, 2021

Por otro lado, Kevan Cress [22] analiza el diseño de MeasureIt-ARCH, un complemento de software con licencia GNU GPL desarrollado por el autor para agregar funcionalidad al software Blender. MeasureIt-ARCH proporciona herramientas simples para dimensionar modelos 3D, así como soporte básico para la definición y dibujo de líneas. Estas herramientas están diseñadas para usarse junto con el conjunto de herramientas de modelado y renderizado que posee Blender.

Si bien los modelos creados con MeasureIt-ARCH son fundamentalmente convencionales, al igual que la mayoría de las técnicas que emplea MeasureIt-ARCH para crearlos, este complemento proporciona dos métodos simples y relativamente novedosos en sus sistemas de dibujo. Por un lado, un nuevo método para la colocación de elementos de dimensión en el espacio 3D que se basa en el concepto tridimensional de la dimensión. Este método de colocación de dimensiones no depende de un plano de trabajo 2D, una convención que es común en el software de diseño asistido por computadora estándar de la industria. Por otro lado, también implementa un nuevo enfoque para dibujar líneas de silueta que opera transformando la geometría de los modelos con silueta en 4D 'Clip Space' (figura 19).

El enfoque modular de Blender para el almacenamiento de datos y la manipulación de la geometría contribuye a la comprensión de lo que se denomina la "mecánica de los gráficos por ordenador". Esto puede verificarse en el sistema de modificadores de Blender, su enfoque de datos vinculados, y el sistema de controladores, que permiten apilar las operaciones de manera flexible para producir resultados complejos a partir de una geometría.

El modelado en Blender no trata de ocultar la forma en que el ordenador maneja los datos en la infografía 3D. El enfoque modular de las herramientas de modelado contribuye a desarrollar el "pensamiento de diseño" y a utilizar Blender como una herramienta de toma de decisiones para el diseño de conceptos y no sólo como una herramienta de visualización.

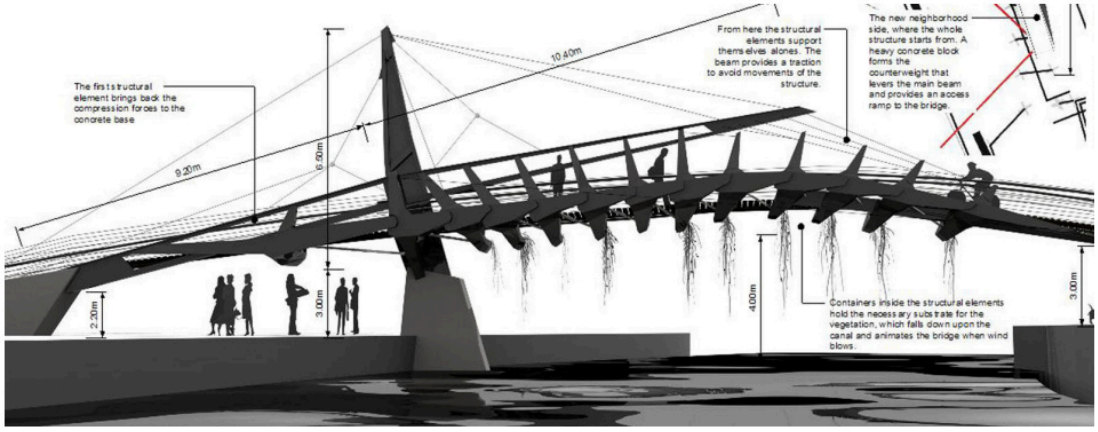


Figura 19. Yorik Van Havre – Pennington, Road Bridge Case Study at Leeds-Liverpool. A parametric bridge derived from a Beizer curve, as seen in, ´Blender, an Open Source Design Tool´, by Alexandros Siglas and Theodoreos Dounas

3. CASO DE ESTUDIO: GENERACIÓN AUTOMATIZADA DE ARCOS

3.1. Definición

El arco es una estructura comprimida que es capaz de cubrir grandes luces, y puede considerarse como uno de los elementos estructurales básicos en todo tipo de obras de ingeniería civil y de arquitectura.

Para el técnico, el arco es o pretende ser antifunicular de las cargas; una pieza curva que, resistiendo sólo o principalmente a compresión, transmite los pesos propios y los que sobre él insisten a dos apoyos distanciados entre sí [23].

El arco ha sido el tipo estructural que más ha perdurado en el diseño y construcción de puentes. Los romanos difundieron esta forma en todos los territorios conquistados hace más de dos mil años. Aún perduran muchos arcos de piedra construidos para acueductos y puentes en Europa y Asia menor.

Como las fuerzas internas principales son de compresión, en la antigüedad fue ampliamente usado con materiales como la piedra y la mampostería de ladrillo de arcilla, de buena resistencia a la compresión, y en formas circulares que producían básicamente compresión.

En el dimensionamiento del arco resulta fundamental definir la directriz del mismo de modo que se ajuste lo más posible al funicular de sus cargas, garantizando que todos sus elementos estén comprimidos y que no se producen esfuerzos de flexión. Si la ley de reparto de cargas es fija, la directriz puede amoldarse al funicular de esas cargas, pero si la sobrecarga es variable y carga más intensamente en una zona del arco, esa zona tiende a hundirse, levantándose el arco por otro lado y apareciendo flexiones de distintos signos.

El funicular de cargas es la curva que describe un cable suspendido por sus extremos, sometido a cargas en su longitud. Para el caso de un arco de espesor constante sometido exclusivamente a su peso propio el funicular es la catenaria. Para el caso de carga uniformemente repartida a lo largo de la cuerda, la directriz teórica es la parábola de segundo grado.

En la práctica de puentes, con el peso del tablero, tímpanos más o menos aligerados y arcos de espesor variable, van bien las parábolas de cuarto o mejor de sexto grado. Cuando la sobrecarga móvil es fuerte respecto al peso propio, la forma de la directriz

pierde importancia; y lo que se puede hacer es tantear los funiculares y leyes de tensiones máximas para elegir una directriz apropiada [23].

Mucho se ha escrito sobre la elección de la directriz conveniente del arco; y, efectivamente, interesa su afinamiento cuando se trata de grandes luces y fuertes cargas muertas. Aparte del rebajamiento que suele venir impuesto por otras condiciones, no de tipo resistente, la forma de la directriz no viene influida sensiblemente por los esfuerzos térmicos o de retracción, sino más bien por las cargas muertas y por el tipo de sobrecarga viva que haya de soportar el arco [23].

Los arcos generan esfuerzos horizontales que deben ser absorbidos por los apoyos. Para un valor constante de la carga “p” y para una misma luz de vano “L”, al variar la flecha del arco “h” la componente vertical RAv se mantiene constante (RAv = p * L/2). Por otro lado, la componente horizontal RAh aumenta a medida que disminuye la altura “h”, por lo que el valor del empuje horizontal es inversamente proporcional a su altura. Es decir, para reducir el valor del empuje horizontal en los apoyos, el arco debería ser lo más alto posible, tal y como puede observarse en la figura siguiente:

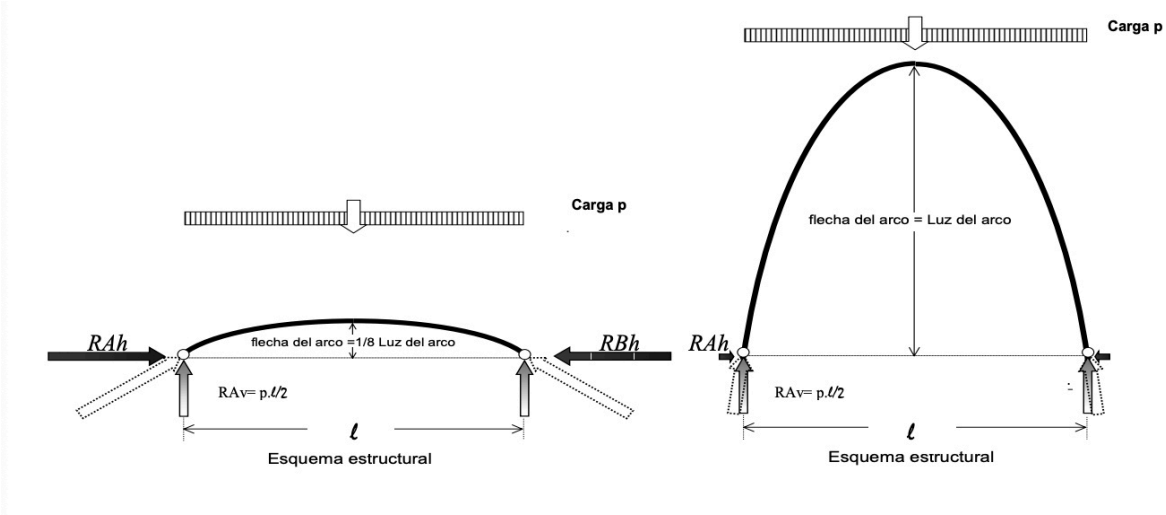


Figura 20. Recuperado de https://es.slideshare.net/woodcarmelle/8-sistemas-estructurales-de-forma-activa-el-arco

Frecuentemente, el rebajamiento del arco es libre dentro de ciertos límites; cuanto mayor es el rebajamiento, mayor es la compresión en el arco y el empuje sobre el estribo. Por el contrario, al disminuir el rebajamiento sin reducir la luz del arco, aumenta la flexión transversal por empujes laterales de viento y, en cambio, disminuyen los esfuerzos de tracción, etc. Sino hay razones funcionales, constructivas o de otro tipo, para la elección del rebajamiento, el problema llega a ser embarazoso. Hay que resolverlo por tanteos; pero, normalmente, se eligen rebajamientos entre 1/5 y 1/7 [23].

La flecha del arco “h” no puede disminuirse excesivamente, no solo por el aumento de los empujes horizontales sino también por el aumento de la flexión, e incluso del pandeo ayudado por el acortamiento de la directriz, fenómeno que puede agravarse en los puentes de hormigón, por efecto de la deformación lenta.

En arcos de puentes con dimensiones extraordinarias todos estos aspectos cobran especial importancia, debido a que las razones económicas, de disminución de material y de peso, pasan a primer plano. En los arcos pequeños con tímpanos macizos, la rigidez permite una gran libertad en su trazado.

Para el caso objeto de estudio se considera un arco de hormigón armado de 30 metros de luz y 6 metros de flecha.

Empleando el arco como tipo estructural esencial, en este trabajo se describe el proceso de generación automática que permite la definición de geometrías complejas de los sistemas estructurales más representativos, tales como el puente arco.

El puente arco es el tipo estructural idóneo para rangos de luces de unos 200 metros, incluso de los puentes arco de hormigón. Los puentes arco metálicos podrían llegar a luces de 500 metros. La ubicación del tablero bien por encima del arco (figura 21), o bien por debajo de él (figura 22), dependerá de la forma de la cerrada, es decir, de la profundidad del barranco o cauce a salvar, y de la geología de sus márgenes. De esta forma, en un barranco poco profundo, sólo podremos recurrir a un tablero superior a costa de rebajar considerablemente el arco, lo que exige que el material en las márgenes del barranco sea muy competente, debido a la elevada compresión que el arco transmite a sus estribos por razón de su rebajamiento. En caso contrario, habría que recurrir a un arco con tablero inferior.

Tal y como se ha mencionado con anterioridad, en este trabajo se validará la aplicabilidad del uso del software Blender en el modelado geométrico de un puente arco a partir del arco generado previamente.

Figura 21. Puente de Los Tilos ubicado en la carretera C-830 de Santa Cruz de La Palma a Puntagorda por el Norte. Tramo: Tenagüa - Los Sauces. Recuperado de https://www.ferrovial.com/es-es/negocio/proyectos/arco-de-los-tilos/

Figura 22. Viaducto de Erques ubicado en el suroeste de la isla de Tenerife, en la carretera de Adeje a Santiago del Teide. Recuperado de https://newsroom.ferrovial.com/es/videos/viaducto-de-erques/

3.2. Diseño e Implementación en Blender

3.2.1 Introducción

Blender es un software de diseño 3D libre y de código abierto (Open Source), desde el 13 de octubre de 2002. El término *software libre* utilizado por la Free Software Foundation (fundadores del Proyecto GNU y creadores de la Licencia Pública General GNU) hace referencia a que cualquiera puede utilizar el programa para cualquier propósito, así como copiarlo y distribuirlo, modificarlo y tener acceso al código fuente, y mejorarlo liberando versiones propias.

Blender incluye una amplia gama de herramientas esenciales tales como el modelado, renderizado, animación y rigging, edición de video, VFX, composición, texturizado y algunos tipos de simulaciones. Posee una interfaz gráfica OpenGL que es personalizable con scripts de Python y la capacidad de utilizar y desarrollar extensiones (add-ons).

En torno a Blender se agrupan millones de usuarios con una clara mentalidad colaborativa y de aprendizaje compartido, constituyendo una comunidad sólida y activa que constituye una de sus características más importantes.

El modelado se realiza a través de una serie de objetos que se componen de dos partes claramente diferenciadas:

- El objeto en sí: contiene la información sobre la posición, la rotación y el tamaño de un elemento concreto.
- Datos del objeto (*ObData*): contiene todo lo demás. Para el caso de las mallas almacena la geometría, la lista de materiales, los grupos de vértices, etc. Para el caso de las cámaras almacena la distancia focal, la profundidad de campo, el tamaño del sensor, etc. Cada objeto tiene un enlace a sus datos-objeto asociados, y un mismo dato-objeto puede ser compartido por muchos objetos.

Los principales objetos utilizados en el modelado son los siguientes:

- Mallas: objetos compuestos por vértices, aristas y caras poligonales, que pueden ser editados con las herramientas de edición de mallas de Blender.
- Curvas: objetos definidos matemáticamente que pueden manipularse con tiradores o puntos de control (en lugar de vértices), para editar su longitud y curvatura.
- Superficies: objetos definidos matemáticamente que se manipulan con puntos de control. Son útiles para formas simples redondeadas y paisajes orgánicos.

- Metaball: objetos formados por una función matemática (sin vértices ni puntos de control) que define el volumen 3D en el que existe el objeto. Cuando se juntan dos o más metaballs, se fusionan redondeando suavemente la conexión, apareciendo como un objeto unificado.
- Texto.
- Volúmenes: contenedor para archivos OpenVDB generados por otro software o por el Simulador de Fluidos de Blender.
- Elementos vacíos: objetos nulos que son simples nodos de transformación visual que no se renderizan. Son útiles para controlar la posición o el movimiento de otros objetos.
- Enrejados.
- Imágenes: objetos vacíos que muestran imágenes en el Viewport 3D. Estas imágenes pueden utilizarse para ayudar a los artistas a modelar o animar.
- Amatures: se utiliza para el rigging de modelos 3D para hacerlos animados.
- Luces: objetos vacíos que emiten luz y se utilizan para iluminar la escena en los renders.
- Cámaras: es la cámara virtual que se utiliza para determinar lo que aparece en el render.
- Campos de fuerza: objetos vacíos que dan a las simulaciones fuerzas externas, creando movimiento, y son representados en el Viewport 3D como pequeños objetos de control.

Blender permite la programación a través de Python y la creación y uso de extensiones (add-ons). Además, incluye una gran variedad de herramientas de animación, los archivos que genera son exportables en diferentes formatos, posee varios motores gráficos para el renderizado, el entorno de trabajo es personalizable según necesidades del usuario, etc.

3.2.2 Entorno de Trabajo

La instalación del software se realiza desde la página principal de Blender https://www.blender.org/

La interfaz de Blender está dividida en tres partes principales: la barra superior (en la zona superior, de color azul y naranja), la barra de estado (en la zona inferior, de color blanco) y las áreas (en la zona central, de color amarillo), tal y como se aprecia en la figura siguiente:

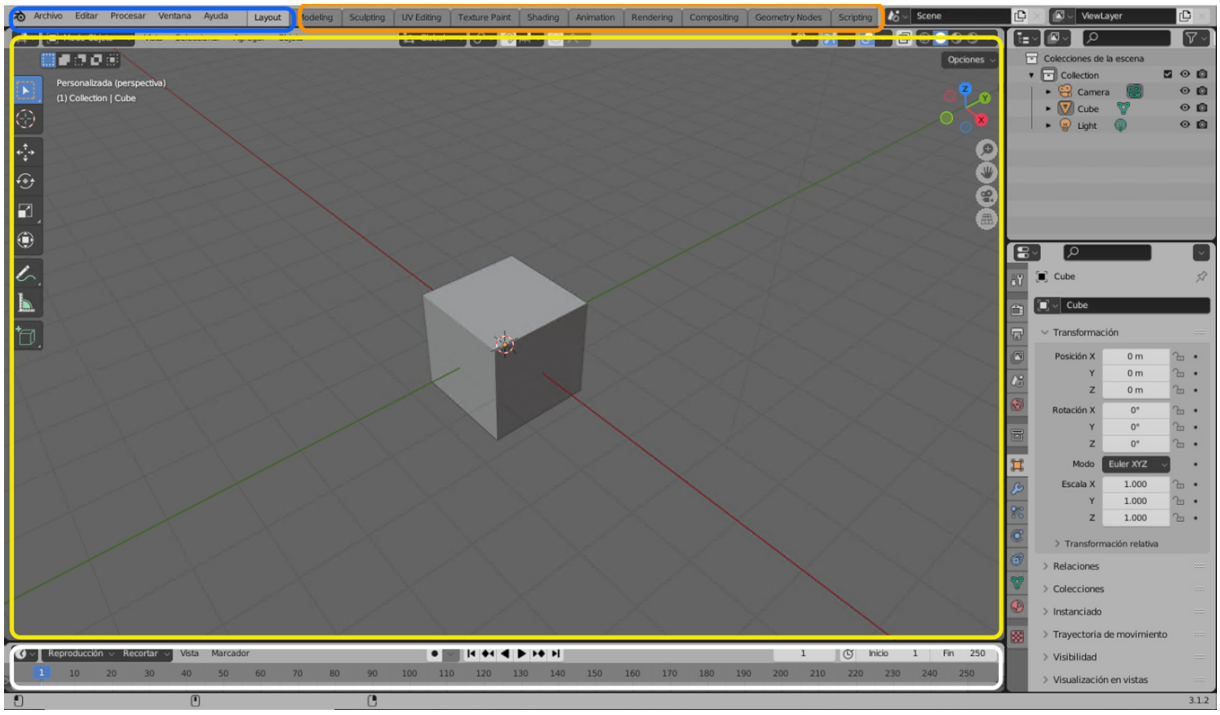


Figura 23. Diseño de interfaz de Blender. Barra superior, barra de estado y áreas. Captura de pantalla. Fuente: elaboración propia

La **barra superior** está constituida por cinco menús: archivo, editar, procesar, ventana y ayuda (en color azul) y por los espacios de trabajo, que son diseños de ventanas predefinidos (en color naranja).

Cada espacio de trabajo está dedicado a una tarea específica, como por ejemplo modelado, animación, renderizado, etc. Cuando se inicia Blender, el espacio de trabajo por defecto es Layout y contiene los editores de vista 3D, listado, propiedades y líneas de tiempo.

Los espacios de trabajo disponibles en la versión 3.1 de Blender son los siguientes:

- Modelar: Para la modificación de geometría con herramientas de modelado.
- Esculpir: Para la modificación de mallas con herramientas de escultura.
- Edición de UV: Para el mapeo de coordenadas de textura de imagen a superficies 3D.
- Pintar Texturas: Para colorear imágenes de texturas en el 3D Viewport.
- Sombreado: Para especificar propiedades de materiales para su renderizado.
- Animación: Para hacer que las propiedades de los objetos dependan del tiempo.
- Renderizado: Para visualizar y analizar los resultados de renderización.
- Composición: Para combinar y posprocesar imágenes e información de renderizado.
- Nodos de Geometría: Para modelado procedural usando Nodos de Geometría.

- **Secuencias de Comandos (para la programación): Para interactuar con la API Python de Blender y escribir scripts**.

Las ventanas de edición existentes en la versión de Blender utilizada en el desarrollo del trabajo son las siguientes:

De tipo general

- 3D Viewport: ventana utilizada para la edición del modelo.
- Image Editor: utilizada para la edición de elementos tipo imagen.
- UV Editor: se utiliza para la edición de *uv maps*.
- Compositor: para la composición de imágenes o películas mediante un sistema de nodos y uniones.
- Texture Node Editor: para la generación de texturas mediante un sistema de nodos y uniones.
- Geometry Node Editor: para la edición de geometrías mediante nodos y uniones en el modificador (modifier) Geometry Node.
- Shader Editor: utilizada para la edición de materiales que dan color a los elementos.
- Video Sequencer: utilizada para la edición de vídeos.
- Movie Clip Editor.

Para la animación

- Dope Sheet.
- Timeline.
- Graph Editor.
- Drivers.
- Nonlinear Animation.

Para la programación

- Text Editor.
- Python Console.
- Info.

Para el manejo de datos

- Outliner: ventana que permite visualizar los elementos del modelo agrupados en conjuntos llamados escenas y colecciones, en un sistema de carpetas en cascada

arborescente. Permite activar o desactivar la visualización de los elementos del modelo, el cambio de nombre su selección o eliminación entre otros.

- Properties: utilizada para la edición de variables de la escena o el objeto activo. Posee múltiples categorías, que son determinados iconos que incluyen infinidad de parámetros y herramientas.
- File Browser: buscador de archivo similar al explorador de Windows.
- Asset Browser: interfaz principal para organizar y usar recursos.
- Spreadsheet: se utiliza para inspeccionar atributos de geometría.
- Preferences: permite cambiar las preferencias de Blender, esta misma ventana de edición es accesible a través del menú principal del programa.

La **barra de estado** en la zona inferior muestra información contextual como atajos de teclado, mensajes e información estadística.

La ventana de Blender está dividida en una serie de rectángulos llamados **Áreas** (figura 24). Las áreas reservan espacio en la pantalla para Editores, como Vista 3D o Listado. Cada editor ofrece una función específica. Las áreas se agrupan en Espacios de Trabajo, que están orientados a tareas particulares (modelado, animación, etc.).

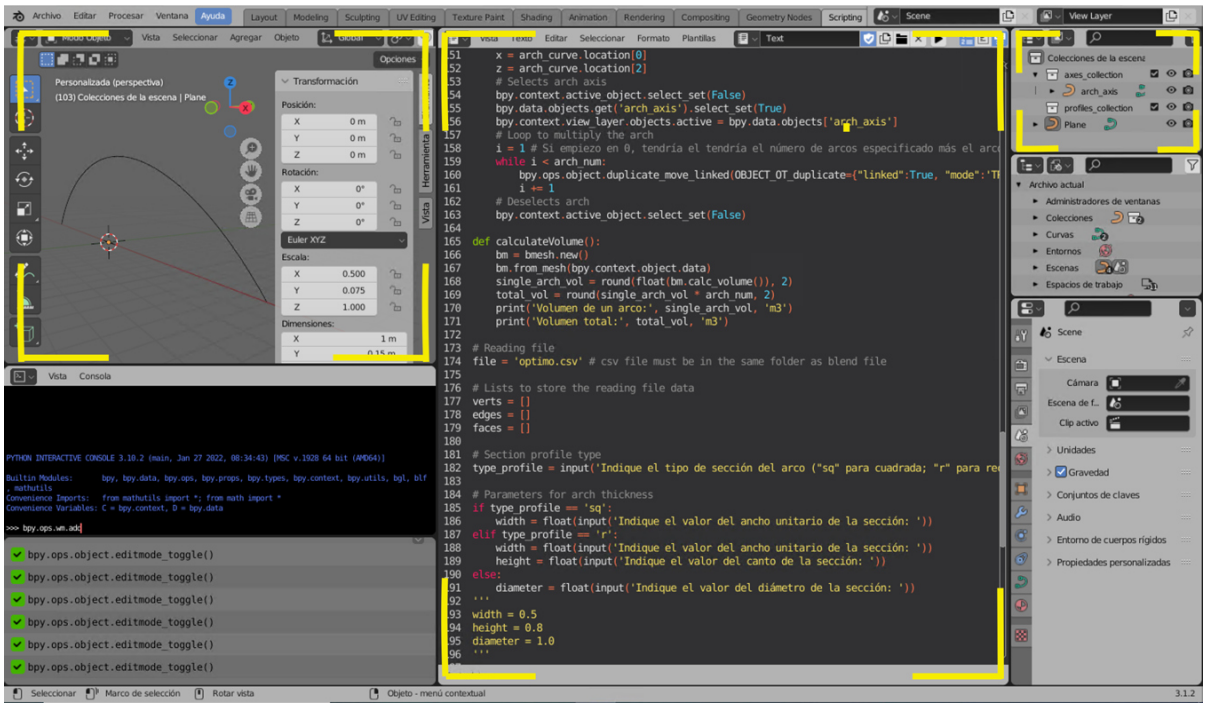


Figura 24. Diseño de interfaz de Blender. Áreas (con bordes resaltados en color amarillo). Captura de pantalla. Fuente: elaboración propia

Todo Editor en Blender está dividido en **Regiones**. Éstas pueden tener elementos estructurados más pequeños como pestañas y paneles con botones, controles y complementos ubicados en su interior. Las pestañas se emplean para controlar secciones solapadas en la interfaz del usuario, mientras que los paneles constituyen la unidad organizativa más pequeña en la interfaz.

3.2.3 Python

Python es un lenguaje de programación creado por Guido van Rossum a principios de los años 90, cuyo nombre está inspirado en el grupo de cómicos ingleses “Monty Python”. Se trata de un lenguaje interpretado o de script, es decir, en lugar de compilar el código a lenguaje máquina que pueda comprender y ejecutar directamente un ordenador (lenguaje compilado), se ejecuta utilizando un programa intermedio llamado intérprete.

Una de las capacidades más interesantes de Blender es la posibilidad de crear y ejecutar scripts de Python dentro de la propia herramienta con el objeto de crear nuevos modelos de geometría o modificar objetos en Blender. Las dos formas más comunes de ejecutar scripts de Python son utilizando el editor de texto incorporado o introduciendo comandos en la consola de Python. La consola de Python proporciona un método de acceso rápido a la API, es decir, al conjunto de instrucciones que proporciona Python para interactuar con Blender. Además de permitir la interacción mediante consola. Blender permite la ejecución de códigos mediante el editor de textos incorporado a través del lenguaje de programación Python.

Además, es posible la creación de extensiones para integrar herramientas en los distintos entornos de trabajo que posee el programa.

La descripción de las herramientas que contiene Blender dedicadas a la programación son las siguientes:

- *Text Editor*: editor de texto para la programación y ejecución de códigos.
- *Python Console*: consola de Python que permite el acceso directo a la API de Blender.
- *Info*: historial de las acciones que se han realizado en el programa, así como de los mensajes de advertencia y error que puedan surgir.

Tal y como se ha definido con anterioridad, este trabajo tiene por objeto demostrar la aplicabilidad de Blender en los sistemas más representativos de la ingeniería civil, empleando para ello “el arco” como tipo estructural básico.

Por este motivo, se ha definido un primer código que permite la generación automática de geometrías tipo arco y el cálculo de su volumen para, posteriomente, desarrollar un segundo código que permita el procesado de geometrías más complejas en entornos de ingeniería estructural a partir del arco ya creado.

3.3. Desarrollo de Scripts de Python en Blender para la defición geométrica de un arco

A continuación, se hace un desglose de las líneas de código o funciones utilizadas en el trabajo y relacionadas con la generación del arco objeto de estudio, el cual se ha dividido en tres apartados: III.3.1.- Importación de librerías, III.3.2.- Declaración de funciones y III.3.3.- Ejecución del código.

III.3.1.- Librerías y add-ons requeridos

La importación de la librería “bpy” resulta trascendental para ejecutar el programa, ya que permite controlar el funcionamiento de Blender desde la consola de Python. En este sentido, cabe indicar que la mayor complejidad del código que se ha desarrollado en Blender no es el uso de Python en sí, sino el uso de la librería “bpy”, ya que tiene una sintaxis particular. La librería “csv” permite leer las coordenadas del arco desde el archivo .CSV en el que se encuentran. La utilidad de la librería “bmesh” radica en que posibilita el cálculo del volumen de los arcos o geometrías que se generan a través del código. Por último, la librería “addon_utils” permite la activación de forma automática de un add-on o extensión de Python, lo cual es necesario para el cálculo del volumen.*import bpy*

Import csv

Import bmesh

Import addon_utils

III.3.2.- Declaración de funciones

```
def enable3dprintAddon():
    addon_name = 'object_print3d_utils'
    isenabled = addon_utils.check(addon_name)[0]
    if not isenabled:
        addon_utils.enable('object_print3d_utils')
```

Las funciones en Python se declaran por medio de la palabra reservada “def”, y todo lo que pertenezca a la función va indentado hacia dentro.

A través de esta función el programa verifica si el add-on “3D-Print Toolbox” se encuentra habilitado en el entorno de trabajo de Blender. En caso contrario, lo habilita.

Este add-on trae consigo una función que permite “hacer desplegables” las mallas de los objetos, es decir, convertir las mallas en tipo “manifold”, con el objeto de que el cálculo del volumen sea correcto.

```
def extractVertsFromFile():
    # Reads the file and extracts the vertices
    with open(file, 'r', encoding='utf-8-sig') as csvfile:
        reader = csv.reader(csvfile, delimiter=',')
        for row in reader:
            vert = (float(row[0]), float(row[2]), float(row[1]))
            verts.append(vert)
    # Joins the vertices
    for i, vert in enumerate(verts):
        startpoint = I
        if i < len(verts)-1:
            endpoint = i+1
        edge = startpoint, endpoint
        edges.append(edge)
```

Esta función “extractVertsFromFile” importa el archivo .CSV de datos de entrada que contiene el listado de coordenadas que representan todos los puntos de la directriz del arco, extrae los valores que precisa y los guarda en una lista llamada “verts”. Asimismo, la función une los vértices entre sí. Para ello, extrae mediante un bucle *for* los índices de los vértices almacenados en la lista anterior y los almacena a su vez en otra lista llamada “edges” de la siguiente manera: [(0, 1), (1, 2), (2, 3), …].

La función “with open” es la que abre y lee (se ha establecido el modo lectura con la letra “r”) el archivo .CSV.

En determinadas líneas de código se incluye una almohadilla (#) con el objeto de hacer anotaciones dentro del código. Cuando se ejecuta el código, el programa las ignora y continúa su ejecución como si no estuviesen.

```
def createSceneCollections():
    global ax_coll
    ax_coll = bpy.data.collections.new('axes_collection')
    bpy.context.scene.collection.children.link(ax_coll)
    global prof_coll
    prof_coll = bpy.data.collections.new('profiles_collection')
    bpy.context.scene.collection.children.link(prof_coll)
```

La función “createSceneCollections” permite organizar la información que se genera dentro del archivo de dibujo, guardando los objetos creados en dos colecciones, una que permite guardar los arcos creados y otra que permite guardar la forma de la sección transversal del arco, la cual se extruirá a lo largo del eje del mismo.

```
def createArchAxis():
    global arch_curve
    #Creates the arch axis mesh
    new_mesh = bpy.data.meshes.new(name='axis_mesh')
    new_mesh.from_pydata(verts, edges, faces)
    # Creates the object mesh
    new_object = bpy.data.objects.new('arch_axis_curve', new_mesh)
    # Adds the object mesh to the correspondent collection
    ax_coll.objects.link(new_object)
    # Selects the object mesh
    arch = bpy.data.objects.get('arch_axis_curve')
    arch.select_set(True)
    # Converts the object mesh to a curve
    bpy.context.view_layer.objects.active = bpy.context.scene.objects.get('arch_axis_curve')
```

```
    bpy.ops.object.convert(target='CURVE')
    # Renames the arch curve
    arch_curve = bpy.data.objects.get('arch_axis_curve')
    arch_curve.name = 'arch_axis'
```

Con esta función “createArchAxis” el software genera el eje del arco objeto de estudio a partir de los datos almacenados en las listas “verts” y “edges”, y lo guarda en la colección correspondiente. Inicialmente, genera un objeto tipo malla que luego transforma en curva, con el objeto de poder extrusionar posteriormente la sección transversal alrededor del eje.

```
def createSquareExtrusionProfile(width):
    # Creates the section profile
   bpy.ops.mesh.primitive_plane_add(enter_editmode=False, align='WORLD', location=(0,   0, 0), scale=(1, 1, 1))
    # Sets the thickness of the arch
    bpy.context.object.dimensions = (width, width, 0)
    # Converts the object to a curve
    bpy.ops.object.convert(target='CURVE')
    # Adds the section profile to the correspondent collection
    square_profile = bpy.context.scene.objects.get('Plano')
    prof_coll.objects.link(square_profile)
    # Remove the section profile from the scene collection
    bpy.context.scene.collection.objects.unlink(square_profile)
    # Renames the section profile
    square_profile.name = 'square_profile'
    bpy.context.object.data.name = 'square_profile_curve'
    # Selects arch axis
    bpy.context.active_object.select_set(False)
    bpy.data.objects.get('arch_axis').select_set(True)
    bpy.context.view_layer.objects.active = bpy.data.objects['arch_axis']
```

```
    # Assigns section profile to arch axis
    bpy.context.object.data.bevel_mode = 'OBJECT'
    bpy.context.object.data.use_fill_caps = True # To close the mesh
    bpy.context.object.data.bevel_object = square_profile
    # Transforms extruded profile along curve to mesh
    bpy.ops.object.convert(target='MESH')
    # Hides section profile
    bpy.context.view_layer.objects.active = bpy.data.objects['square_profile']
    bpy.context.object.hide_set(True)
```

Esta función “createSquareExtrusionProfile” crea una sección transversal cuadrada, en el caso de que el usuario indique al programa que desea esa opción. Pide como argumento el ancho (“width”). Una vez que la sección transversal se ha extrusionado alrededor de la curva que conforma el eje del arco, vuelve a transformar el conjunto en una malla, ocultando el cuadrado creado.

```
def createRectangularExtrusionProfile(width, height):
    # Creates the section profile
   bpy.ops.mesh.primitive_plane_add(enter_editmode=False, align='WORLD', location=(0, 0, 0), scale=(1, 1, 1))
    # Sets the thickness of the arch
    bpy.context.object.dimensions = (width, height, 0)
    # Converts the object to a curve
    bpy.ops.object.convert(target='CURVE')
    # Adds the section profile to the created collection
    rectangular_profile = bpy.context.scene.objects.get('Plano')
    prof_coll.objects.link(rectangular_profile)
    # Remove the section profile from the scene collection
    bpy.context.scene.collection.objects.unlink(rectangular_profile)
    # Renames the section profile
    rectangular_profile.name = 'rectangular_profile'
```

```
    bpy.context.object.data.name = 'rectangular_profile_curve'
    # Selects arch axis
    bpy.context.active_object.select_set(False)
    bpy.data.objects.get('arch_axis').select_set(True)
    bpy.context.view_layer.objects.active = bpy.data.objects['arch_axis']
    # Assigns section profile to arch axis
    bpy.context.object.data.bevel_mode = 'OBJECT'
    bpy.context.object.data.use_fill_caps = True # To close the mesh
    bpy.context.object.data.bevel_object = rectangular_profile
    # Transforms extruded profile along curve to mesh
    bpy.ops.object.convert(target='MESH')
    # Hides section profile
    bpy.context.view_layer.objects.active = bpy.data.objects['rectangular_profile']
    bpy.context.object.hide_set(True)
```

Esta función “createRectangularExtrusionProfile” crea una sección transversal rectangular, en el caso de que el usuario indique al programa que desea esa opción. Pide como argumentos el ancho (“width”) y el canto (“height”). Una vez que la sección transversal se ha extrusionado alrededor de la curva que conforma el eje del arco, vuelve a transformar el conjunto en una malla, ocultando el rectángulo creado.

```
def createCircularExtrusionProfile(diameter):
    # Creates the section profile
    bpy.ops.mesh.primitive_circle_add(radius=1, enter_editmode=False, align='WORLD', location=(0, 0, 0), scale=(1, 1, 1))
    # Sets the thickness of the arch
    bpy.context.object.dimensions = (diameter, diameter, 0)
    # Converts the object to a curve
    bpy.ops.object.convert(target='CURVE')
    # Adds the section profile to the created collection
    circular_profile = bpy.context.scene.objects.get('Círculo')
```

```
    prof_coll.objects.link(circular_profile)
    # Remove the section profile from the scene collection
    bpy.context.scene.collection.objects.unlink(circular_profile)
    # Renames the section profile
    circular_profile.name = 'circular_profile'
    bpy.context.object.data.name = 'circular_profile_curve'
    # Selects arch axis
    bpy.context.active_object.select_set(False)
    bpy.data.objects.get('arch_axis').select_set(True)
    bpy.context.view_layer.objects.active = bpy.data.objects['arch_axis']
    # Assigns section profile to arch axis
    bpy.context.object.data.bevel_mode = 'OBJECT'
    bpy.context.object.data.use_fill_caps = True # To close the mesh
    bpy.context.object.data.bevel_object = circular_profile
    # Transforms extruded profile along curve to mesh
    bpy.ops.object.convert(target='MESH')
    # Hides section profile
    bpy.context.view_layer.objects.active = bpy.data.objects['circular_profile']
    bpy.context.object.hide_set(True)
```

Esta función “createCircularExtrusionProfile” crea una sección transversal circular, en el caso de que el usuario indique al programa que desea esa opción. Pide como argumento el diámetro de la sección (“diameter”). Una vez que la sección transversal se ha extrusionado alrededor de la curva que conforma el eje del arco, vuelve a transformar el conjunto en una malla, ocultando el círculo creado.

```
def makeManifold():
    # Selects arch axis
    bpy.context.active_object.select_set(False)
    bpy.data.objects.get('arch_axis').select_set(True)
```

```
    bpy.context.view_layer.objects.active = bpy.data.objects['arch_axis']

    # Makes mesh manifold

    bpy.ops.mesh.print3d_clean_non_manifold()
```

Esta función “def makeManifold" convierte la malla del arco generado en tipo manifold, haciendo uso de la función correspondiente del add-on 3D-Print Toolbox. Como paso previo, selecciona la malla del arco.

```
def multiplyArch(arch_num, equidist):

    # Defines variables relative to coordinates

    x = arch_curve.location[0]

    z = arch_curve.location[2]

    # Selects arch axis

    bpy.context.active_object.select_set(False)

    bpy.data.objects.get('arch_axis').select_set(True)

    bpy.context.view_layer.objects.active = bpy.data.objects['arch_axis']

    # Loop to multiply the arch

    i = 1 # Si empiezo en 0, tendría el tendría el número de arcos especificado más el arco original

    while i < arch_num:

bpy.ops.object.duplicate_move_linked(OBJECT_OT_duplicate={"linked":True, "mode":'TRANSLATION'}, TRANSFORM_OT_translate={"value":(x, equidist, z)})

    i += 1

    # Deselects arch

    bpy.context.active_object.select_set(False)
```

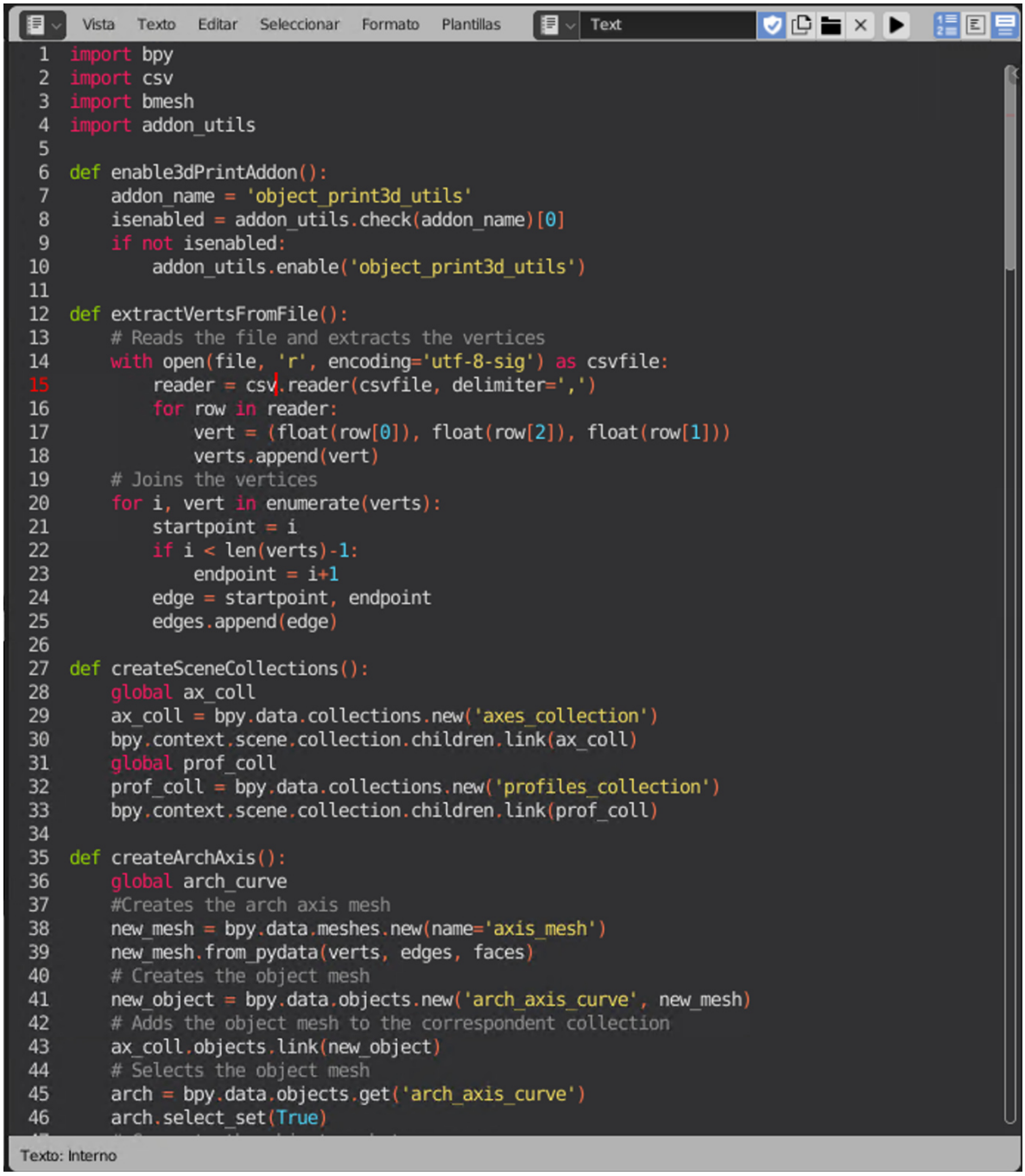
Esta función “multiplyArch(arch_num, equidist" permite generar el número de arcos que decida el usuario y los coloca desfasados sobre el eje “y”. Pide como argumentos el número de arcos y su equidistancia.

```
def calculateVolume():

    bm = bmesh.new()

    bm.from_mesh(bpy.context.object.data)
```

```
single_arch_vol = round(float(bm.calc_volume()), 2)
total_vol = round(single_arch_vol * arch_num, 2)
print('Volumen de un arco:', single_arch_vol, 'm3')
print('Volumen total:', total_vol, 'm3')
```

Esta función “calculateVolume" calcula el volumen de un arco y el volumen total de los arcos generados. A su vez, redondea el resultado a 2 decimales. Para calcular el volumen hace uso de la librería “bmesh”.

Se incluye a continuación una figura del editor de texto incorporado en el software que se ha empleado para desarrollar el primer código que permite la generación automática de geometrías tipo arco y el cálculo de su volumen.




```python
import bpy
import csv
import bmesh
import addon_utils

def enable3dPrintAddon():
    addon_name = 'object_print3d_utils'
    isenabled = addon_utils.check(addon_name)[0]
    if not isenabled:
        addon_utils.enable('object_print3d_utils')

def extractVertsFromFile():
    # Reads the file and extracts the vertices
    with open(file, 'r', encoding='utf-8-sig') as csvfile:
        reader = csv.reader(csvfile, delimiter=',')
        for row in reader:
            vert = (float(row[0]), float(row[2]), float(row[1]))
            verts.append(vert)
    # Joins the vertices
    for i, vert in enumerate(verts):
        startpoint = i
        if i < len(verts)-1:
            endpoint = i+1
        edge = startpoint, endpoint
        edges.append(edge)

def createSceneCollections():
    global ax_coll
    ax_coll = bpy.data.collections.new('axes_collection')
    bpy.context.scene.collection.children.link(ax_coll)
    global prof_coll
    prof_coll = bpy.data.collections.new('profiles_collection')
    bpy.context.scene.collection.children.link(prof_coll)

def createArchAxis():
    global arch_curve
    #Creates the arch axis mesh
    new_mesh = bpy.data.meshes.new(name='axis_mesh')
    new_mesh.from_pydata(verts, edges, faces)
    # Creates the object mesh
    new_object = bpy.data.objects.new('arch_axis_curve', new_mesh)
    # Adds the object mesh to the correspondent collection
    ax_coll.objects.link(new_object)
    # Selects the object mesh
    arch = bpy.data.objects.get('arch_axis_curve')
    arch.select_set(True)
```

Figura 25. Editor de texto incluido en Blender con una parte del código creado para generar geometrías tipo arcos. Captura de pantalla. Fuente: elaboración propia

III.3.3.- Ejecución del código

```
# Reading file

file = 'optimo.csv' # csv file must be in the same folder as blend file
```

Esta línea del código define la variable “file” como ruta relativa del archivo .CSV desde el que se va a extraer la información de las coordenadas. Es importante que esté guardado en la misma carpeta que el archivo BLEND, o el programa no lo detectará.

```
# Lists to store the reading file data

verts = []

edges = []

faces = []
```

Estas son las listas que van a almacenar la información necesaria para crear la malla que definirá el eje del arco.

```
# Section profile type

type_profile = input('Indique el tipo de sección del arco ("sq" para cuadrada; "r" para rectangular; "c" para circular): ')
```

Este es el texto que aparece por pantalla para seleccionar el tipo de sección transversal del arco.

```
# Parameters for arch thickness

if type_profile == 'sq':

    width = float(input('Indique el valor del ancho unitario de la sección: '))

elif type_profile == 'r':

    width = float(input('Indique el valor del ancho unitario de la sección: '))

    height = float(input('Indique el valor del canto de la sección: '))

else:

    diameter = float(input('Indique el valor del diámetro de la sección: '))

'''

width = 0.5
```

```
height = 0.8
diameter = 1.0
'''
```

Este texto solicita valores en función del tipo de sección transversal escogida para el arco. Al igual que el texto precedido por almohadilla, el texto entre comillas triples es texto comentado.

```
# Parameters for arch duplication
arch_num = int(input('Indique el número de arcos que desea dibujar: '))
if arch_num > 1:
    equidist = float(input('Indique el valor equidistancia: '))
else:
    equidist = 0.0
'''
arch_num = 4
equidist = 2.5
'''
```

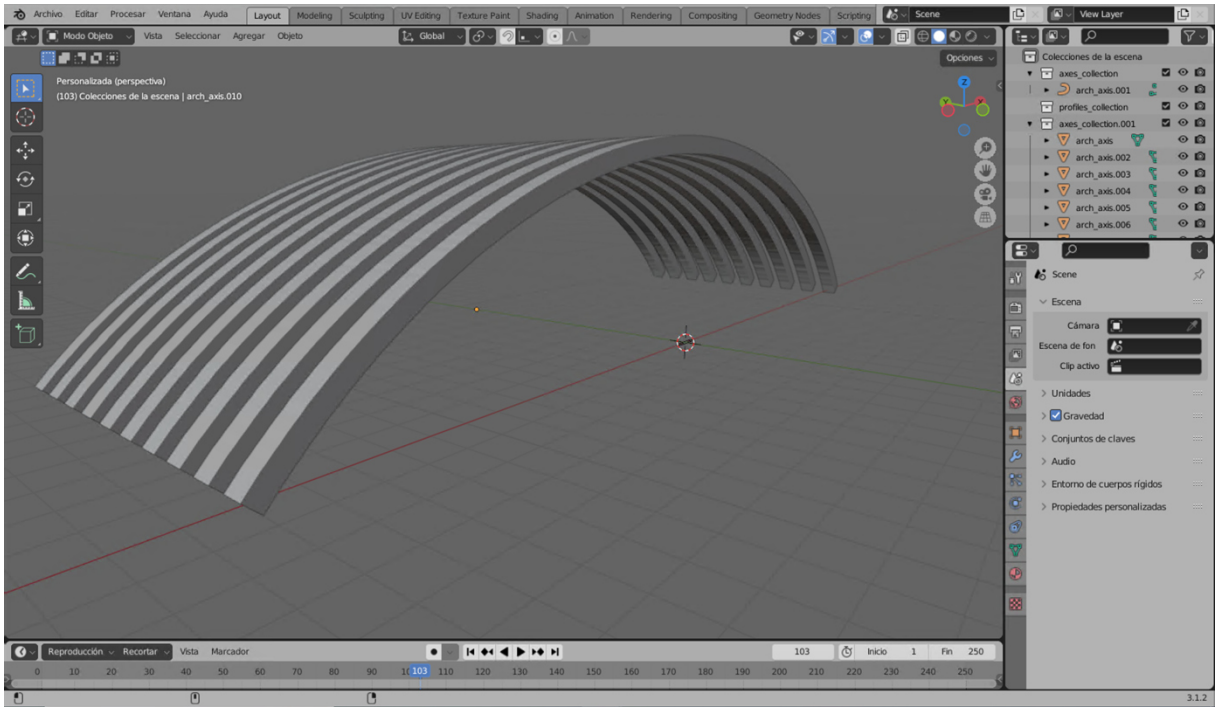
Este texto solicita al usuario que indique el número de arcos a dibujar y su equidistancia.

```
Enable3dPrintAddon()
createSceneCollections()
extractVertsFromFile()
createArchAxis()
if type_profile == 'sq':
    createSquareExtrusionProfile(width)
elif type_profile == 'r':
    createRectangularExtrusionProfile(width, height)
else:
    createCircularExtrusionProfile(diameter)
makeManifold()
```

```
multiplyArch(arch_num, equidist)

calculateVolume()
```

Finalmente, llamamos a las distintas funciones creadas para que se ejecuten.

Se incluyen a continuación una serie de figuras que resultan de la ejecución del código, para diferentes secciones transversales de arcos (cuadrada, rectangular y circular) que incluyen el cálculo del volumen correspondiente.

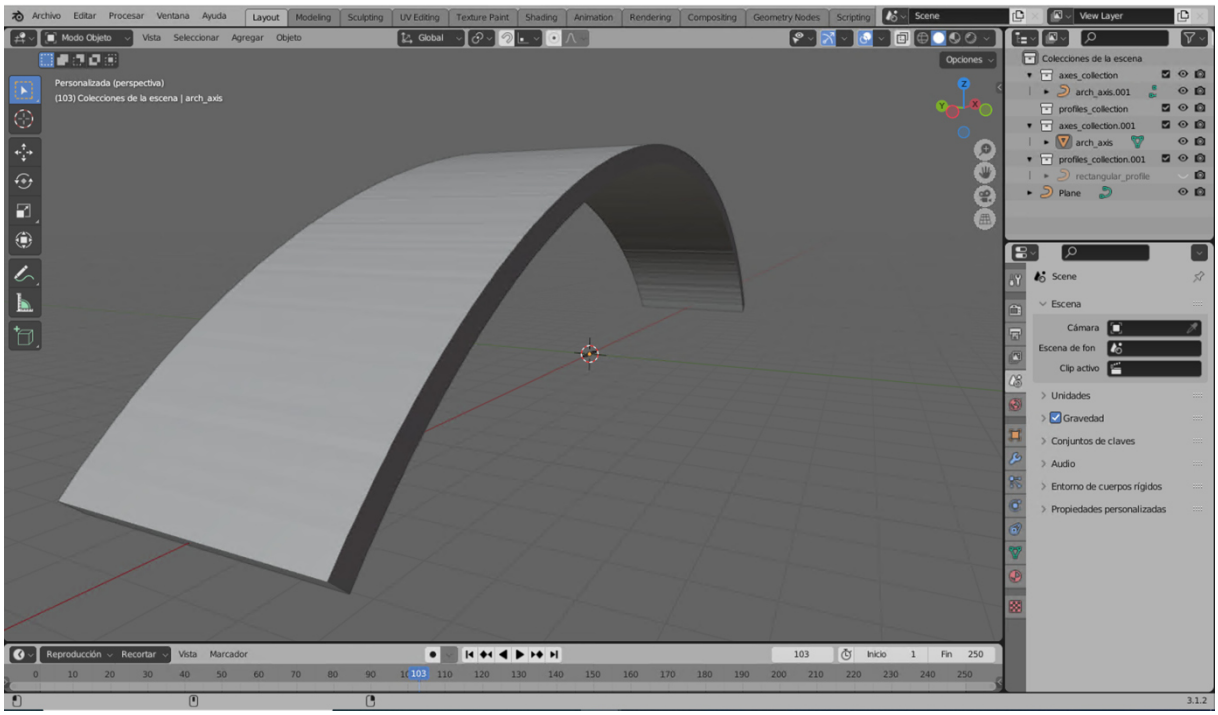


(a)

```
Read prefs: C:\Users\Usuario2\AppData\Roaming\Blender Foundation\Blender\3.1\config\userpref.blend
Read blend: C:\Users\Usuario2\Desktop\Gara_TFM\scriptVerticeCreationFromCSV_6.blend
Indique el tipo de sección del arco ("sq" para cuadrada; "r" para rectangular; "c" para circular): sq
Indique el valor del ancho unitario de la sección: 0.5
Indique el número de arcos que desea dibujar: 10
Indique el valor equidistancia: 1
Info: Removidos: 0 vértices, 0 bordes, 0 caras
Info: 8 vértices removidos
Info: Removidos: 0 vértices, 0 bordes, 0 caras
Info: Modified: -8 vertices, -8 edges, +0 faces
Volumen de un arco: 8.3 m3
Volumen total: 83.0 m3
```

(b)

Figura 26. Generación automática mediante código de diez arcos de sección transversal cuadrada, de 0,50 m de ancho, equidistantes 1,00 m entre sí y con un volumen igual a 83 m^3 (a) y consola de Python para dicha generación (b). Captura de pantalla. Fuente: elaboración propia

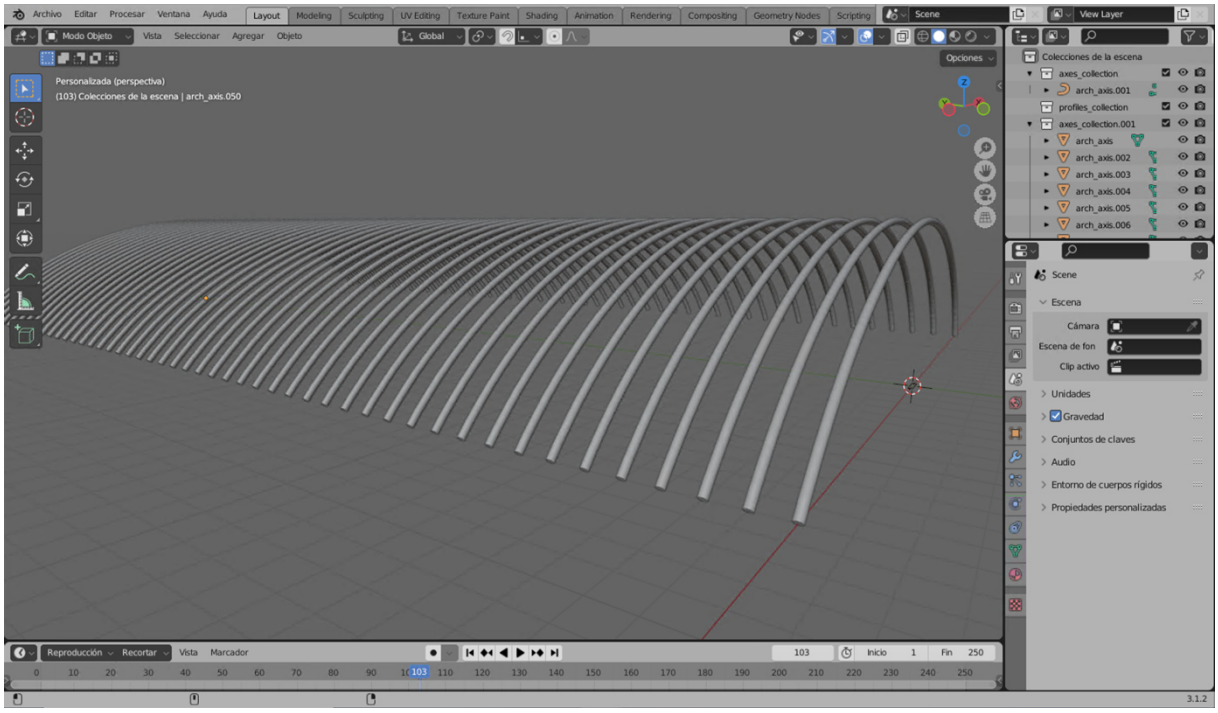


(a)

```
Read prefs: C:\Users\Usuario2\AppData\Roaming\Blender Foundation\Blender\3.1\config\userpref.blend
Read blend: C:\Users\Usuario2\Desktop\Gara_TFM\scriptVerticeCreationFromCSV_6.blend
Indique el tipo de sección del arco ("sq" para cuadrada; "r" para rectangular; "c" para circular): r
Indique el valor del ancho unitario de la sección: 5
Indique el valor del canto de la sección: 0.5
Indique el número de arcos que desea dibujar: 1
Info: Removidos: 0 vértices, 0 bordes, 0 caras
Info: 8 vértices removidos
Info: Removidos: 0 vértices, 0 bordes, 0 caras
Info: Modified: -8 vertices, -8 edges, +0 faces
Volumen de un arco: 83.02 m3
Volumen total: 83.02 m3
```

(b)

Figura 27. Generación automática mediante código de un arco de sección transversal rectangular, de 5 m de ancho y 0,50 m de canto y con un volumen igual a 83,02 m^3 (a) y consola de Python para dicha generación (b). Captura de pantalla. Fuente: elaboración propia



(a)

```
Read prefs: C:\Users\Usuario2\AppData\Roaming\Blender Foundation\Blender\3.1\config\userpref.blend
Read blend: C:\Users\Usuario2\Desktop\Gara_TFM\scriptVerticeCreationFromCSV_6.blend
Indique el tipo de sección del arco ("sq" para cuadrada; "r" para rectangular; "c" para circular): c
Indique el valor del diámetro de la sección: 0.25
Indique el número de arcos que desea dibujar: 50
Indique el valor equidistancia: 1
Info: Removidos: 0 vértices, 0 bordes, 0 caras
Info: 64 vértices removidos
Info: Removidos: 0 vértices, 0 bordes, 0 caras
Info: Modified: -64 vertices, -64 edges, +0 faces
Volumen de un arco: 1.62 m3
Volumen total: 81.0 m3
```

(b)

Figura 28. Generación automática mediante código de cincuenta arcos de sección transversal circular, de 0,25 m de diámetro, equidistantes 1,00 m entre sí y con un volumen igual a 81 m^3 (a) y consola de Python para dicha generación (b). Captura de pantalla. Fuente: elaboración propia

Estos tres ejemplos representan la efectividad que aporta un software como Blender al procesado automático de la geometría de un tipo estructural ya que, una vez definida dicha geometría, se logran extraer mediante código variables de interés de la misma.

En los ejemplos mostrados, el código programado permite ejecutar una geometría tipo arco en Blender, así como la obtención de forma ágil y totalmente automatizada de determinadas características y atributos de la misma, tales como el volumen de la estructura generada, el coste de su encofrado, etc.

En contexto BIM, el procedimiento estándar consistía en ejecutar geometrías a través de programas tales como Autocad, el modelizador gráfico de cualquier programa de cálculo de estructuras, un software BIM para, a continuación, utilizar otros programas con el objeto de obtener el resto de variables de interés (excel, Presto).

La programación de un código en Blender dota de mayor efectividad a este procedimiento al integrar en un único programa la obtención de geometrías complejas y de las propiedades o características que se precisen de forma totalmente automatizada. Este trabajo demuestra que el procedimiento clásico resulta más complejo al requerir el empleo de varios programas de forma independiente.

Para poder validar el método, se comienza modelando el tipo estructural básico, que es el arco. La utilidad del software Blender quedará demostrada a través de la obtención mediante código del volumen de hormigón y superficie de encofrado en geometrías complejas.

3.3.1 Incidencias

Durante la ejecución del código se han registrado distintas incidencias de interés y que se detallan a continuación:

1.- Primer error que arroja el código durante su ejecución (figura 29):

Error: File written by newer Blender binary (300.42), expect loss of data!

2.- Segundo error que arroja el código durante su ejecución (figura 29):

TypeError: CollectionObjects.link(): error with argument 1, "object" - Function.object does not support a ´None´ assignment Object type

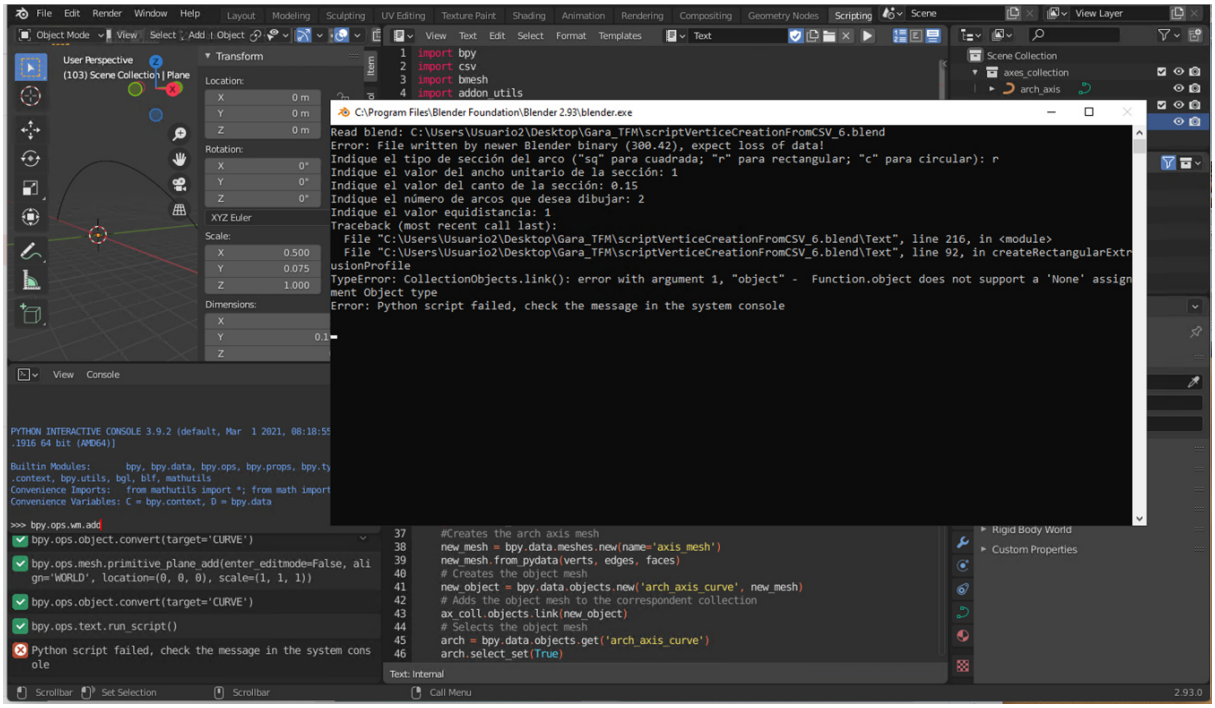


Figura 29. Captura de pantalla de las dos primeras incidencias detectadas durante el proceso de ejecución del código. Fuente: elaboración propia

El primer error está relacionado con la versión de Blender de que se trate. Es decir, si el código se ejecuta a través de una versión de Blender diferente a aquella desde la cual ha sido creado (versión 3.1), el código no se ejecuta y da error, ya que los códigos han sido desarrollados y comprobados en la versión 3.1 de Blender, sin la implementación de medidas de verificación de la compatibilidad de los códigos entre diferentes versiones del software, dado que se entiende que estas cuestiones no forman parte del objeto de este trabajo.

No obstante, para comprobar este hecho se realizaron varias pruebas de ejecución del código en dos versiones de Blender diferentes (la versión 2.93 y la versión 3.1) y mediante el empleo de diferentes ordenadores (ordenador propio y el escritorio virtual), obteniendo los resultados que se muestran en la siguiente tabla:

Tabla 2. Resumen de los resultados obtenidos tras la verificación del software

Versión	Ordenador	Error	Resultado
2.93	Propio	Error 1	A pesar de arrojar el error 1, el código se ejecuta finalmente
2.93	Escritorio virtual	Errores 1 y 2	Errores 1 y 2; Los arcos no se representan
3.1	Escritorio virtual	Sin error	Los arcos se representan correctamente

El segundo error se produce en las líneas 92 y 216 del código, y está relacionado con el primer error a través del cual el software advierte sobre la posibilidad de que pueda producirse pérdida de información al ejecutar el código a través de una versión de Blender diferente de la versión de creación.

3.- Tercer error que arroja el código durante su ejecución (figura 30):

FileNotFoundError: [Errno 2] No such file or directory: ´optimo.csv´

Este error está relacionado con las rutas relativas, las cuales no funcionan correctamente cuando el código se ejecuta desde el escritorio virtual. Es decir, cuando se utiliza el escritorio virtual, el código no detecta el archivo .CSV.

Para solventar esta incidencia, resulta necesario añadir en el código la ruta completa de la carpeta en la cual se aloja el archivo .CSV. De esta forma, la línea 174 del código quedaría como se indica a continuación:

file = ´C:/Users/Usuario2/Desktop/Gara_TFM/optimo.csv´

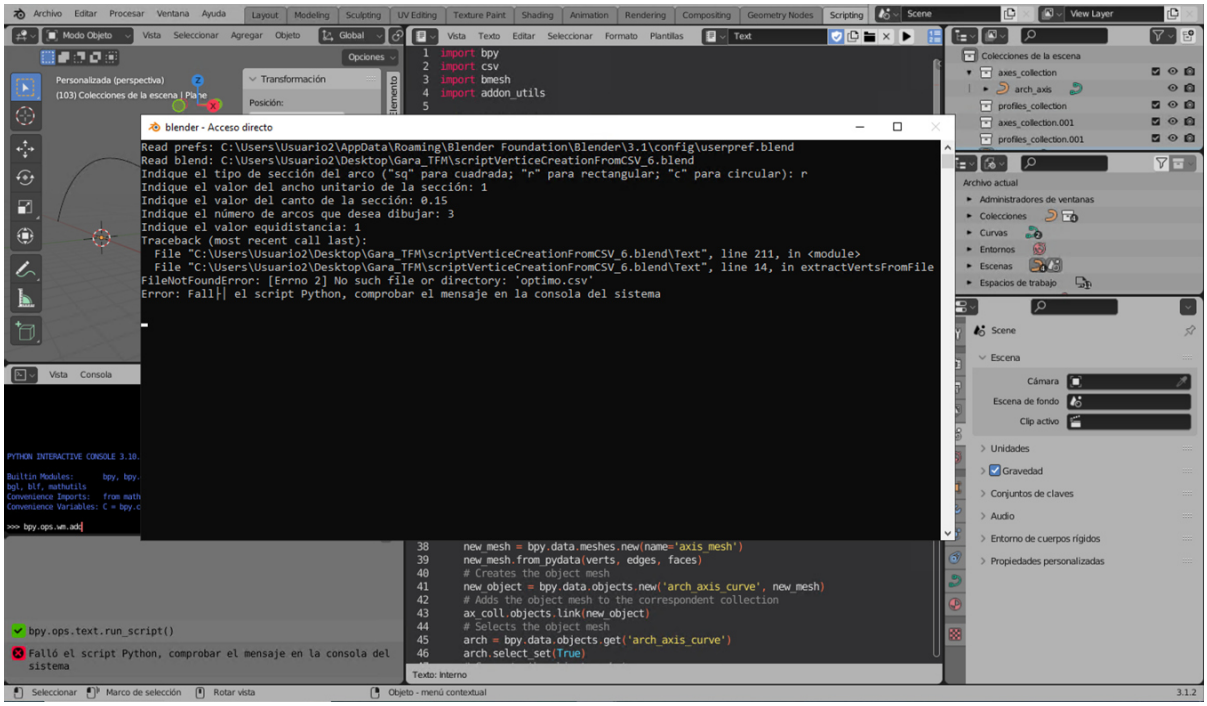


Figura 30. Captura de pantalla de la tercera incidencia detectada durante el proceso de ejecución del código. Fuente: elaboración propia

3.4. Desarrollo de un Menú en Blender para el modelado geométrico de un puente arco

3.4.1 Procedimiento

Tal y como se ha indicado en apartados anteriores, este trabajo tiene por objeto abordar la automatización del procesado de la información geométrica en el contexto de los tipos estructurales de aplicación frecuente en ingeniería civil.

En la sección 3.3 de este trabajo se ha descrito el primer código programado con Python que permite leer un conjunto de datos de entrada desde un archivo externo en formato .CSV y los transforma en una geometría tipo arco representada a través de una interfaz gráfica.

Empleando el arco como tipo estructural esencial, en este apartado se describe el proceso de generación del segundo código que permite la definición de geometrías complejas de los sistemas estructurales más representativos tales como el puente arco, y la obtención de determinadas características y atributos de los mismos, de forma totalmente automatizada.

A esa geometría generada se le asigna, además, una clase IFC a través de la librería ifcOpenShell.

Para este segundo código se ha desarrollado un add-on que amplía las funcionalidades básicas del software Blender al mostrarse en pantalla un menú totalmente personalizado a través del cual puede ejecutarse el código programado.

Se describe a continuación el proceso seguido para la programación de este segundo código:

En primer lugar, se ha procedido a la creación de los scripts que conforman el add-on para la generación automática de arcos. Para que aparezca en pantalla el menú del add-on se deben ejecutar los siguientes archivos desde el panel *Scripting* del programa: *archUtilOperators.py* y *archUtilPanel.py*, tal y como se muestra en figura 31.

El funcionamiento del add-on así desarrollado es el siguiente:

- El botón *Select file* del menú del add-on permite seleccionar el archivo de datos externos que se desea importar en Blender.
- El botón *Number of arches* permite especificar el número de arcos a dibujar y su equidistancia.

- Los botones *Square profile* y *Rectangular profile* definen la sección transversal del arco. Al pulsar sobre dichos botones, aparece a continuación un menú emergente que permite especificar las dimensiones del arco.
- Los botones *Area report* y *Volumen report* generan los informes de superficie de encofrado y volumen de hormigón necesarios para la ejecución de los arcos (figura 32).

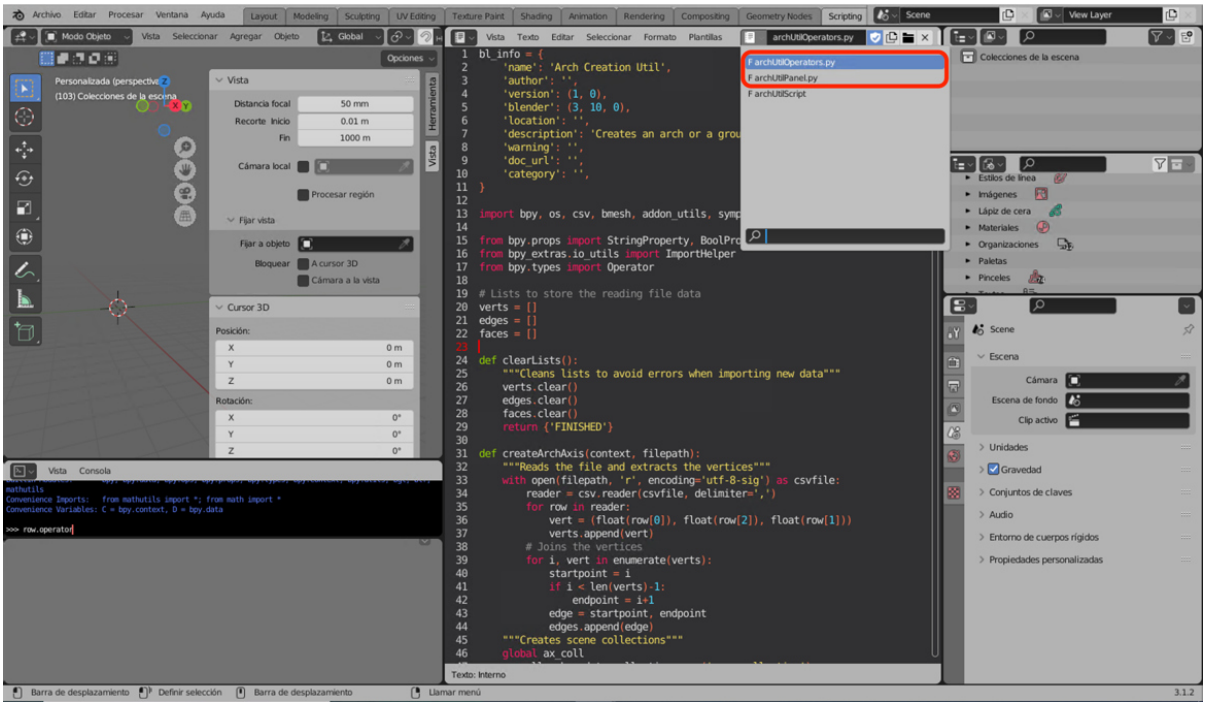
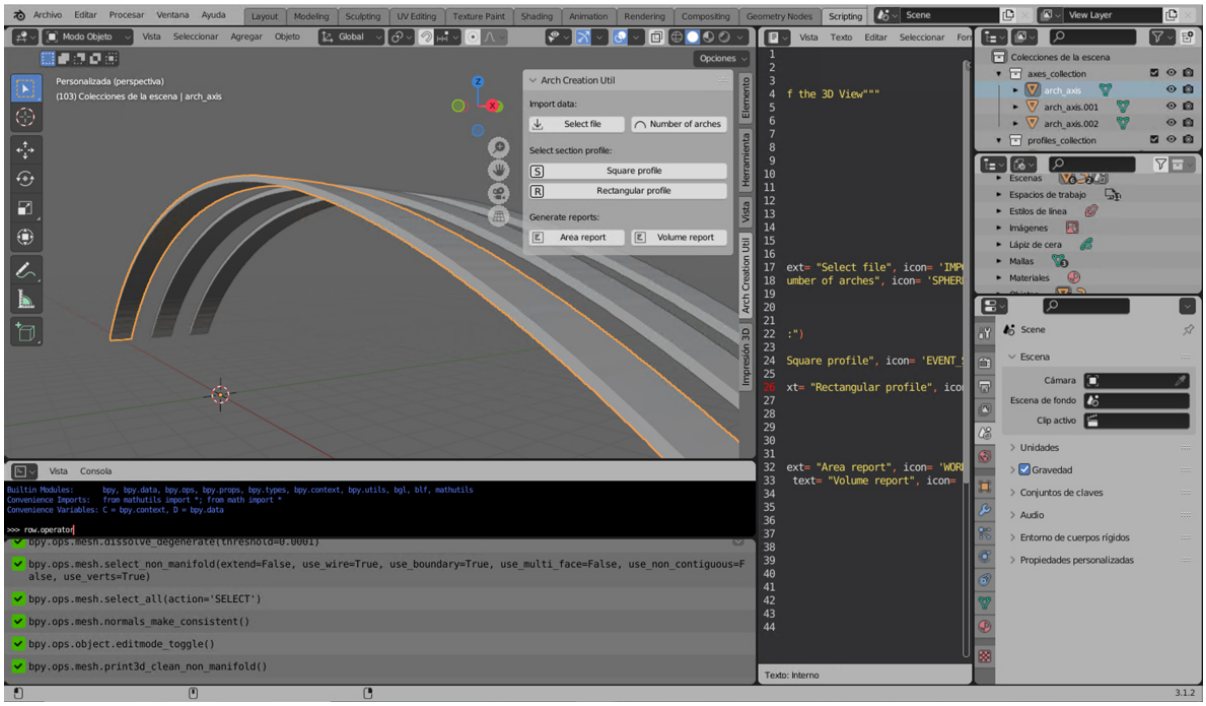
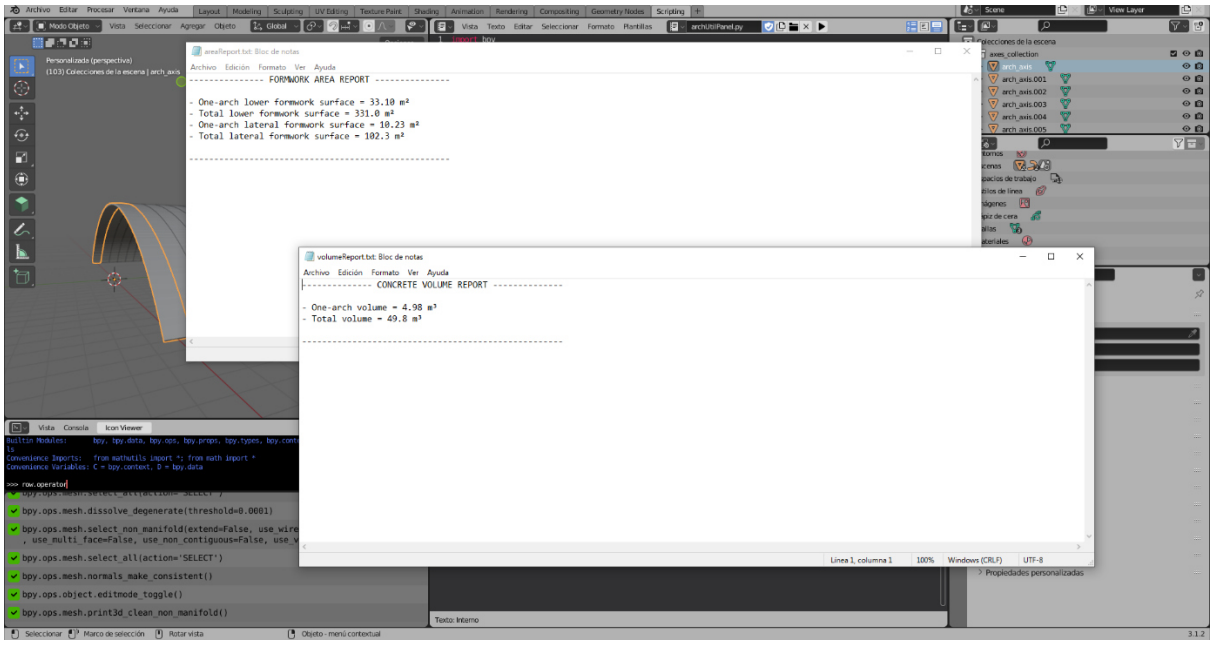


Figura 31. Captura de pantalla que muestra los archivos *archUtilOperators.py* y *archUtilPanel.py* para la generación automatizada de arcos. Fuente: elaboración propia



(a)



(b)

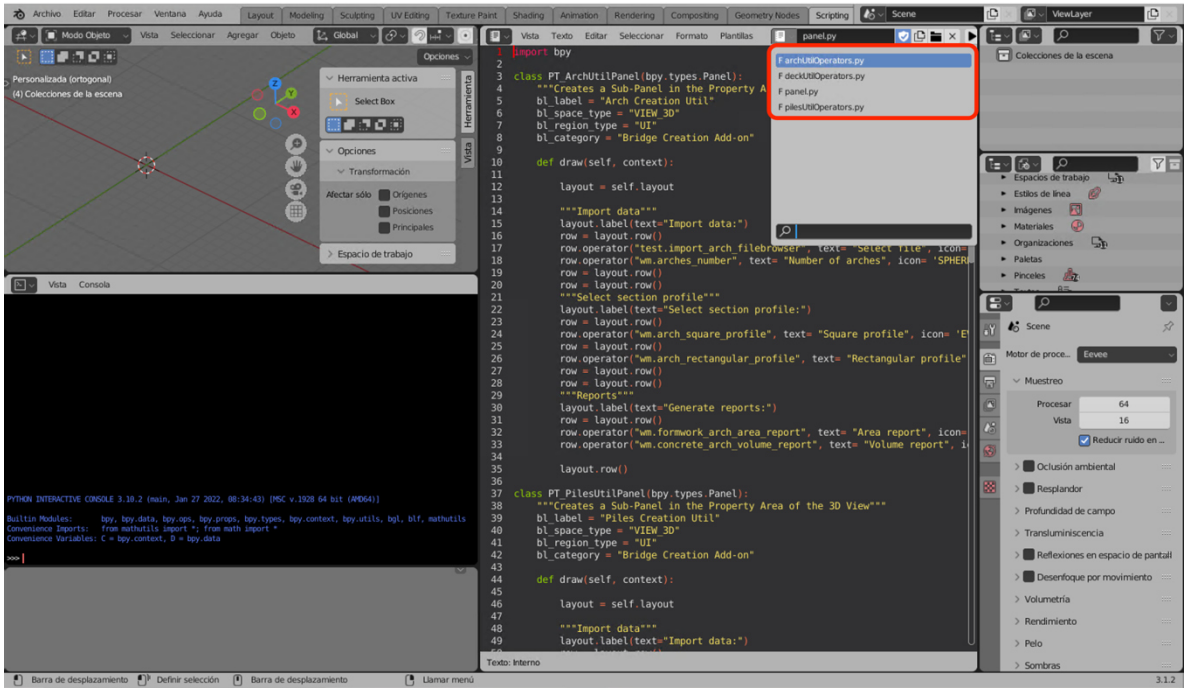
Figura 32. Captura de pantalla que muestra el menú del add-on que permite la generación automatizada de arcos (a) y los informes de superficie de encofrado y volumen de hormigón con extensión .txt. Fuente: elaboración propia

En segundo lugar, a partir del script creado para el arco se programan dos nuevos scripts de Python para el resto de elementos constitutivos del puente arco: las péndolas y el tablero para, a continuación, unir todos los scripts en un único add-on. Con el objeto de que aparezca en pantalla el menú del add-on se deben ejecutar los siguientes archivos desde el panel *Scripting* del programa: *F archUtilOperators.py*, *F pilesUtilOperators.py*, *F deckUtilOperators.py* y *F panel.py F* (figura 33).

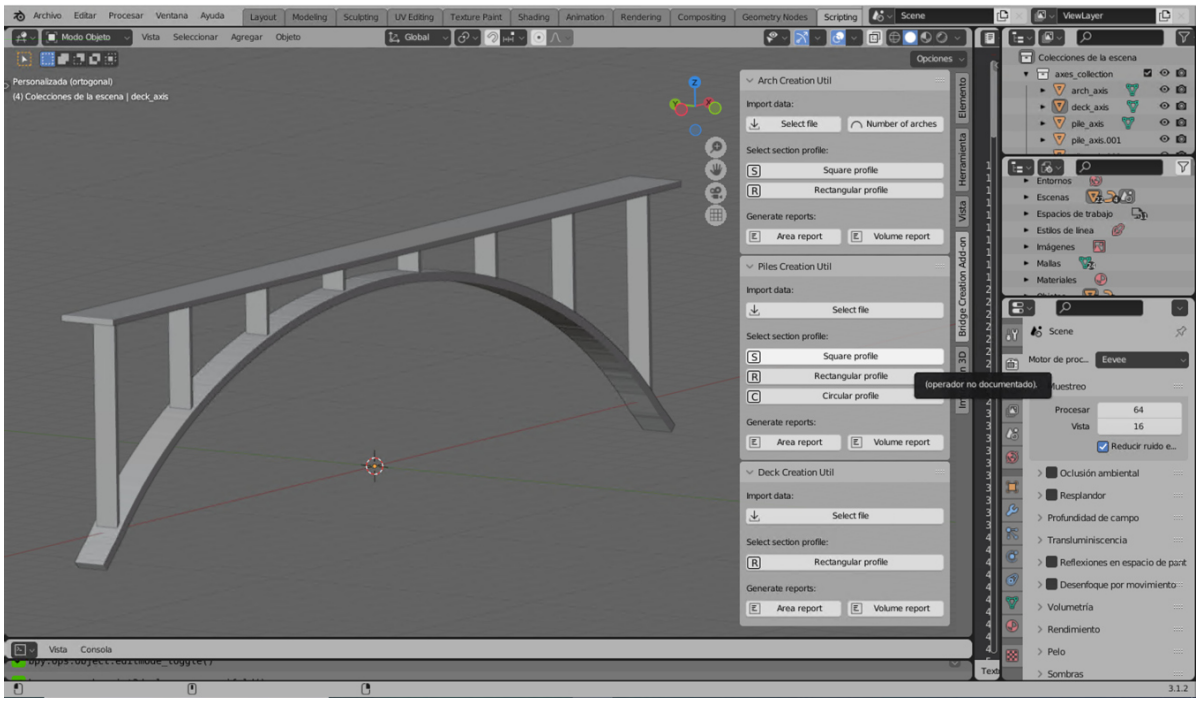
El funcionamiento del add-on así desarrollado se describe a continuación:

- El botón *Arch Creation Util* del menú del add-on permite la selección de las siguientes instrucciones:
    - El botón *Select file* permite seleccionar el archivo de datos externos que se desea importar en Blender.
    - El botón *Number of arches* permite especificar el número de arcos a dibujar y su equidistancia.
    - Los botones *Square profile* y *Rectangular profile* definen la sección transversal del arco. Al pulsar sobre dichos botones, aparece a continuación un menú emergente que permite especificar las dimensiones del arco.
    - Los botones *Area report* y *Volumen report* generan los informes de superficie de encofrado y volumen de hormigón necesarios para la ejecución de los arcos.
- El botón *Piles Creation Util* del menú del add-on permite la selección de las siguientes instrucciones:
    - El botón *Select file* permite seleccionar el archivo de datos externos que se desea importar en Blender.
    - Los botones *Square profile*, *Rectangular profile* y *Circular profile* definen la sección transversal de las péndolas del puente arco. Al pulsar sobre dichos botones, aparece a continuación un menú emergente que permite especificar las dimensiones de dichas péndolas.
    - Los botones *Area report* y *Volumen report* generan los informes de superficie de encofrado y volumen de hormigón necesarios para la ejecución de las péndolas.
- El botón *Deck Creation Util* del menú del add-on permite la selección de las siguientes instrucciones:

- El botón *Select file* permite seleccionar el archivo de datos externos que se desea importar en Blender.
- El botón *Square profile* define la sección transversal define la sección transversal del tablero del puente arco. Al pulsar sobre dicho botón, aparece a continuación un menú emergente que permite especificar las dimensiones del tablero.
- Los botones *Area report* y *Volumen report* generan los informes de superficie de encofrado y volumen de hormigón necesarios para la ejecución del tablero.



(a)



(b)

Figura 33. Captura de pantalla con los archivos *F archUtilOperators.py*, *F pilesUtilOperators.py*, *F deckUtilOperators.py* y *F panel.py* (a) y el menú del add-on que permite la generación automatizada del puente arco (b). Fuente: elaboración propia

A continuación, se hace un desglose de las líneas de código o funciones utilizadas en el trabajo y relacionadas con la generación del puente arco, tanto para el script de modelado del arco *archUtilOpertors.py* (anexo 1) como para el script correspondiente al menú del add-on *panel.py* (anexo 2). No se ha procedido al desglose de los scripts relacionados con las péndolas (anexo 3) y con el tablero (anexo 4) del puente arco por ser muy similares al script desarrollado para el arco. El código completo de modelado geométrico del puente arco se incluye en los anexos 1 a 4.

De esta forma, las distintas líneas de código que se muestran a continuación han sido programadas y complementadas a partir de algunos de los comandos y sentencias ya comentados en el apartado 3. Los distintos comandos así definidos permiten la representación de geometrías complejas a partir del arco programado mediante el código inicial.

Con el objeto de que este apartado resulte accesible a cualquier ingeniero con independencia de los conocimientos que posea sobre programación, se ha procedido a la explicación sucinta sólo de aquellas líneas del código más trascendentales.

ArchUtilOperators.py

```
import bpy, os, csv, bmesh, addon_utils, sympy, math

from bpy.props import StringProperty, BoolProperty

from bpy_extras.io_utils import ImportHelper

from bpy.types import Operator
```

Al igual que en la programación del primer código del arco, en el caso del puente arco se hace necesario como paso previo la importación de las librerías o módulos necesarios para ejecutar el programa. Además de las librerías ya incluidas para el caso del arco, se añaden dos nuevos módulos relacionados con el cálculo de áreas (Sympy y Math).

De la librería “bpy” se importan los submódulos “StringProperty”, “BoolProperty”, “ImportHelper” y “Operator”.

El submódulo “Operator” es necesario para que las clases definidas se ejecuten. Los operadores en Blender ejecutan una acción en el momento en el que son activados. Por lo tanto, si se define una clase y entre paréntesis no se indica “Operator”, ésta no va realizar la acción o acciones asignadas.

El resto de submódulos se utilizan en la clase “OT_ImportFilebrowser” y son necesarios para poder seleccionar y leer el archivo .CSV del que se pretende extraer las coordenadas.

```
# Lists to store the reading file data

verts = []

edges = []

faces = []
```

Estas son las listas que van a almacenar la información necesaria para crear la malla que definirá el eje del arco. Esta creación previa de listas permite ejecutar la función “CreateArchAxis”.

```
def clearLists():

    """Cleans lists to avoid errors when importing new data"""

    verts.clear()

    edges.clear()
```

```
    faces.clear()

    return {'FINISHED'}
```

Esta función “clearLists" sirve para limpiar las listas creadas con el objeto de evitar que la importación de nuevos archivos de datos externos, sin limpiar previamente las listas, impida ejecutar correctamente el programa. La limpieza de listas se ejecuta en la clase “OT_ImportFilebrowser”.

Todas las funciones creadas incluyen una última línea con el texto “return {'FINISHED'}”. La instrucción "return" en Python solicita que se devuelva el valor de la función una vez ejecutada, por eso siempre va al final de las funciones definidas.

Básicamente, cuando se utiliza el elemento “FINISHED”, el operador ejecuta la función y una vez que finaliza, sale de la misma.

```
def createArchAxis(context, filepath):
    """Reads the file and extracts the vertices"""
    with open(filepath, 'r', encoding='utf-8-sig') as csvfile:
```

Se ha establecido el modo lectura con la letra “r” (read).

```
        reader = csv.reader(csvfile, delimiter=',')
        for row in reader:
            vert = (float(row[0]), float(row[2]), float(row[1]))
            verts.append(vert)
        # Joins the vertices
        for i, vert in enumerate(verts):
            startpoint = I
            if i < len(verts)-1:
                endpoint = i+1
            edge = startpoint, endpoint
            edges.append(edge)
    """Creates scene collections"""
    global ax_coll
```

```
    ax_coll = bpy.data.collections.new('axes_collection')
    bpy.context.scene.collection.children.link(ax_coll)
```

Dado que existen determinadas variables que forman parte de más de una función, al definirlas deben ir precedidas del texto “global” con el objeto de que el programa pueda reconocerlas a lo largo de todo el código y funcione correctamente.

```
global prof_coll
    prof_coll = bpy.data.collections.new('profiles_collection')
    bpy.context.scene.collection.children.link(prof_coll)
```

El extracto de código comprendido entre la línea “global ax_coll” y la línea “bpy.context.scene.collection.children.link(prof_coll)” permite organizar la información que se genera dentro del archivo de dibujo, guardando los objetos creados en dos colecciones, una que permite guardar los arcos creados (así como las péndolas y el tablero del puente arco) y otra que permite guardar la forma de la sección transversal del arco, la cual se extruirá a lo largo del eje del mismo.

De esta forma se consigue una correcta organización de la información dentro del archivo de dibujo.

```
    """Creates arch axis"""
    global arch_curve
    # Creates the arch axis mesh
    new_mesh = bpy.data.meshes.new(name='axis_mesh')
    new_mesh.from_pydata(verts, edges, faces)
    ## Creates the object mesh
    new_object = bpy.data.objects.new('arch_axis_curve', new_mesh)
    """ Adds the object mesh to the correspondent collection"""
    ax_coll.objects.link(new_object)
    # Selects the object mesh
    arch = bpy.data.objects.get('arch_axis_curve')
    arch.select_set(True)
```

```
    # Converts the object mesh to a curve
    bpy.context.view_layer.objects.active = bpy.context.scene.objects.get('arch_axis_curve')
    bpy.ops.object.convert(target='CURVE')
    # Renames the arch curve
    arch_curve = bpy.data.objects.get('arch_axis_curve')
    arch_curve.name = 'arch_axis'
    return {'FINISHED'}
```

Esta función importa el archivo .CSV de datos de entrada que contiene el listado de coordenadas que representan todos los puntos de la directriz del arco, extrae los valores que precisa y los guarda en una lista llamada “verts”. Asimismo, la función une los vértices entre sí. Para ello, extrae mediante un bucle *for* los índices de los vértices almacenados en la lista anterior y los almacena a su vez en otra lista llamada “edges” de la siguiente manera: [(0, 1), (1, 2), (2, 3), …].

La función “with open” es la que abre y lee el archivo .CSV.

En determinadas líneas de código se incluye una almohadilla (#) con el objeto de hacer anotaciones dentro del código. Cuando se ejecuta el código, el programa las ignora y continúa su ejecución como si no estuviesen, de igual forma ocurre con el texto encerrado entre comillas consecutivas (‘‘‘ ’’’).

Con esta función el software genera el eje del arco objeto de estudio a partir de los datos almacenados en las listas “verts” y “edges”, y lo guarda en la colección correspondiente. Inicialmente, genera un objeto tipo malla que luego transforma en curva, con el objeto de poder extrusionar posteriormente la sección transversal alrededor del eje.

```
def enable3dPrintAddon():
    addon_name = 'object_print3d_utils'
    isenabled = addon_utils.check(addon_name)[0]
    if not isenabled:
        addon_utils.enable('object_print3d_utils')
    return {'FINISHED'}
```

A través de esta función el programa verifica si el add-on “3D-Print Toolbox” se encuentra habilitado en el entorno de trabajo de Blender. En caso contrario, lo habilita.

Este add-on trae consigo una función que permite “hacer desplegables” las mallas de los objetos, es decir, convertir las mallas en tipo “manifold”, con el objeto de que el cálculo del volumen sea correcto. A diferencia del arco y del tablero, para el caso de las péndolas se ha procedido al cálculo manual de su volumen al estar constituidas por varios elementos.

```
def makeManifold():
    # Selects arch axis
    bpy.data.objects.get('arch_axis').select_set(True)
    bpy.context.view_layer.objects.active = bpy.data.objects['arch_axis']
    # Makes mesh manifold
    bpy.ops.mesh.print3d_clean_non_manifold()
    bpy.ops.object.select_all(action='DESELECT')
    return {'FINISHED'}
```

Esta función convierte la malla del arco generado en tipo manifold, haciendo uso de la función correspondiente del add-on 3D-Print Toolbox. Como paso previo, selecciona la malla del arco.

```
def createSquareExtrusionProfile(width):
    # Creates the section profile
    bpy.ops.mesh.primitive_plane_add(enter_editmode=False, align='WORLD', location=(0,0,
0), scale=(1, 1, 1))
    # Sets the thickness of the arch
    bpy.context.object.dimensions = (width, width, 0)
    # Converts the object to a curve
    bpy.ops.object.convert(target='CURVE')
    # Adds the section profile to the correspondent collection
    square_profile = bpy.context.scene.objects.get('Plano')
    prof_coll.objects.link(square_profile)
    # Remove the section profile from the scene collection
```

```
    bpy.context.scene.collection.objects.unlink(square_profile)
    # Renames the section profile
    square_profile.name = 'square_profile'
    bpy.context.object.data.name = 'square_profile_curve'
    # Selects arch axis
    bpy.context.active_object.select_set(False)
    bpy.data.objects.get('arch_axis').select_set(True)
    bpy.context.view_layer.objects.active = bpy.data.objects['arch_axis']
    # Assigns section profile to arch axis
    bpy.context.object.data.bevel_mode = 'OBJECT'
    bpy.context.object.data.use_fill_caps = True # To close the mesh
    bpy.context.object.data.bevel_object = square_profile
    # Transforms extruded profile along curve to mesh
    bpy.ops.object.convert(target='MESH')
    # Hides section profile
    for obj in bpy.data.collections['profiles_collection'].all_objects:
        if 'square' in obj.name:
            bpy.context.view_layer.objects.active = obj
            bpy.context.object.hide_set(True)
    bpy.ops.object.select_all(action='DESELECT')
    return {'FINISHED'}
```

Esta función crea una sección transversal cuadrada, en el caso de que el usuario indique al programa que desea esa opción. Pide como argumento el ancho (“width”). Una vez que la sección transversal se ha extrusionado alrededor de la curva que conforma el eje del arco, vuelve a transformar el conjunto en una malla, ocultando el cuadrado creado.

```
def createRectangularExtrusionProfile(width, height):
    # Creates the section profile
```

```
  bpy.ops.mesh.primitive_plane_add(enter_editmode=False, align='WORLD', location=(0, 0, 0), scale=(1, 1, 1))
    # Sets the thickness of the arch
    bpy.context.object.dimensions = (width, height, 0)
    # Converts the object to a curve
    bpy.ops.object.convert(target='CURVE')
    # Adds the section profile to the created collection
    rectangular_profile = bpy.context.scene.objects.get('Plano')
    prof_coll.objects.link(rectangular_profile)
    # Remove the section profile from the scene collection
    bpy.context.scene.collection.objects.unlink(rectangular_profile)
    # Renames the section profile
    rectangular_profile.name = 'rectangular_profile'
    bpy.context.object.data.name = 'rectangular_profile_curve'
    # Selects arch axis
    bpy.context.active_object.select_set(False)
    bpy.data.objects.get('arch_axis').select_set(True)
    bpy.context.view_layer.objects.active = bpy.data.objects['arch_axis']
    # Assigns section profile to arch axis
    bpy.context.object.data.bevel_mode = 'OBJECT'
    bpy.context.object.data.use_fill_caps = True # To close the mesh
    bpy.context.object.data.bevel_object = rectangular_profile
    # Transforms extruded profile along curve to mesh
    bpy.ops.object.convert(target='MESH')
    # Hides section profile
     for obj in bpy.data.collections['profiles_collection'].all_objects:
        if 'rectangular' in obj.name:
            bpy.context.view_layer.objects.active = obj
```

```
        bpy.context.object.hide_set(True)

    bpy.ops.object.select_all(action='DESELECT')

    return {'FINISHED'}
```

Esta función crea una sección transversal rectangular, en el caso de que el usuario indique al programa que desea esa opción. Pide como argumentos el ancho (“width”) y el canto (“height”). Una vez que la sección transversal se ha extrusionado alrededor de la curva que conforma el eje del arco, vuelve a transformar el conjunto en una malla, ocultando el rectángulo creado.

```
def multiplyArch(arch_num, equidist):

    # Defines variables relative to coordinates

    x = arch_curve.location[0]

    z = arch_curve.location[2]

    # Selects arch axis

    bpy.context.active_object.select_set(False)

    bpy.data.objects.get('arch_axis').select_set(True)

    bpy.context.view_layer.objects.active = bpy.data.objects['arch_axis']

    # Loop to multiply the arch

    i = 1 # Si empiezo en 0, tendría el tendría el número de arcos especificado más el arco original

    while i < arch_num:

bpy.ops.object.duplicate_move_linked(OBJECT_OT_duplicate={"linked":True, "mode":'TRANSLATION'}, TRANSFORM_OT_translate={"value":(x, equidist, z)})

    i += 1

    # Deselects arch

    bpy.context.active_object.select_set(False)

    return {'FINISHED'}
```

Esta función permite generar el número de arcos que decida el usuario y los coloca desfasados sobre el eje “y”. Pide como argumentos el número de arcos y su equidistancia.

```
def surfaceFormworkArch(width, height):
```

```
## Lower surface
# Extracts the free span value of the arch
arch_start = abs(verts[0][0])
arch_end = abs(verts[-1][0])
free_span = arch_start + arch_end
# Extracts the maximum height of the arch
pos0, pos1, pos2 = max(verts, key=lambda item: item[2])
arch_height = pos2
# Calculates the radius of the circumference
R = sympy.symbols('R')
equation = sympy.Eq((R-arch_height)**2 + (free_span/2)**2, R**2)
result = sympy.solve(equation, R)
circ_radius = round(result[0], 2)
# Calculates the circular sector angle
sector_angle = math.degrees(math.atan((free_span/2) / (circ_radius-arch_height))) * 2
# Calculates the arch length
arch_length = round((2 * math.pi * circ_radius * sector_angle) / 360, 2)
# Calculates the lower arch surface
global low_arch_surf
low_arch_surf = round(arch_length * width, 2)
global total_low_arch_surf
total_low_arch_surf = round(low_arch_surf * arch_num, 2)
## Lateral surface
global lat_arch_surf
lat_arch_surf = round(((arch_length + width) * height) * 2, 2)
global total_lat_arch_surf
total_lat_arch_surf = round(lat_arch_surf * arch_num, 2)
```

```
    return {'FINISHED'}
```

Esta función calcula el área de encofrado necesaria para la ejecución del arco. En ella se hace uso de las librerías SymPy y “mpmath”. La función da como resultado la superficie de encofrado de la cara inferior del arco y de las caras laterales del mismo, en metros cuadrados, para lo cual hace uso de los valores “width” y “height” introducidos manualmente por el usuario.

```
def calculateVolume():
    bm = bmesh.new()
    bm.from_mesh(bpy.context.object.data)
    global single_arch_vol
    single_arch_vol = round(float(bm.calc_volume()), 2)
    global total_vol
    total_vol = round(single_arch_vol * arch_num, 2)
     return {'FINISHED'}
```

Esta función calcula el volumen de un arco y el volumen total de los arcos generados, en metros cúbicos. A su vez, redondea el resultado a 2 decimales. Para calcular el volumen hace uso de la librería “bmesh”.

```
class OT_ImportFilebrowser(Operator, ImportHelper):
    bl_idname = "test.import_arch_filebrowser"
    bl_label = "Import file"
    bl_description = "Allows importing an arch axis coordinate dataset"
    filter_glob: StringProperty(
        default='*.txt;*.csv',
        options={'HIDDEN'}
    )
    def execute(self, context):
    enable3dPrintAddon()
    clearLists()
```

```
createArchAxis(context, self.filepath)

return {'FINISHED'}
```

Esta es la primera clase que incluye el código. Las clases son aquellos procesos que queremos que realice el add-on y cada una puede incluir un conjunto de funciones de las definidas con anterioridad.

La clase “OT_ImportFilebrowser” permite que se abra una ventana emergente por la que navegar hasta encontrar el archivo .CSV desde el que importar las coordenadas para crear los ejes de los elementos a modelar en Blender.

La estructura de todas las clases definidas en el add-on del trabajo se define a continuación:

- 3 líneas iniciales identificativas: “bl_idname”, “bl_label” y “bl_description”.
  - El texto que se escribe a continuación de “bl_idname” es coincidente con el que se escribe en el script llamado “panel.py”, en el que se configuran los botones del menú del add-on. Dicho texto sirve para llamar a la clase desde el script y que ésta se ejecute al pulsar en el botón correspondiente.
  - El texto de la segunda línea (“bl_label”) es el que aparecerá en el botón del menú del add-on.
  - La tercera línea (“bl_description”) permite que aparezca un texto explicativo sobre la función de un determinado botón del menú del add-on al situarse sobre él sin pulsarlo. Dicho texto es el que se incluye en la línea “bl_description”.
- Una función llamada “execute” que tiene como argumentos las palabras “self” y “context”. Dentro de esta función se escriben de manera indentada todas aquellas funciones definidas al inicio del código que queramos que se ejecuten dentro de esta clase.
- Una función llamada “invoke”. Para el caso particular de esta clase no resulta necesario incluir esta función, pero sí para el resto de clases dado que permite que el botón del menú que ejecuta las funciones de una determinada clase aparezcan.

Para que el código funcione correctamente, es necesario registrar las clases. En este sentido, se puede operar de dos formas diferentes:

1. Las clases se pueden registrar una a una al final del código, haciendo uso de unas funciones llamadas “register” y “unregister”:

```
def register():
    OT_ImportFilebrowser(Operator, ImportHelper)
    WM_OT_NumberOfArches(Operator)
. . .

def unregister():
    OT_ImportFilebrowser(Operator, ImportHelper)
    WM_OT_NumberOfArches(Operator)
. . .
```

2. En el caso de que se disponga de muchas clases, se puede crear una lista en la que se nombren a todas las clases del código, utilizando después un bucle for que las recorra :

```
classes = [
    OT_ImportFilebrowser,
    WM_OT_NumberOfArches,
    WM_OT_SquareProfile,
    WM_OT_RectangularProfile,
    WM_OT_FormworkAreaReport,
    WM_OT_ConcreteVolumeReport,
]
def register():
    from bpy.utils import register_class
    for cls in classes:
        register_class(cls)
def unregister():
    from bpy.utils import unregister_class
    for cls in reversed(classes):
```

```
                    unregister_class(cls)
```

Finalmente, y con el objeto de que se ejecuten únicamente las clases registradas y no todo el código, se incluirá lo que se indica a continuación:

```
if __name__ == "__main__":
    register()
class WM_OT_NumberOfArches(Operator):
    bl_idname = "wm.arches_number"
    bl_label = "Number Of Arches Dialogue Box"
    bl_description = "Indicate the number of arches to be drawn"
    arch_num_val : bpy.props.IntProperty(name= "Number of arches to draw: ", default= 1)
    equidist_val : bpy.props.FloatProperty(name= "Equidistance (m): ", default= 1)
    def execute(self, context):
        global arch_num
        arch_num = self.arch_num_val
        equidist = self.equidist_val
        multiplyArch(arch_num, equidist)
        return {'FINISHED'}
    def invoke(self, context, event):
        return context.window_manager.invoke_props_dialog(self)
```

Esta clase se encarga de modelar tantos arcos como el usuario solicite por pantalla. Esto es posible gracias a las líneas “arch_num_val” y “equidist_val”.

```
class WM_OT_SquareProfile(Operator):
    bl_idname = "wm.arch_square_profile"
    bl_label = "Square Profile Dialogue Box"
    bl_description = "Extrudes a square profile along the arch axis"
    # Asks the value of the dimension in the dialog box
    width_val : bpy.props.FloatProperty(name= "Width (m): ", default= 1)
```

```
    def execute(self, context):
        global width
        width = self.width_val # Associates the dimensions with the specified value
        global height
        height = width # To avoid problems in the calculation of area and volume
        createSquareExtrusionProfile(width)
        makeManifold()
        return {'FINISHED'}
    def invoke(self, context, event):
        """Invokes the dialog box"""
        return context.window_manager.invoke_props_dialog(self)
```

Esta clase tiene por objeto la extrusión del perfil. Tras su creación, se ejecuta la función "makeManifold" para que el cálculo del volumen del arco sea correcto.

```
class WM_OT_RectangularProfile(Operator):
    bl_idname = "wm.arch_rectangular_profile"
    bl_label = "Rectangular Profile Dialogue Box"
    bl_description = "Extrudes a rectangular profile along the arch axis"
    width_val : bpy.props.FloatProperty(name= "Width (m): ", default= 1)
    height_val : bpy.props.FloatProperty(name= "Height (m): ", default= 1)
    def execute(self, context):
        global width
        width = self.width_val
        global height
        height = self.height_val
        createRectangularExtrusionProfile(width, height)
        makeManifold()
        return {'FINISHED'}
```

```
    def invoke(self, context, event):
        return context.window_manager.invoke_props_dialog(self)
```

Esta clase tiene el mismo objeto que la anterior, pero para el caso de arcos de secciones transversales rectangulares.

```
class WM_OT_FormworkAreaReport(Operator):
    bl_idname = "wm.formwork_arch_area_report"
    bl_label = "Formwork Info Area"
    bl_description = "Generates a .txt report with the formwork areas required to execute the modelled arches"
    def execute(self, context):
        surfaceFormworkArch(width, height)
        with open('archAreaReport.txt', 'w', encoding='utf-8') as areaReportFile:
```

En este caso, estamos creando un archivo con extensión .txt para almacenar los datos de áreas de encofrado del arco modelado. Aunque se utilice la función “with open”, realmente se crea un nuevo archivo TXT en la misma ruta en la que está situado el archivo de Blender, y se utiliza la letra “w” para indicar al programa que se quiere abrir en modo escritura, no lectura.

```
            areaReportFile.write('------------ FORMWORK ARCH AREA REPORT ------------' + '\n' + '\n')
            areaReportFile.write('- One-arch lower formwork surface = ' + str(low_arch_surf) + ' m²' + '\n')
            areaReportFile.write('- Total lower formwork surface = ' + str(total_low_arch_surf) + ' m²' + '\n')
            areaReportFile.write('- One-arch lateral formwork surface = ' + str(lat_arch_surf) + ' m²' + '\n')
            areaReportFile.write('- Total lateral formwork surface = ' + str(total_lat_arch_surf) + ' m²' + '\n' + '\n')
            areaReportFile.write('-------------------------------------------------' + '\n')
    return {'FINISHED'}
```

Con esta clase se extraen los valores de superficie de encofrado calculados y se escriben en un archivo TXT. El archivo en cuestión estará compuesto por 6 líneas de texto, tal y como se desprende del código. En este sentido, resulta trascendental transformar los valores numéricos de áreas a un dato tipo string (str) o cadena de texto.

```
class WM_OT_ConcreteVolumeReport(Operator):
    bl_idname = "wm.concrete_arch_volume_report"
```

```
    bl_label = "Concrete Info Volume"
    bl_description = "Generates a .txt report with the arch volumes"
    def execute(self, context):
        calculateVolume()
        with open('archVolumeReport.txt', 'w', encoding='utf-8') as volumeReportFile:
       volumeReportFile.write('------------ CONCRETE ARCH VOLUME REPORT ------------' + '\n'
+ '\n')
                volumeReportFile.write('- One-arch volume = ' + str(single_arch_vol) + ' m³' + '\n')
                volumeReportFile.write('- Total volume = ' + str(total_vol) + ' m³' + '\n' + '\n')
                volumeReportFile.write('-----------------------------------------------------' + '\n')
        return {'FINISHED'}
```

Esta clase tiene el mismo objeto que la anterior, pero para la generación del informe del volumen de hormigón.

```
classes = [
    WM_OT_NumberOfArches,
    WM_OT_SquareProfile,
    WM_OT_RectangularProfile,
    WM_OT_FormworkAreaReport,
    WM_OT_ConcreteVolumeReport,
]
```

Para que el programa no de error, al final de cada clase a registrar hay que colocar una coma (,), incluso tras la última clase de todas.

```
def register():
    from bpy.utils import register_class
    for cls in classes:
        register_class(cls)
    def unregister():
        from bpy.utils import unregister_class
```

```
    for cls in reversed(classes):
        unregister_class(cls)
if __name__ == "__main__":
    register()
```

Se repite la secuencia de líneas del código, tal y como se ha mencionado con anterioridad.

Panel.py

La visualización del menú del add-on se puede configurar de muchas formas diferentes. A continuación se muestra la seleccionada para este trabajo:

```
import bpy
class PT_ArchUtilPanel(bpy.types.Panel):
    """Creates a Sub-Panel in the Property Area of the 3D View"""
    bl_label = "Arch Creation Util"
    bl_space_type = "VIEW_3D"
    bl_region_type = "UI"
    bl_category = "Bridge Creation Add-on"
```

Los parámetros “bl_space_type” y “bl_region_type” son función del lugar en el que se quiera que aparezca el menú.

```
def draw(self, context):
    layout = self.layout
    """Import data"""
    layout.label(text="Import data:")
    row = layout.row()
```

Los “row = layout.row()” son líneas en blanco que se dejan para separar los distintos botones del menú del add-on.

```
    row.operator("test.import_arch_filebrowser", text= "Select file", icon= 'IMPORT')
```

```
row.operator("wm.arches_number", text= "Number of arches", icon= 'SPHERECURVE')
```

Los “row.operator(…)” constituyen los botones del menú del add-on. En cada una de estas líneas se llama a las clases creadas en los otros scripts a través de su “bl_idename”. Con independencia de que se haya asignado una “bl_label” en cada una de las clases, si por la razón que sea se quiere poner al botón un texto diferente, podrá hacerse en el apartado “text= …” de la línea que configura el botón.

Aquí, por ejemplo, estamos llamando a las clases “OT_ImportFilebrowser” y “WM_OT_NumberOfArches”.

```
row = layout.row()

row = layout.row()

"""Select section profile"""

layout.label(text="Select section profile:")

row = layout.row()

row.operator("wm.arch_square_profile", text= "Square profile", icon= 'EVENT_S')
```

El texto “icon= …” al final de cada línea que configura un botón se refiere al icono que aparece en cada uno de los botones. Para poder visualizar los distintos símbolos que pueden emplearse es necesario activar en Editar > Preferencias el add-on “Development: Icon Viewer” que trae Blender por defecto.

```
row = layout.row()

row.operator("wm.arch_rectangular_profile", text= "Rectangular profile", icon= 'EVENT_R')

row = layout.row()

row = layout.row()

"""Reports"""

layout.label(text="Generate reports:")

row = layout.row()

row.operator("wm.formwork_arch_area_report", text= "Area report", icon= 'WORDWRAP_ON')

row.operator("wm.concrete_arch_volume_report", text= "Volume report", icon= 'WORDWRAP_ON')
```

```
    layout.row()
```

Para el caso que nos ocupa, se definen un total de tres clases para cada uno de los submenús del add-on: generación de arcos, generación de péndolas y generación del tablero. Dado que en este caso las clases no se refieren a operadores sino a menús, el argumento que va entre paréntesis junto al nombre de la clase es “bpy.types.Panel”.

```
class PT_PilesUtilPanel(bpy.types.Panel):
    """Creates a Sub-Panel in the Property Area of the 3D View"""
    bl_label = "Piles Creation Util"
    bl_space_type = "VIEW_3D"
    bl_region_type = "UI"
    bl_category = "Bridge Creation Add-on"
    def draw(self, context):
        layout = self.layout
        """Import data"""
        layout.label(text="Import data:")
        row = layout.row()
        row.operator("test.import_piles_filebrowser", text= "Select file", icon= 'IMPORT')
        row = layout.row()
        row = layout.row()
        """Select section profile"""
        layout.label(text="Select section profile:")
        row = layout.row()
        row.operator("wm.piles_square_profile", text= "Square profile", icon= 'EVENT_S')
        row = layout.row()
        row.operator("wm.piles_rectangular_profile", text= "Rectangular profile", icon= 'EVENT_R')
        row = layout.row()
        row.operator("wm.piles_circular_profile", text= "Circular profile", icon= 'EVENT_C')
```

```
        row = layout.row()

        row = layout.row()

        """Reports"""

        layout.label(text="Generate reports:")

        row = layout.row()

        row.operator("wm.formwork_piles_area_report", text= "Area report", icon= 'WORDWRAP_ON')

        row.operator("wm.concrete_piles_volume_report", text= "Volume report", icon= 'WORDWRAP_ON')

        layout.row()

class PT_DeckUtilPanel(bpy.types.Panel):

    """Creates a Sub-Panel in the Property Area of the 3D View"""

    bl_label = "Deck Creation Util"

     bl_space_type = "VIEW_3D"

    bl_region_type = "UI"

    bl_category = "Bridge Creation Add-on"

    def draw(self, context):

        layout = self.layout

        """Import data"""

        layout.label(text="Import data:")

        row = layout.row()

        row.operator("test.import_deck_filebrowser", text= "Select file", icon= 'IMPORT')

        row = layout.row()

        row = layout.row()

        """Select section profile"""

        layout.label(text="Select section profile:")

        row = layout.row()

        row.operator("wm.deck_rectangular_profile", text= "Rectangular profile", icon= 'EVENT_R')
```

```
        row = layout.row()
        row = layout.row()
        """Reports"""
        layout.label(text="Generate reports:")
        row = layout.row()
        row.operator("wm.formwork_deck_area_report", text= "Area report", icon= 'WORDWRAP_ON')
        row.operator("wm.concrete_deck_volume_report", text= "Volume report", icon= 'WORDWRAP_ON')
        layout.row()
    classes = [
        PT_ArchUtilPanel,
        PT_PilesUtilPanel,
        PT_DeckUtilPanel,
    ]
    def register():
        from bpy.utils import register_class
        for cls in classes:
            register_class(cls)
    def unregister():
        from bpy.utils import unregister_class
        for cls in reversed(classes):
            unregister_class(cls)
    if __name__ == "__main__":
        register()
```

3.4.2 Incidencias

Durante el desarrollo y ejecución de este segundo código se han registrado varias incidencias de interés, tales como las que se enumeran a continuación:

1.- Primer error que arroja el código durante la ejecución del add-on para la generación automática de arcos (figura 34):

ModuleNotFoundError: No module named ´sympy´

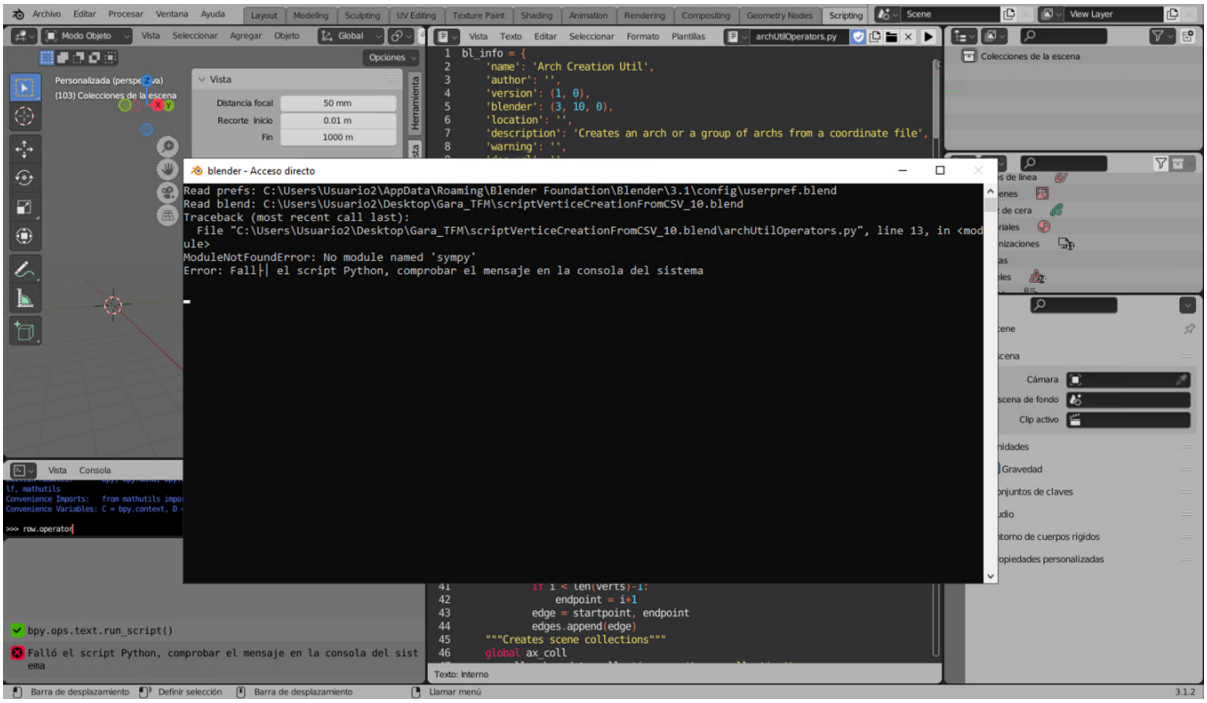


Figura 34. Captura de pantalla de la primera incidencia detectada durante el proceso de ejecución del segundo código. Fuente: elaboración propia

Este primer error está relacionado con la librería SymPy, la cual resulta necesario para realizar cálculos de áreas a través del código programado. Para que el código funcione correctamente, las carpetas del módulo SymPy deben guardarse dentro de la carpeta raíz de Blender, siguiendo la ruta que se indica a continuación:

C:\Program Files\Blender Foundation\Blender 3.1\3.1\python\lib\site-packages

Cabe mencionar que la ruta puede variar en función del lugar de instalación de Blender o de la versión del programa que se esté utilizando.

2.- Segundo error que arroja el código durante la ejecución del add-on para la generación automática de arcos (figura 35):

ModuleNotFoundError: No module named ´mpmath´

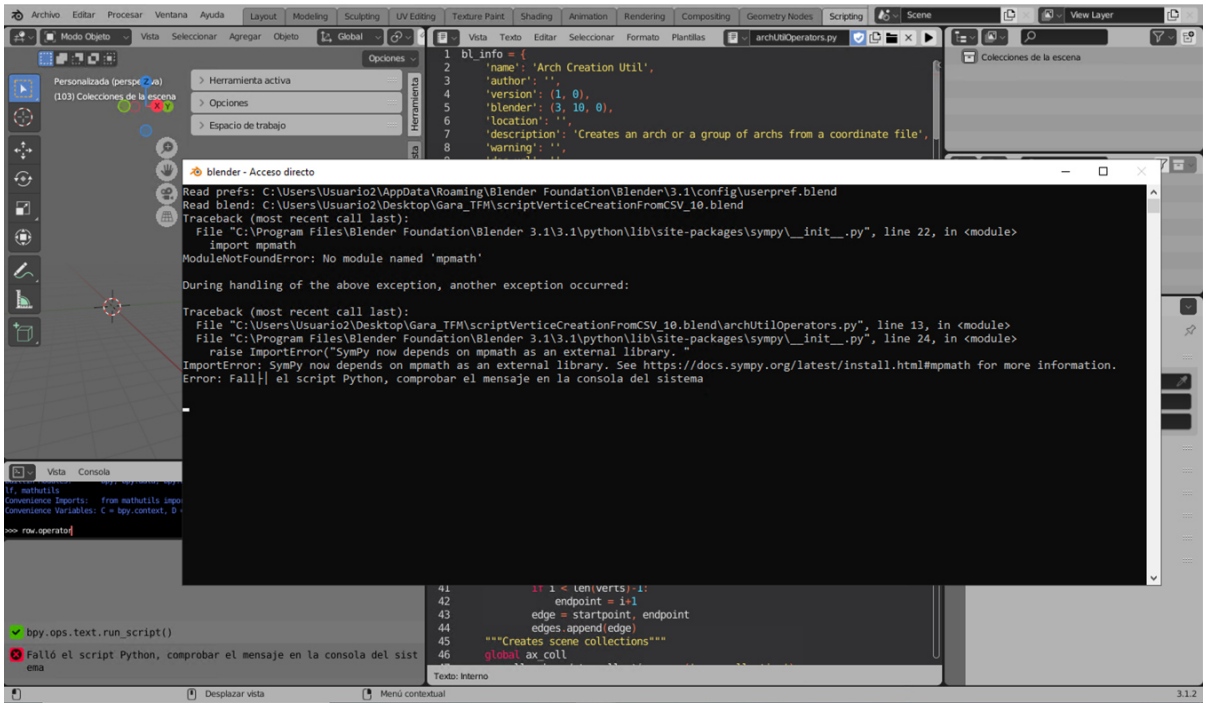


Figura 35. Captura de pantalla de la segunda incidencia detectada durante el proceso de ejecución del segundo código. Fuente: elaboración propia

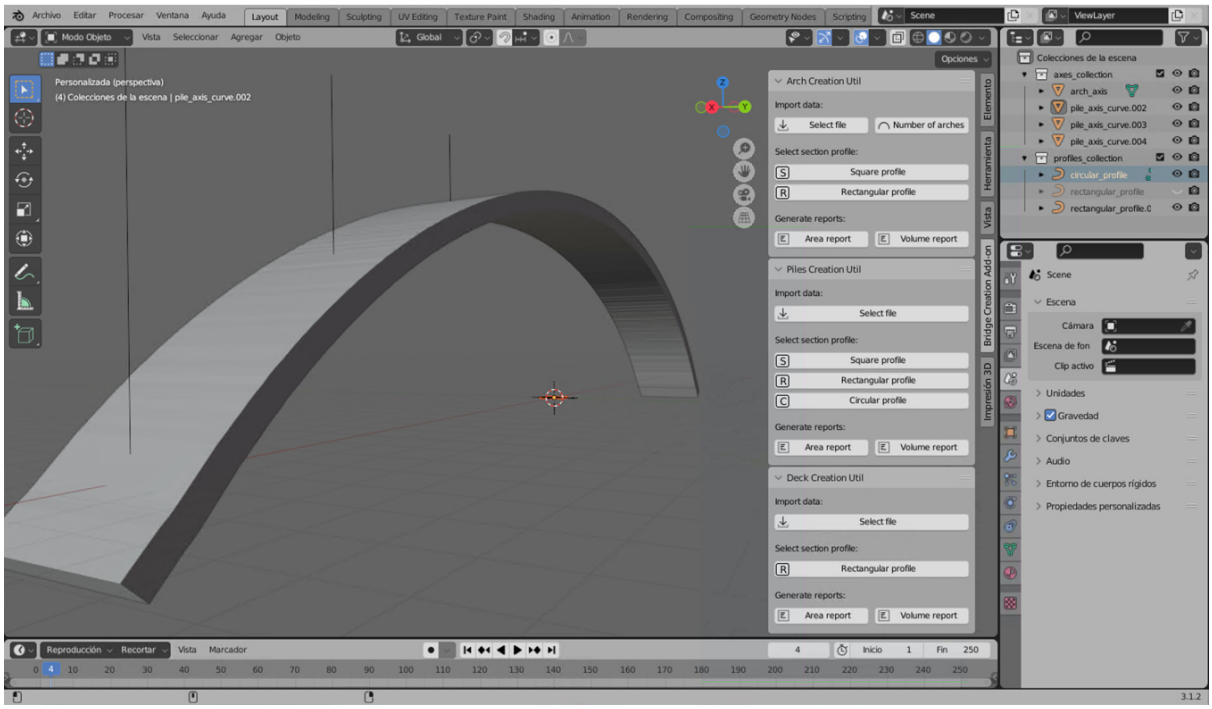
Al igual que para el caso anterior, este error está relacionado con el módulo mpmath, el cual también debe incluirse en la ruta de módulos de Blender para que el código funcione con normalidad. La librería SymPy hace uso de este módulo.

3.- La tercera incidencia detectada durante el desarrollo del segundo código para la generación automática del puente arco está relacionada con el script de las péndolas del puente (figura 36).

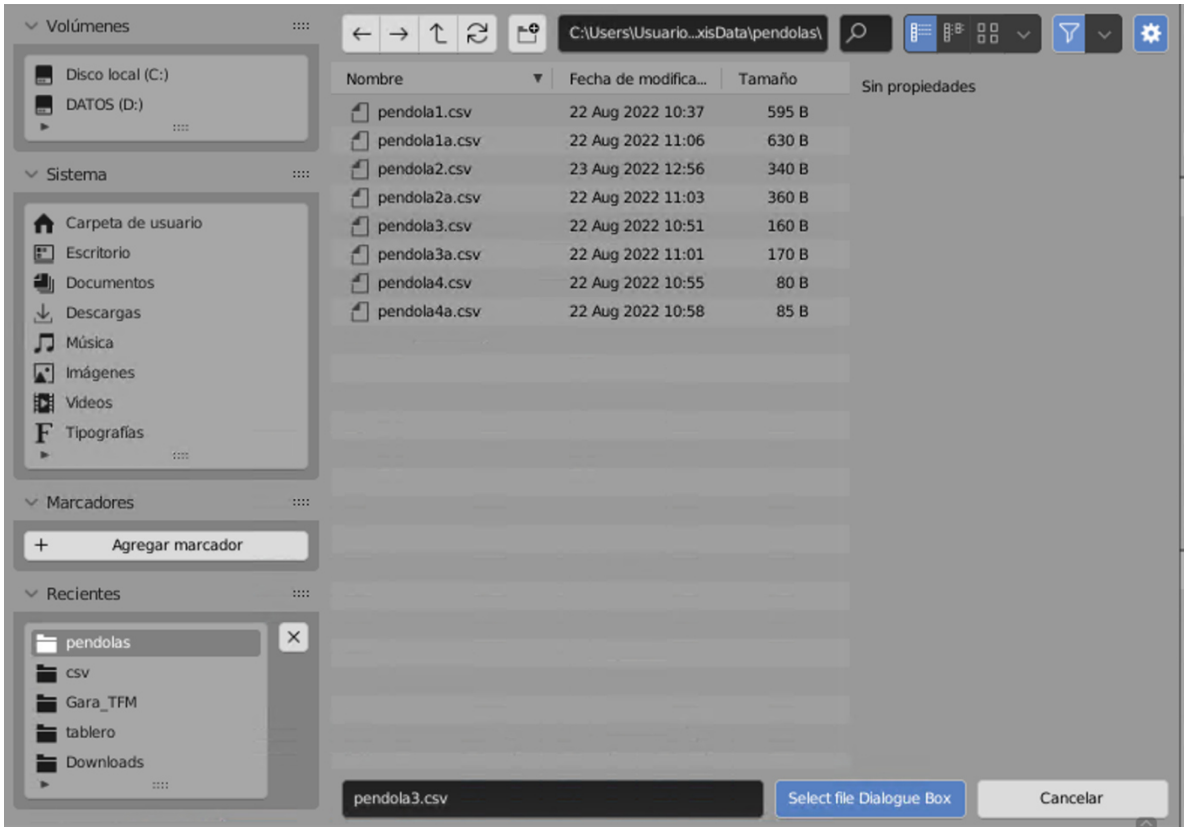
A diferencia de lo que ocurre con el arco (y también con el tablero), cuya definición geométrica se ha realizado previamente mediante un listado de coordenadas que representan todos los puntos de su directriz, en Blender resulta necesario que la geometría de las péndolas esté basada en un total de ocho archivos externos que contienen la definición de su geometría, uno para cada péndola, con el objeto de que el programa sea capaz de procesar dicha geometría.

Es decir, el código programado no es capaz de importar a través de un único listado de coordenadas la información geométrica relacionada con las ocho péndolas que constituyen el puente arco, lo cual constituye una debilidad del mismo cuando se trate de un número elevado de elementos a modelar, perdiendo efectividad en su ejecución.

Las figuras que se muestran a continuación representan la fase de modelado del eje de tres de las ocho péndolas que conforman el puente arco.



(a)



(b)

Figura 36. Captura de pantalla de la tercera incidencia detectada durante el proceso de ejecución del script del puente arco, con el eje de tres de las ocho péndolas (a) y con una imagen de la carpeta desde donde se importan los datos externos necesarios para su modelado (b)

4. RESULTADOS Y CONCLUSIONES

En este trabajo se emplea *el arco* como tipo estructural esencial para demostrar la aplicabilidad del software Blender en los sistemas más representativos de la ingeniería civil, tales como los puentes arco, los túneles y las presas arco.

Empleando el arco como tipo estructural esencial, se desarrolla un código que permite el procesado automático de la definición de geometrías complejas tales como el *puente arco*, mediante el ensamblaje de geometrías más simples, y la obtención de determinadas características y variables asociadas a esa geometría. A la geometría así generada se le asigna, además, una clase IFC a través de la librería ifcOpenShell.

Para este código se ha desarrollado un add-on que amplía las funcionalidades básicas del software Blender al mostrarse en pantalla un menú totalmente personalizado a través del cual puede ejecutarse el programa.

En los ejemplos mostrados, el código programado permite la integración en un software opensource de la definición de geometrías complejas a partir de elementos básicos, así como la obtención de forma ágil y totalmente automatizada de determinados atributos de interés de la misma, tales como el volumen de la estructura generada o la superficie necesaria de encofrado.

En contexto BIM, el procedimiento estándar consiste en ejecutar geometrías a través de la combinación de varios programas que permitan, a su vez, obtener el resto de variables que se precisen. La implementación de un código programado con Python dota de mayor efectividad a este procedimiento al prescindir de otros programas necesarios a tal fin. De esta forma, el procedimiento convencional resulta más complejo al requerir el empleo de varios programas de forma independiente.

Los diferentes procesos así implementados se validan mediante su aplicación en el diseño y planificación de unidades concretas integradas en proyectos de ingeniería estructural, con el posterior objeto de incorporar este proceso en el ámbito de la optimización estructural.

Este trabajo demuestra que la aplicabilidad de Blender mediante la programación en Python en el procesado geométrico de sistemas de ingeniería civil es posible.

5. LÍNEAS DE TRABAJO FUTURO

La línea de trabajo futuro tiene por objeto la incorporación del procesado geométrico de tipos estructurales a través de Blender en otros ámbitos tales como la optimización estructural.

En un proceso de optimización estructural existe una función objetivo, que puede maximizarse o minimizarse. En los experimentos que se han realizado con arcos hasta la fecha, una de las variables de respuesta de la estructura que se incluyen dentro de esa función objetivo es el momento flector del arco. A tal fin, la inclusión de dicho flector dentro de la función objetivo a optimizar exige realizar una llamada en modo batch a un software de cálculo de estructuras.

Un punto de encuentro entre estos procedimientos de optimización y la tecnología BIM podría consistir en ampliar la función objetivo mediante la inclusión en la misma de otros resultados de interés que fueran calculables mediante un software BIM. Entre estos nuevos resultados podría encontrarse el costo total de ejecución de la estructura del arco, o su plazo de ejecución.

De esta forma, en cada uno de los arcos candidatos que el optimizador va evaluando durante el proceso de optimización, además de calcular el flector o cualquier resultado mecánico del candidato, también podrían calcularse otros resultados de interés como son su coste de construcción o su plazo de ejecución (utilizando para ello el software BIM), de forma que todos estos resultados puedan introducirse en la función objetivo global del problema, lo que permitiría al optimizador decidir tomando como referencia ya no sólo la respuesta estructural del arco, sino también otros aspectos de interés del mismo relacionados con su ejecución y su coste.

Para poder hacer esto, es necesario que el software BIM pueda ser ejecutado en modo batch (con todo lo que ello implica: abrir el software BIM desde una línea de comando de un código externo, leer archivos tanto de datos de entrada como de instrucciones, y generar archivos de resultados) a fin de poder integrar el BIM en la rutina de optimización y automatizar todo el proceso conjunto.

BIBLIOGRAFÍA

[1] The Art of Natural Graphic Man-Machine Conversation. J. D. Foley, V. L. Wallace 1974

[2] Seystic, 2020 https://seystic.com/bim-la-historia-del-building-information-modelling/

[3] Augmenting the Human Intellect; A conceptual framework. Douglas Engelbart, 1962

[4] The Use of Computers Instead of Drawings in Building Design. Charles M. Eastman, 1975

[5] Building Modelling the key to integrated construction CAD. Robert Aish, 1986

[6] Historia de Blender https://docs.blender.org/manual/es/dev/getting_started/about/history.html

[7] A report for the Government Construction Client Group. Building Information Modelling (BIM) Working Party Strategy Paper

[8] Directiva 2014/24/UE del Parlamento Europeo y del Consejo de 26 de febrero de 2014, sobre contratación pública

[9] Los puentes, ahora con BRIM. José Carlos Lino, 2016 https://www.bimcommunity.com/news/load/141/los-puentes-ahora-con-brim

[10] Guía de inspecciones básicas de obras de paso. Red de Carreteras del Estado. Ministerio de Fomento, 2009

[11] A Parameter-Driven Method for Modeling Bridge Defects through IFC. Shuyuan Xu, Jun Wang, Xiangyu Wang, Peng Wu, Wenchi Shou and Chao Liu, 2020

[12] Towards intelligent structural design of buildings: A BIM-based solution. Tofigh Hamidavi, Sepehr Abrishami, M. Reza Hosseini, 2020

[13] BIM-Enabled Structural Design: Impacts and Future Developments in Structural Modelling, Analysis and Optimisation Processes. Hung-Lin Chi, Xiangyu Wang, Yi Jiao, 2014

[14] Sustainability and Interoperability: An Economic Study on BIM Implementation by a Small Civil Engineering Firm. José Ángel Aranda, Norena Martín-Dorta, Ferrán Naya, Julián Conesa-Pastor and Manuel Contero, 2020

[15] A parametric BIM approach to foster bridge project design and analysis. Alexis Girardet, Conrad Boton, 2021

[16] Improving interoperability between architectural and structural design models: An industry foundation classes-based approach with web-based tools. Zhen-Zhong Hu, Xiao-Yang Zhang, Heng-Wei Wang and Mohamad Kassem, 2016

[17] Assessment of model-based data exchange between architectural design and structural analysis. Goran Sibenik, Iva Kovacic, 2020

[18] The IFC-Bridge project - Extending the IFC standard to enable high-quality exchange of bridge information models. André Borrmann, Sergej Muhič, Juha Hyvärinen, Tim Chipman, Stefan Jaud, Christophe Castaing, Claude Dumoulin, Thomas Liebich, Laura Mol, 2019

[19] Application of 3D Bridge Information Modeling to Design and Construction of Bridges. CS Shim, NR Yun, HH Song, 2011

[20] Bridge damage: Detection, IFC-based semantic enrichment and visualization. Dušan Isailović, Vladeta Stojanovic, Matthias Trapp, Rico Richter, Rade Hajdin, Jürgen Döllner, 2020

[21] Uso de Blender a través de la programación en la ingeniería de caminos, canales y puertos. Miguel Puech Oriol, 2021

[22] Measurelt-ARCH: A Tool for Facilitating Architectural Design in the Open Source Software Blender. Kevan Cress, 2020

[23] Razón y ser de los tipos estructurales. Eduardo Torroja Miret, 2007

ANEXO 1. Código del script del puente arco (archUtilOperators.py)

archUtilOperators.py

```
import bpy, os, csv, bmesh, addon_utils, sympy, math

from bpy.props import StringProperty, BoolProperty
from bpy_extras.io_utils import ImportHelper
from bpy.types import Operator
# Lists to store the reading file data

verts = []

edges = []

faces = []

def clearLists():

    """Cleans lists to avoid errors when importing new data"""

    verts.clear()

    edges.clear()

    faces.clear()

    return {'FINISHED'}

def createArchAxis(context, filepath):

    """Reads the file and extracts the vertices"""

    with open(filepath, 'r', encoding='utf-8-sig') as csvfile:

        reader = csv.reader(csvfile, delimiter=',')

        for row in reader:

            vert = (float(row[0]), float(row[2]), float(row[1]))

            verts.append(vert)

        # Joins the vertices

        for i, vert in enumerate(verts):

            startpoint = i

            if i < len(verts)-1:

                endpoint = i+1
```

```
            edge = startpoint, endpoint

            edges.append(edge)

    """Creates scene collections"""

    global ax_coll

    ax_coll = bpy.data.collections.new('axes_collection')

    bpy.context.scene.collection.children.link(ax_coll)

    global prof_coll

    prof_coll = bpy.data.collections.new('profiles_collection')

    bpy.context.scene.collection.children.link(prof_coll)

    """Creates arch axis"""

    global arch_curve

    # Creates the arch axis mesh

    new_mesh = bpy.data.meshes.new(name='axis_mesh')

    new_mesh.from_pydata(verts, edges, faces)

    ## Creates the object mesh

    new_object = bpy.data.objects.new('arch_axis_curve', new_mesh)

    """ Adds the object mesh to the correspondent collection"""

    ax_coll.objects.link(new_object)

    # Selects the object mesh

    arch = bpy.data.objects.get('arch_axis_curve')

    arch.select_set(True)

    # Converts the object mesh to a curve

    bpy.context.view_layer.objects.active                                    =
bpy.context.scene.objects.get('arch_axis_curve')

    bpy.ops.object.convert(target='CURVE')

    # Renames the arch curve

    arch_curve = bpy.data.objects.get('arch_axis_curve')

    arch_curve.name = 'arch_axis'

    return {'FINISHED'}
```

```
def enable3dPrintAddon():

    addon_name = 'object_print3d_utils'

    isenabled = addon_utils.check(addon_name)[0]

    if not isenabled:

        addon_utils.enable('object_print3d_utils')

    return {'FINISHED'}

def makeManifold():

    # Selects arch axis

    bpy.data.objects.get('arch_axis').select_set(True)

    bpy.context.view_layer.objects.active = bpy.data.objects['arch_axis']

    # Makes mesh manifold

    bpy.ops.mesh.print3d_clean_non_manifold()

    bpy.ops.object.select_all(action='DESELECT')

    return {'FINISHED'}

def createSquareExtrusionProfile(width):

    # Creates the section profile

    bpy.ops.mesh.primitive_plane_add(enter_editmode=False,            align='WORLD',
location=(0, 0, 0), scale=(1, 1, 1))

    # Sets the thickness of the arch

    bpy.context.object.dimensions = (width, width, 0)

    # Converts the object to a curve

    bpy.ops.object.convert(target='CURVE')

    # Adds the section profile to the correspondent collection

    square_profile = bpy.context.scene.objects.get('Plano')

    prof_coll.objects.link(square_profile)

    # Remove the section profile from the scene collection

    bpy.context.scene.collection.objects.unlink(square_profile)

    # Renames the section profile

    square_profile.name = 'square_profile'
```

```
    bpy.context.object.data.name = 'square_profile_curve'

    # Selects arch axis

    bpy.context.active_object.select_set(False)

    bpy.data.objects.get('arch_axis').select_set(True)

    bpy.context.view_layer.objects.active = bpy.data.objects['arch_axis']

    # Assigns section profile to arch axis

    bpy.context.object.data.bevel_mode = 'OBJECT'

    bpy.context.object.data.use_fill_caps = True # To close the mesh

    bpy.context.object.data.bevel_object = square_profile

    # Transforms extruded profile along curve to mesh

    bpy.ops.object.convert(target='MESH')

    # Hides section profile

    for obj in bpy.data.collections['profiles_collection'].all_objects:

        if 'square' in obj.name:

            bpy.context.view_layer.objects.active = obj

            bpy.context.object.hide_set(True)

    bpy.ops.object.select_all(action='DESELECT')

    return {'FINISHED'}

def createRectangularExtrusionProfile(width, height):

    # Creates the section profile

    bpy.ops.mesh.primitive_plane_add(enter_editmode=False,            align='WORLD',
location=(0, 0, 0), scale=(1, 1, 1))

    # Sets the thickness of the arch

    bpy.context.object.dimensions = (width, height, 0)

    # Converts the object to a curve

    bpy.ops.object.convert(target='CURVE')

    # Adds the section profile to the created collection

    rectangular_profile = bpy.context.scene.objects.get('Plano')

    prof_coll.objects.link(rectangular_profile)
```

```
    # Remove the section profile from the scene collection
    bpy.context.scene.collection.objects.unlink(rectangular_profile)
    # Renames the section profile
    rectangular_profile.name = 'rectangular_profile'
    bpy.context.object.data.name = 'rectangular_profile_curve'
    # Selects arch axis
    bpy.context.active_object.select_set(False)
    bpy.data.objects.get('arch_axis').select_set(True)
    bpy.context.view_layer.objects.active = bpy.data.objects['arch_axis']
    # Assigns section profile to arch axis
    bpy.context.object.data.bevel_mode = 'OBJECT'
    bpy.context.object.data.use_fill_caps = True # To close the mesh
    bpy.context.object.data.bevel_object = rectangular_profile
    # Transforms extruded profile along curve to mesh
    bpy.ops.object.convert(target='MESH')
    # Hides section profile
    for obj in bpy.data.collections['profiles_collection'].all_objects:
        if 'rectangular' in obj.name:
            bpy.context.view_layer.objects.active = obj
            bpy.context.object.hide_set(True)
    bpy.ops.object.select_all(action='DESELECT')
    return {'FINISHED'}
def multiplyArch(arch_num, equidist):
    # Defines variables relative to coordinates
    x = arch_curve.location[0]
    z = arch_curve.location[2]
    # Selects arch axis
    bpy.context.active_object.select_set(False)
    bpy.data.objects.get('arch_axis').select_set(True)
```

```
    bpy.context.view_layer.objects.active = bpy.data.objects['arch_axis']

    # Loop to multiply the arch

    i = 1 # Si empiezo en 0, tendría el tendría el número de arcos especificado más
el arco original

    while i < arch_num:

        bpy.ops.object.duplicate_move_linked(OBJECT_OT_duplicate={"linked":True,
"mode":'TRANSLATION'}, TRANSFORM_OT_translate={"value":(x, equidist, z)})

        i += 1

    # Deselects arch

    bpy.context.active_object.select_set(False)

    return {'FINISHED'}

def surfaceFormworkArch(width, height):

    ## Lower surface

    # Extracts the free span value of the arch

    arch_start = abs(verts[0][0])

    arch_end = abs(verts[-1][0])

    free_span = arch_start + arch_end

    # Extracts the maximum height of the arch

    pos0, pos1, pos2 = max(verts, key=lambda item: item[2])

    arch_height = pos2

    # Calculates the radius of the circumference

    R = sympy.symbols('R')

    equation = sympy.Eq((R-arch_height)**2 + (free_span/2)**2, R**2)

    result = sympy.solve(equation, R)

    circ_radius = round(result[0], 2)

    # Calculates the circular sector angle

    sector_angle     =     math.degrees(math.atan((free_span/2)     /     (circ_radius-
arch_height))) * 2

    # Calculates the arch length

    arch_length = round((2 * math.pi * circ_radius * sector_angle) / 360, 2)
```

```
    # Calculates the lower arch surface
    global low_arch_surf
    low_arch_surf = round(arch_length * width, 2)
    global total_low_arch_surf
    total_low_arch_surf = round(low_arch_surf * arch_num, 2)
    ## Lateral surface
    global lat_arch_surf
    lat_arch_surf = round(((arch_length + width) * height) * 2, 2)
    global total_lat_arch_surf
    total_lat_arch_surf = round(lat_arch_surf * arch_num, 2)
    return {'FINISHED'}
def calculateVolume():
    bm = bmesh.new()
    bm.from_mesh(bpy.context.object.data)
    global single_arch_vol
    single_arch_vol = round(float(bm.calc_volume()), 2)
    global total_vol
    total_vol = round(single_arch_vol * arch_num, 2)
    return {'FINISHED'}
class OT_ImportFilebrowser(Operator, ImportHelper):
    bl_idname = "test.import_arch_filebrowser"
    bl_label = "Import file"
    bl_description = "Allows importing an arch axis coordinate dataset"
    filter_glob: StringProperty(
        default='*.txt;*.csv',
        options={'HIDDEN'}
    )
    def execute(self, context):
        enable3dPrintAddon()
```

```
        clearLists()
        createArchAxis(context, self.filepath)
        return {'FINISHED'}
class WM_OT_NumberOfArches(Operator):
    bl_idname = "wm.arches_number"
    bl_label = "Number Of Arches Dialogue Box"
    bl_description = "Indicate the number of arches to be drawn"
    arch_num_val : bpy.props.IntProperty(name= "Number of arches to draw: ", default=
1)
    equidist_val : bpy.props.FloatProperty(name= "Equidistance (m): ", default= 1)
    def execute(self, context):
        global arch_num
        arch_num = self.arch_num_val
        equidist = self.equidist_val
        multiplyArch(arch_num, equidist)
        return {'FINISHED'}
    def invoke(self, context, event):
        return context.window_manager.invoke_props_dialog(self)
class WM_OT_SquareProfile(Operator):
    bl_idname = "wm.arch_square_profile"
    bl_label = "Square Profile Dialogue Box"
    bl_description = "Extrudes a square profile along the arch axis"
    # Asks the value of the dimension in the dialog box
    width_val : bpy.props.FloatProperty(name= "Width (m): ", default= 1)
    def execute(self, context):
        global width
        width = self.width_val # Associates the dimensions with the specified value
        global height
        height = width # To avoid problems in the calculation of area and volume
```

```
        createSquareExtrusionProfile(width)
        makeManifold()
        return {'FINISHED'}
    def invoke(self, context, event):
        """Invokes the dialog box"""
        return context.window_manager.invoke_props_dialog(self)
class WM_OT_RectangularProfile(Operator):
    bl_idname = "wm.arch_rectangular_profile"
    bl_label = "Rectangular Profile Dialogue Box"
    bl_description = "Extrudes a rectangular profile along the arch axis"
    width_val : bpy.props.FloatProperty(name= "Width (m): ", default= 1)
    height_val : bpy.props.FloatProperty(name= "Height (m): ", default= 1)
    def execute(self, context):
        global width
        width = self.width_val
        global height
        height = self.height_val
        createRectangularExtrusionProfile(width, height)
        makeManifold()
        return {'FINISHED'}
    def invoke(self, context, event):
        return context.window_manager.invoke_props_dialog(self)
class WM_OT_FormworkAreaReport(Operator):
    bl_idname = "wm.formwork_arch_area_report"
    bl_label = "Formwork Info Area"
    bl_description = "Generates a .txt report with the formwork areas required to
execute the modelled arches"
    def execute(self, context):
        surfaceFormworkArch(width, height)
```

```
        with open('archAreaReport.txt', 'w', encoding='utf-8') as areaReportFile:

            areaReportFile.write('------------- FORMWORK ARCH AREA REPORT ----------
---' + '\n' + '\n')

            areaReportFile.write('- One-arch lower formwork surface = ' +
str(low_arch_surf) + ' m²' + '\n')

            areaReportFile.write('- Total lower formwork surface = ' +
str(total_low_arch_surf) + ' m²' + '\n')

            areaReportFile.write('- One-arch lateral formwork surface = ' +
str(lat_arch_surf) + ' m²' + '\n')

            areaReportFile.write('- Total lateral formwork surface = ' +
str(total_lat_arch_surf) + ' m²' + '\n' + '\n')

            areaReportFile.write('-----------------------------------------------
----' + '\n')

        return {'FINISHED'}

class WM_OT_ConcreteVolumeReport(Operator):

    bl_idname = "wm.concrete_arch_volume_report"

    bl_label = "Concrete Info Volume"

    bl_description = "Generates a .txt report with the arch volumes"

    def execute(self, context):

        calculateVolume()

        with open('archVolumeReport.txt', 'w', encoding='utf-8') as volumeReportFile:

            volumeReportFile.write('------------ CONCRETE ARCH VOLUME REPORT -------
-----' + '\n' + '\n')

            volumeReportFile.write('- One-arch volume = ' + str(single_arch_vol) + '
m³' + '\n')

            volumeReportFile.write('- Total volume = ' + str(total_vol) + ' m³' +
'\n' + '\n')

            volumeReportFile.write('-----------------------------------------------
------' + '\n')

        return {'FINISHED'}

classes = [

    OT_ImportFilebrowser,

    WM_OT_NumberOfArches,

    WM_OT_SquareProfile,
```

```
    WM_OT_RectangularProfile,
    WM_OT_FormworkAreaReport,
    WM_OT_ConcreteVolumeReport,
]
def register():
    from bpy.utils import register_class
    for cls in classes:
        register_class(cls)
def unregister():
    from bpy.utils import unregister_class
    for cls in reversed(classes):
        unregister_class(cls)
if __name__ == "__main__":
    register()
```

ANEXO 2. Código del script del puente arco (panel.py)

panel.py

```
import bpy

class PT_ArchUtilPanel(bpy.types.Panel):

    """Creates a Sub-Panel in the Property Area of the 3D View"""

    bl_label = "Arch Creation Util"

    bl_space_type = "VIEW_3D"

    bl_region_type = "UI"

    bl_category = "Bridge Creation Add-on"

    def draw(self, context):

        layout = self.layout

        """Import data"""

        layout.label(text="Import data:")

        row = layout.row()

        row.operator("test.import_arch_filebrowser", text= "Select file", icon=
'IMPORT')

        row.operator("wm.arches_number", text= "Number of arches", icon=
'SPHERECURVE')

        row = layout.row()

        row = layout.row()

        """Select section profile"""

        layout.label(text="Select section profile:")

        row = layout.row()

        row.operator("wm.arch_square_profile", text= "Square profile", icon=
'EVENT_S')

        row = layout.row()

        row.operator("wm.arch_rectangular_profile", text= "Rectangular profile",
icon= 'EVENT_R')

        row = layout.row()
```

```
        row = layout.row()

        """Reports"""

        layout.label(text="Generate reports:")

        row = layout.row()

        row.operator("wm.formwork_arch_area_report", text= "Area report", icon=
'WORDWRAP_ON')

        row.operator("wm.concrete_arch_volume_report", text= "Volume report", icon=
'WORDWRAP_ON')

        layout.row()

class PT_PilesUtilPanel(bpy.types.Panel):

    """Creates a Sub-Panel in the Property Area of the 3D View"""

    bl_label = "Piles Creation Util"

    bl_space_type = "VIEW_3D"

    bl_region_type = "UI"

    bl_category = "Bridge Creation Add-on"

    def draw(self, context):

        layout = self.layout

        """Import data"""

        layout.label(text="Import data:")

        row = layout.row()

        row.operator("test.import_piles_filebrowser", text= "Select file", icon=
'IMPORT')

        row = layout.row()

        row = layout.row()

        """Select section profile"""

        layout.label(text="Select section profile:")

        row = layout.row()

        row.operator("wm.piles_square_profile", text= "Square profile", icon=
'EVENT_S')

        row = layout.row()
```

```
        row.operator("wm.piles_rectangular_profile", text= "Rectangular profile",
icon= 'EVENT_R')

        row = layout.row()

        row.operator("wm.piles_circular_profile", text= "Circular profile", icon=
'EVENT_C')

        row = layout.row()

        row = layout.row()

        """Reports"""

        layout.label(text="Generate reports:")

        row = layout.row()

        row.operator("wm.formwork_piles_area_report", text= "Area report", icon=
'WORDWRAP_ON')

        row.operator("wm.concrete_piles_volume_report", text= "Volume report", icon=
'WORDWRAP_ON')

        layout.row()

class PT_DeckUtilPanel(bpy.types.Panel):

    """Creates a Sub-Panel in the Property Area of the 3D View"""

    bl_label = "Deck Creation Util"

    bl_space_type = "VIEW_3D"

    bl_region_type = "UI"

    bl_category = "Bridge Creation Add-on"

    def draw(self, context):

        layout = self.layout

        """Import data"""

        layout.label(text="Import data:")

        row = layout.row()

        row.operator("test.import_deck_filebrowser", text= "Select file", icon=
'IMPORT')

        row = layout.row()

        row = layout.row()

        """Select section profile"""
```

```
        layout.label(text="Select section profile:")

        row = layout.row()

        row.operator("wm.deck_rectangular_profile", text= "Rectangular profile",
icon= 'EVENT_R')

        row = layout.row()

        row = layout.row()

        """Reports"""

        layout.label(text="Generate reports:")

        row = layout.row()

        row.operator("wm.formwork_deck_area_report", text= "Area report", icon=
'WORDWRAP_ON')

        row.operator("wm.concrete_deck_volume_report", text= "Volume report", icon=
'WORDWRAP_ON')

        layout.row()

classes = [

    PT_ArchUtilPanel,

    PT_PilesUtilPanel,

    PT_DeckUtilPanel,

]

def register():

    from bpy.utils import register_class

    for cls in classes:

        register_class(cls)

def unregister():

    from bpy.utils import unregister_class

    for cls in reversed(classes):

        unregister_class(cls)

if __name__ == "__main__":

    register()
```

ANEXO 3. Código del script del puente arco (pilesUtilOperators.py)

pilesUtilOperators.py

```
import bpy, os, csv, math


from bpy.props import StringProperty, BoolProperty

from bpy_extras.io_utils import ImportHelper

from bpy.types import Operator


# Lists to store the reading file data

verts = []

edges = []

faces = []

piles_heights = []

piles_surfaces = []

piles_volumes = []


def clearLists():

    """Cleans lists to avoid errors when importing new data"""

    verts.clear()

    edges.clear()

    faces.clear()

    return {'FINISHED'}


def createPilesAxis(context, filepath):

    """Reads the file and extracts the vertices"""

    with open(filepath, 'r', encoding='utf-8-sig') as csvfile:

        reader = csv.reader(csvfile, delimiter=',')

        for row in reader:

            vert = (float(row[0]), float(row[2]), float(row[1]))
```

```
            verts.append(vert)

        # Joins the vertices

        for i, vert in enumerate(verts):

            startpoint = i

            if i < len(verts)-1:

                endpoint = i+1

            edge = startpoint, endpoint

            edges.append(edge)

    """Creates pile axis"""

    global pile_curve

    # Creates the arch axis mesh

    new_mesh = bpy.data.meshes.new(name='axis_mesh')

    new_mesh.from_pydata(verts, edges, faces)

    ## Creates the object mesh

    new_object = bpy.data.objects.new('pile_axis_curve', new_mesh)

    """ Adds the object mesh to the correspondent collection"""

    global ax_coll

    ax_coll = bpy.data.collections.get('axes_collection') # The collection already
exists

    ax_coll.objects.link(new_object)

    # Selects the object mesh

    pile = bpy.data.objects.get('pile_axis_curve')

    pile.select_set(True)

    # Converts the object mesh to a curve

    bpy.context.view_layer.objects.active                                    =
bpy.context.scene.objects.get('pile_axis_curve')

    bpy.ops.object.convert(target='CURVE')

    # Renames the deck curve

    pile_curve = bpy.data.objects.get('pile_axis_curve')

    pile_curve.name = 'pile_axis'

    pile.select_set(False)

    return {'FINISHED'}
```

```
def createSquareExtrusionProfile(width):

    # Creates the section profile

    bpy.ops.mesh.primitive_plane_add(enter_editmode=False,            align='WORLD',
location=(0, 0, 0), scale=(1, 1, 1))

    # Sets the thickness of the arch

    bpy.context.object.dimensions = (width, width, 0)

    # Converts the object to a curve

    bpy.ops.object.convert(target='CURVE')

    # Adds the section profile to the correspondent collection

    global prof_coll

    prof_coll = bpy.data.collections['profiles_collection']

    square_profile = bpy.context.scene.objects.get('Plano')

    prof_coll.objects.link(square_profile)

    # Remove the section profile from the scene collection

    bpy.context.scene.collection.objects.unlink(square_profile)

    # Renames the section profile

    square_profile.name = 'square_profile'

    bpy.context.object.data.name = 'square_profile_curve'

    # Assigns section profile to pile axis

    for obj in bpy.data.collections['axes_collection'].all_objects:

        if 'pile' in obj.name:

            bpy.context.active_object.select_set(True)

            bpy.context.view_layer.objects.active = obj

            bpy.context.object.data.bevel_mode = 'OBJECT'

            bpy.context.object.data.use_fill_caps = True # To close the mesh

            bpy.context.object.data.bevel_object = square_profile

    # Transforms pile curves into meshes

    for obj in bpy.data.collections['axes_collection'].all_objects:

        if 'pile' in obj.name:

            bpy.context.active_object.select_set(True)
```

```
            bpy.context.view_layer.objects.active = obj
            bpy.ops.object.convert(target='MESH')
    bpy.ops.object.select_all(action='DESELECT')
    # Hides section profile
    for obj in bpy.data.collections['profiles_collection'].all_objects:
        if 'square' in obj.name:
            bpy.context.view_layer.objects.active = obj
            bpy.context.object.hide_set(True)
    bpy.ops.object.select_all(action='DESELECT')
    return {'FINISHED'}


def createRectangularExtrusionProfile(width, length):
    # Para que funcione la función de cálculo de áreas
    global diameter
    diameter = 0
    # Creates the section profile
    bpy.ops.mesh.primitive_plane_add(enter_editmode=False,            align='WORLD',
location=(0, 0, 0), scale=(1, 1, 1))
    # Sets the thickness of the arch
    bpy.context.object.dimensions = (width, length, 0)
    # Converts the object to a curve
    bpy.ops.object.convert(target='CURVE')
    # Adds the section profile to the created collection
    global prof_coll
    prof_coll = bpy.data.collections['profiles_collection']
    rectangular_profile = bpy.context.scene.objects.get('Plano')
    prof_coll.objects.link(rectangular_profile)
    # Remove the section profile from the scene collection
    bpy.context.scene.collection.objects.unlink(rectangular_profile)
    # Renames the section profile
    rectangular_profile.name = 'rectangular_profile'
```

```
    bpy.context.object.data.name = 'rectangular_profile_curve'
    # Assigns section profile to pile axis
    for obj in bpy.data.collections['axes_collection'].all_objects:
        if 'pile' in obj.name:
            bpy.context.active_object.select_set(True)
            bpy.context.view_layer.objects.active = obj
            bpy.context.object.data.bevel_mode = 'OBJECT'
            bpy.context.object.data.use_fill_caps = True # To close the mesh
            bpy.context.object.data.bevel_object = rectangular_profile
    # Transforms pile curves into meshes
    for obj in bpy.data.collections['axes_collection'].all_objects:
        if 'pile' in obj.name:
            bpy.context.active_object.select_set(True)
            bpy.context.view_layer.objects.active = obj
            bpy.ops.object.convert(target='MESH')
    bpy.ops.object.select_all(action='DESELECT')
    # Hides section profile
    for obj in bpy.data.collections['profiles_collection'].all_objects:
        if 'rectangular' in obj.name:
            bpy.context.view_layer.objects.active = obj
            bpy.context.object.hide_set(True)
    bpy.ops.object.select_all(action='DESELECT')
    return {'FINISHED'}


def createCircularExtrusionProfile(diameter):
    # Para que funcione la función de cálculo de áreas
    global width
    width = 0
    global length
    length = 0
    # Creates the section profile
```

```
    bpy.ops.mesh.primitive_circle_add(radius=1, enter_editmode=False, align='WORLD',
location=(0, 0, 0), scale=(1, 1, 1))

    # Sets the thickness of the arch

    bpy.context.object.dimensions = (diameter, diameter, 0)

    # Converts the object to a curve

    bpy.ops.object.convert(target='CURVE')

    # Adds the section profile to the created collection

    global prof_coll

    prof_coll = bpy.data.collections['profiles_collection']

    circular_profile = bpy.context.scene.objects.get('Círculo')

    prof_coll.objects.link(circular_profile)

    # Remove the section profile from the scene collection

    bpy.context.scene.collection.objects.unlink(circular_profile)

    # Renames the section profile

    circular_profile.name = 'circular_profile'

    bpy.context.object.data.name = 'circular_profile_curve'

    for obj in bpy.data.collections['axes_collection'].all_objects:

        if 'pile' in obj.name:

            bpy.context.active_object.select_set(True)

            bpy.context.view_layer.objects.active = obj

            bpy.context.object.data.bevel_mode = 'OBJECT'

            bpy.context.object.data.use_fill_caps = True # To close the mesh

            bpy.context.object.data.bevel_object = circular_profile

    # Transforms pile curves into meshes

    for obj in bpy.data.collections['axes_collection'].all_objects:

        if 'pile' in obj.name:

            bpy.context.active_object.select_set(True)

            bpy.context.view_layer.objects.active = obj

            bpy.ops.object.convert(target='MESH')

    bpy.ops.object.select_all(action='DESELECT')

    # Hides section profile
```

```
    for obj in bpy.data.collections['profiles_collection'].all_objects:
        if 'circular' in obj.name:
            bpy.context.view_layer.objects.active = obj
            bpy.context.object.hide_set(True)
    bpy.ops.object.select_all(action='DESELECT')
    return {'FINISHED'}


def pileHeight():
    # Extracts the length value of the pile
    pile_start = abs(verts[0][-1])
    pile_end = abs(verts[-1][-1])
    global pile_height
    pile_height = round(pile_end - pile_start, 2) # Si el vértice Z inicial de la
péndola es negativo, el código falla
    piles_heights.append(str(pile_height))


def surfaceFormworkPile(width, length, diameter):
    if diameter == 0:
        for i in piles_heights:
            piles_surfaces.append(str(round(((width + length) * float(i)) * 2, 2)))
    else:
        for i in piles_heights:
            piles_surfaces.append(str(round(2 * math.pi * (diameter / 2) * float(i),
2)))
    return {'FINISHED'}


def calculateVolume(width, length, diameter):
    if diameter == 0:
        for i in piles_heights:
            piles_volumes.append(str(round(width * length * float(i), 2)))
    else:
        for i in piles_heights:
```

```
            piles_volumes.append(str(round(math.pi * ((diameter / 2) ** 2) *
float(i), 2)))

    return {'FINISHED'}


class OT_ImportFilebrowser(Operator, ImportHelper):


    bl_idname = "test.import_piles_filebrowser"

    bl_label = "Select file Dialogue Box"

    bl_description = "Allows importing an arch axis coordinate dataset"


    filter_glob: StringProperty(

        default='*.txt;*.csv',

        options={'HIDDEN'}

    )


    def execute(self, context):

        clearLists()

        createPilesAxis(context, self.filepath)

        pileHeight()

        return {'FINISHED'}


class WM_OT_SquareProfile(Operator):


    bl_idname = "wm.piles_square_profile"

    bl_label = "Square Profile Dialoge Box"

    # Asks the value of the dimension in the dialog box

    width_val : bpy.props.FloatProperty(name= "Width (m): ", default= 1)


    def execute(self, context):

        global width

        width = self.width_val # Associates the dimensions with the specified value
```

```
        global length
        length = width # To avoid problems in the calculation of area and volume
        createSquareExtrusionProfile(width)
        return {'FINISHED'}

    def invoke(self, context, event):
        """Invokes the dialog box"""
        return context.window_manager.invoke_props_dialog(self)

class WM_OT_RectangularProfile(Operator):

    bl_idname = "wm.piles_rectangular_profile"
    bl_label = "Rectangular Profile Dialogue Box"
    bl_description = "Extrudes a rectangular profile along the arch axis"

    length_val : bpy.props.FloatProperty(name= "Length (m): ", default= 1)
    width_val : bpy.props.FloatProperty(name= "Width (m): ", default= 1)

    def execute(self, context):
        global length
        length = self.length_val
        global width
        width = self.width_val
        createRectangularExtrusionProfile(width, length)
        return {'FINISHED'}

    def invoke(self, context, event):
        """Invokes the dialog box"""
        return context.window_manager.invoke_props_dialog(self)

class WM_OT_CircularProfile(Operator):
```

```
    bl_idname = "wm.piles_circular_profile"
    bl_label = "Circular Profile Dialoge Box"

    diameter_val : bpy.props.FloatProperty(name= "Diameter (m): ", default= 1)

    def execute(self, context):
        global diameter
        diameter = self.diameter_val
        createCircularExtrusionProfile(diameter)
        return {'FINISHED'}

    def invoke(self, context, event):
        return context.window_manager.invoke_props_dialog(self)

class WM_OT_FormworkAreaReport(Operator):

    bl_idname = "wm.formwork_piles_area_report"
    bl_label = "Formwork Info Area"
    bl_description = "Generates a .txt report with the formwork areas required to
execute the modelled piles"

    def execute(self, context):
        with open('pilesAreaReport.txt', 'a', encoding='utf-8') as areaReportFile:
            surfaceFormworkPile(width, length, diameter)
            areaReportFile.write('------------ FORMWORK PILES AREA REPORT ----------
--' + '\n' + '\n')
            ind = 1
            for i in piles_surfaces:
                areaReportFile.write('- Pile number ' + str(ind) + ': ' + str(i) + '
m²' + '\n')
                ind += 1
```

```
            areaReportFile.write('\n' + '-----------------------------------------
---------' + '\n')

        """Clears lists to avoid errors when recalculating formwork areas"""

        piles_surfaces.clear()

        return {'FINISHED'}


class WM_OT_ConcreteVolumeReport(Operator):


    bl_idname = "wm.concrete_piles_volume_report"

    bl_label = "Concrete Info Volume"

    bl_description = "Generates a .txt report with the piles volumes"


    def execute(self, context):

        with open('pileVolumeReport.txt', 'a', encoding='utf-8') as volumeReportFile:

            calculateVolume(width, length, diameter)

            volumeReportFile.write('------------ CONCRETE PILES VOLUME REPORT ------
------' + '\n' + '\n')

            ind = 1

            for i in piles_volumes:

                volumeReportFile.write('- Pile number ' + str(ind) + ': ' + str(i) +
' m³' + '\n')

                ind += 1

            volumeReportFile.write('\n' + '-----------------------------------------
-------------' + '\n')

            print(piles_volumes)

        """Clears lists to avoid errors when recalculating concrete volumes"""

        piles_heights.clear()

        piles_volumes.clear()

        return {'FINISHED'}


classes = [

    OT_ImportFilebrowser,

    WM_OT_SquareProfile,
```

```
    WM_OT_RectangularProfile,
    WM_OT_CircularProfile,
    WM_OT_FormworkAreaReport,
    WM_OT_ConcreteVolumeReport,
]


def register():
    from bpy.utils import register_class
    for cls in classes:
        register_class(cls)


def unregister():
    from bpy.utils import unregister_class
    for cls in reversed(classes):
        unregister_class(cls)


if __name__ == "__main__":
    register()
```

ANEXO 4. Código del script del puente arco (deckUtilOperators.py)

deckUtilOperators.py

```
import bpy, os, csv, bmesh, addon_utils, math


from bpy.props import StringProperty, BoolProperty

from bpy_extras.io_utils import ImportHelper

from bpy.types import Operator


# Lists to store the reading file data

verts = []

edges = []

faces = []


def clearLists():

    """Cleans lists to avoid errors when importing new data"""

    verts.clear()

    edges.clear()

    faces.clear()

    return {'FINISHED'}


def createDeckAxis(context, filepath):

    """Reads the file and extracts the vertices"""

    with open(filepath, 'r', encoding='utf-8-sig') as csvfile:

        reader = csv.reader(csvfile, delimiter=',')

        for row in reader:

            vert = (float(row[0]), float(row[2]), float(row[1]))

            verts.append(vert)

        # Joins the vertices

        for i, vert in enumerate(verts):
```

```
            startpoint = i
            if i < len(verts)-1:
                endpoint = i+1
            edge = startpoint, endpoint
            edges.append(edge)
    """Creates deck axis"""
    global deck_curve
    # Creates the arch axis mesh
    new_mesh = bpy.data.meshes.new(name='axis_mesh')
    new_mesh.from_pydata(verts, edges, faces)
    ## Creates the object mesh
    new_object = bpy.data.objects.new('deck_axis_curve', new_mesh)
    """ Adds the object mesh to the correspondent collection"""
    global ax_coll
    ax_coll = bpy.data.collections.get('axes_collection') # The collection already
exists
    ax_coll.objects.link(new_object)
    # Selects the object mesh
    deck = bpy.data.objects.get('deck_axis_curve')
    deck.select_set(True)
    # Converts the object mesh to a curve
    bpy.context.view_layer.objects.active                                    =
bpy.context.scene.objects.get('deck_axis_curve')
    bpy.ops.object.convert(target='CURVE')
    # Renames the deck curve
    deck_curve = bpy.data.objects.get('deck_axis_curve')
    deck_curve.name = 'deck_axis'
    deck.select_set(False)
    return {'FINISHED'}


def enable3dPrintAddon():
    addon_name = 'object_print3d_utils'
```

```
    isenabled = addon_utils.check(addon_name)[0]

    if not isenabled:

        addon_utils.enable('object_print3d_utils')

    return {'FINISHED'}


def makeManifold():

    # Selects arch axis

    bpy.data.objects.get('deck_axis').select_set(True)

    bpy.context.view_layer.objects.active = bpy.data.objects['deck_axis']

    # Makes mesh manifold

    bpy.ops.mesh.print3d_clean_non_manifold()

    bpy.ops.object.select_all(action='DESELECT')

    return {'FINISHED'}


def createRectangularExtrusionProfile(width, height):

    # Creates the section profile

    bpy.ops.mesh.primitive_plane_add(enter_editmode=False,            align='WORLD',
location=(0, 0, 0), scale=(1, 1, 1))

    # Sets the thickness of the arch

    bpy.context.object.dimensions = (width, height, 0)

    # Converts the object to a curve

    bpy.ops.object.convert(target='CURVE')

    # Adds the section profile to the created collection

    global prof_coll

    prof_coll = bpy.data.collections['profiles_collection']

    rectangular_profile = bpy.context.scene.objects.get('Plano')

    prof_coll.objects.link(rectangular_profile)

    # Remove the section profile from the scene collection

    bpy.context.scene.collection.objects.unlink(rectangular_profile)

    # Renames the section profile

    rectangular_profile.name = 'rectangular_profile'
```

```
    bpy.context.object.data.name = 'rectangular_profile_curve'
    # Selects deck axis
    bpy.context.active_object.select_set(False)
    bpy.data.objects.get('deck_axis').select_set(True)
    bpy.context.view_layer.objects.active = bpy.data.objects['deck_axis']
    # Assigns section profile to deck axis
    bpy.context.object.data.bevel_mode = 'OBJECT'
    bpy.context.object.data.use_fill_caps = True # To close the mesh
    bpy.context.object.data.bevel_object = rectangular_profile
    # Transforms extruded profile along curve to mesh
    bpy.ops.object.convert(target='MESH')
    # Hides section profile
    for obj in bpy.data.collections['profiles_collection'].all_objects:
        if 'rectangular' in obj.name:
            bpy.context.view_layer.objects.active = obj
            bpy.context.object.hide_set(True)
    bpy.ops.object.select_all(action='DESELECT')
    return {'FINISHED'}


def surfaceFormworkDeck(width, height):
    ## Lower surface
    # Extracts the length value of the deck
    deck_start = abs(verts[0][0])
    deck_end = abs(verts[-1][0])
    deck_length = deck_start + deck_end
    # Calculates the lower deck surface
    global low_deck_surf
    low_deck_surf = round(deck_length * width, 2)
    ## Lateral surface
    global lat_deck_surf
    lat_deck_surf = round(((deck_length + width) * height) * 2, 2)
```

```
    return {'FINISHED'}


def calculateVolume():
    bm = bmesh.new()
    bm.from_mesh(bpy.context.object.data)
    global deck_vol
    deck_vol = round(float(bm.calc_volume()), 2)
    return {'FINISHED'}


class OT_ImportFilebrowser(Operator, ImportHelper):


    bl_idname = "test.import_deck_filebrowser"
    bl_label = "Select file Dialogue Box"
    bl_description = "Allows importing an arch axis coordinate dataset"


    filter_glob: StringProperty(
        default='*.txt;*.csv',
        options={'HIDDEN'}
    )


    def execute(self, context):
        clearLists()
        createDeckAxis(context, self.filepath)
        return {'FINISHED'}


class WM_OT_RectangularProfile(Operator):


    bl_idname = "wm.deck_rectangular_profile"
    bl_label = "Rectangular Profile Dialogue Box"
    bl_description = "Extrudes a rectangular profile along the arch axis"
```

```
    width_val : bpy.props.FloatProperty(name= "Width (m): ", default= 1)

    height_val : bpy.props.FloatProperty(name= "Height (m): ", default= 1)



    def execute(self, context):

        global width

        width = self.width_val

        global height

        height = self.height_val

        createRectangularExtrusionProfile(width, height)

        makeManifold()

        return {'FINISHED'}



    def invoke(self, context, event):

        """Invokes the dialog box"""

        return context.window_manager.invoke_props_dialog(self)



class WM_OT_FormworkAreaReport(Operator):



    bl_idname = "wm.formwork_deck_area_report"

    bl_label = "Formwork Info Area"

    bl_description = "Generates a .txt report with the formwork areas required to
execute the modelled deck"



    def execute(self, context):

        surfaceFormworkDeck(width, height)

        with open('deckAreaReport.txt', 'w', encoding='utf-8') as areaReportFile:

            areaReportFile.write('------------- FORMWORK DECK AREA REPORT ----------
---' + '\n' + '\n')

            areaReportFile.write('- Lower deck formwork surface = ' +
str(low_deck_surf) + ' m²' + '\n')

            areaReportFile.write('- Lateral deck formwork surface = ' +
str(lat_deck_surf) + ' m²' + '\n'+ '\n')
```

```
            areaReportFile.write('-----------------------------------------------
----' + '\n')

        return {'FINISHED'}



class WM_OT_ConcreteVolumeReport(Operator):



    bl_idname = "wm.concrete_deck_volume_report"

    bl_label = "Concrete Info Volume"

    bl_description = "Generates a .txt report with the arch volumes"



    def execute(self, context):

        calculateVolume()

        with open('deckVolumeReport.txt', 'w', encoding='utf-8') as volumeReportFile:

            volumeReportFile.write('------------ CONCRETE DECK VOLUME REPORT -------
-----' + '\n' + '\n')

            volumeReportFile.write('- Total volume = ' + str(deck_vol) + ' m³' + '\n'
+ '\n')

            volumeReportFile.write('-----------------------------------------------
------' + '\n')

        return {'FINISHED'}



classes = [

    OT_ImportFilebrowser,

    WM_OT_RectangularProfile,

    WM_OT_FormworkAreaReport,

    WM_OT_ConcreteVolumeReport,

]



def register():

    from bpy.utils import register_class

    for cls in classes:

        register_class(cls)
```

```
def unregister():

    from bpy.utils import unregister_class

    for cls in reversed(classes):

        unregister_class(cls)


if __name__ == "__main__":

    register()
```