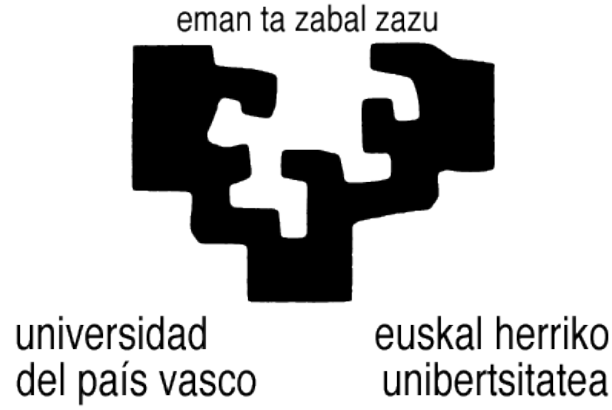

Facultad de Informática / Informatika Fakultatea

Herramienta de creación de modelos STL parametrizables mediante Processing

Alumno/a: D./Dña. Mikel Del Valle Iriondo

Director/a: D./Dña. Abelardo Gil Fournier

Director/a: D./Dña. Carmen Hernández Gómez

Proyecto Fin de Carrera, julio 2014

RESUMEN

Este Proyecto de Fin de Carrera ha sido realizado en colaboración con la empresa Ultra-Lab y el centro Arteleku y trata sobre el diseño, desarrollo e implementación de una interfaz visual para la creación, manipulación, transformación y visualización de diferentes figuras en un entorno 3D. Estas herramientas visuales dan soporte al usuario para que este pueda realizar múltiples modificaciones de las figuras con las que está trabajando de una forma fácil, intuitiva y eficiente.

Índice de contenido

Índice de figuras

Índice de tablas

Índice de algoritmos

Capítulo 1 INTRODUCCIÓN

En este documento se describe el proyecto Fin de Carrera realizado en colaboración con la empresa Ultra-Lab bajo la supervisión de Abelardo Gil y también de Arteleku junto con Daniel Artamendi y dirigido por la profesora de la UPV/EHU Carmen Hernández. El proyecto consiste en el diseño e implementación de una herramienta que facilita la tarea de creación y manipulación de polígonos.

La memoria tiene la estructura que se detalla a continuación:

En este capítulo, se muestra el Documento de Objetivos del Proyecto el cual incluye la planificación de las fases del proyecto. También se presentan los aspectos relacionados con la gestión de riesgos.

En el segundo capítulo, se presenta la herramienta de trabajo *Processing*, trabajos notables realizados con este entorno de programación y las distintas librerías utilizadas en el desarrollo de este proyecto.

En el tercer capítulo, se incluye el diseño y la implementación realizada a fin de obtener la aplicación de generación de figuras.

En el cuarto capítulo, se presentan las conclusiones obtenidas de la realización de este proyecto, además de las futuras líneas de trabajo

Finalmente, se incluye un listado de la bibliografía que se ha utilizado durante la realización de este proyecto y de la memoria.

1.1 *Documento de Objetivos del Proyecto*

En este apartado se expone la motivación personal y de la empresa en cuanto a la realización del proyecto, así como los objetivos del mismo para fijar su alcance.

También se incluyen las cuestiones relacionadas con la organización del proyecto. Y se toman en consideración los problemas que podrían afectar a la realización de este PFC.

1.1.1 *Motivación*

La idea de este proyecto surgió durante la realización de un taller en Arteleku: Tresnak::Tools {07 - Fabricación Digital: Repetir. Transformar. Parametrizar.

Una de las tareas que se trabajó en este taller fue la manipulación de funciones informáticas enfocadas a la tarea de creación de objetos. Para una gran parte de los asistentes al taller, la modificación del código fue una tarea complicada ya que no poseían los conocimientos informáticos necesarios para realizar cambios en el código proporcionado. Por ello, los responsables del taller pensaron que el desarrollo de una herramienta sencilla e intuitiva podía ser una posible ayuda para usuarios con reducidos conocimientos informáticos.

De esta forma surgió la idea de realizar la herramienta que se ha desarrollado en este proyecto. Personalmente, me pareció un proyecto interesante ya que me acercaba de cierta manera a la utilización de impresoras 3D.

1.1.2 *Objetivos*

El objetivo principal que se plantea en este proyecto, es la creación de una herramienta con la que, personas sin conocimiento en lenguajes de programación, puedan realizar distintas figuras para posteriormente imprimirlas en una impresora 3D.

Para poder alcanzar el objetivo principal se plantearon algunos objetivos secundarios:

- Obtener destreza y soltura con el lenguaje de programación *Processing,* el cual era una novedad para el alumno.
- Estudio de las diversas librerías utilizadas en la creación de la herramienta.
- Análisis, implementación y mejora del código facilitado por la empresa Ultra-Lab.

1.2 *Herramientas utilizadas*

Para realizar este proyecto se ha utilizado un equipo de desarrollo constituido por :

- Sistema operativo Windows 7.
- Librerías de *Processing*, *ControlP5* [6], *peasycam* [4], *toxiclibs* [8].
- *Netfabb Studio*.

Para la realización de la memoria se han usado las siguientes aplicaciones:

- *OpenOffice*.
- *GanttProject*.

1.3 *Fases del proyecto*

En este apartado realizaremos una descripción de las fases en las que se divide el proyecto.

- Gestión y planificación inicial.
- Formación.
- Desarrollo de la aplicación.
- Edición de la memoria.
- Preparación de la presentación.

Debemos destacar que se han realizado diferentes reuniones con los codirectores del proyecto mediante la herramienta de videoconferencia "*google hangout"* durante todo el periodo de desarrollo del proyecto.

Las fases que componen el proyecto son las siguientes:

1. Gestión y planificación inicial: fase en la que se determina el alcance del proyecto, se estudian los objetivos a alcanzar y se realiza una correcta planificación.
2. Formación: fase en la que se estudia la tecnología necesaria para la realización del proyecto incluyendo el código entregado por parte de la empresa.
3. Desarrollo: fase en la que se diseña e implementa la aplicación para realizar

figuras.

4. Memoria: fase en la que se recopila toda la información del proyecto y se procede a su redacción.

5. Presentación: fase en la que se prepara la presentación a realizar para la defensa del proyecto.

1.4 *Estimación de tiempo*

El proyecto comenzó a mediados de febrero de 2013 y se desarrolló hasta mediados de julio. Aproximadamente se han dedicado 4 horas diarias a la realización del proyecto.

Hay que señalar el periodo vacacional de Semana Santa (del 28 de marzo al 5 de abril de 2013).

Debemos decir que la presentación del proyecto sufrió un retraso ya que este no pudo ser presentado en la convocatoria deseada, y debido a diversos contratiempos al final se decidió presentarlo en esta convocatoria. Es por ello que gran parte del trabajo realizado se encuadra en el año anterior y, solamente, algunos detalles de la memoria han sido realizados en estos últimos días.

En las siguientes páginas presentamos una descripción más detallada del trabajo realizado dentro de cada fase, acompañada del diagrama de Gantt correspondiente.

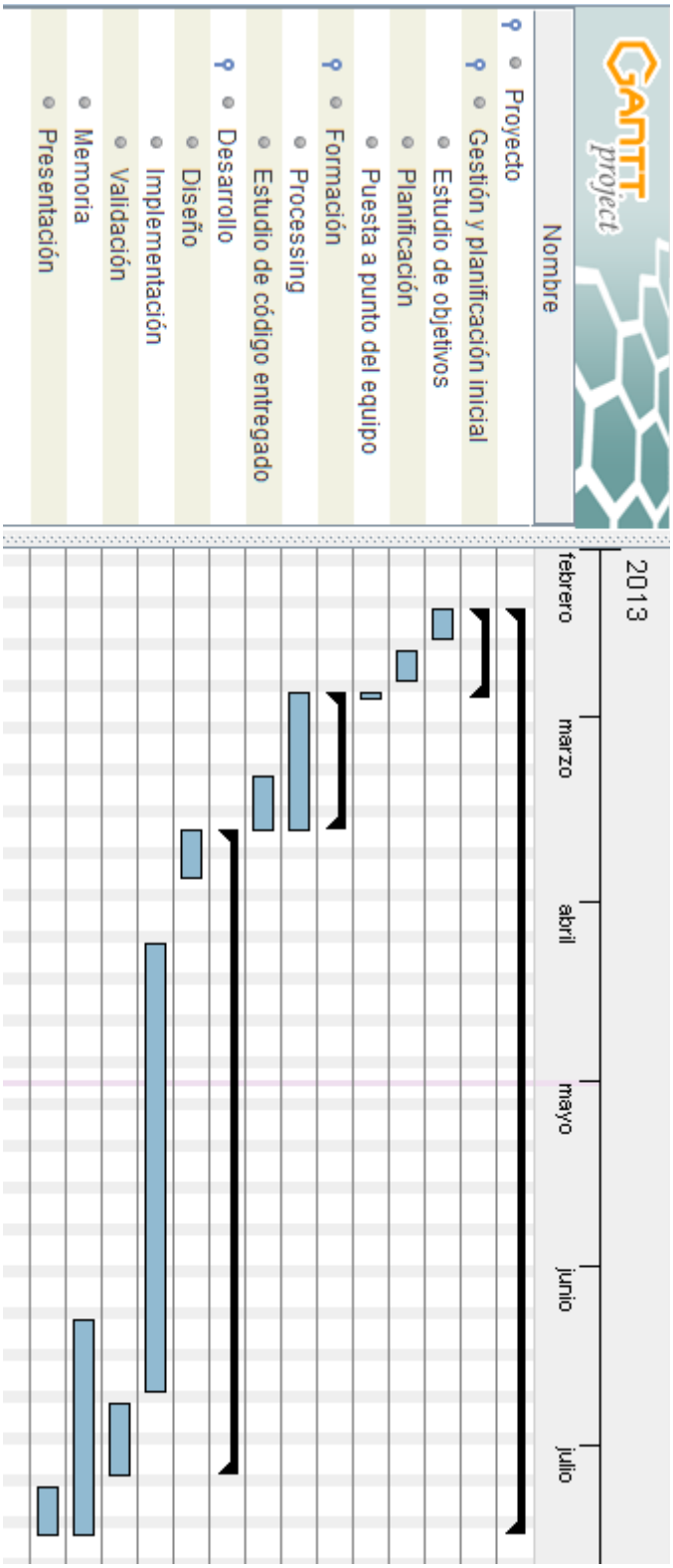

Figura 1: Diagrama Gantt del proyecto

1.4.1 *Fase 1 Gestión y Planificación inicial.*

En esta fase se estudia y planifica la realización del proyecto. Se establecen los objetivos y el alcance.

- Estudio de objetivos: Se establecen los objetivos que se quieren lograr con la realización del proyecto

- Planificación: Organización global de las fases del proyecto y la estimación del tiempo dedicado a cada una de ellas

- Puesta a punto del equipo: Instalación del software a utilizar.

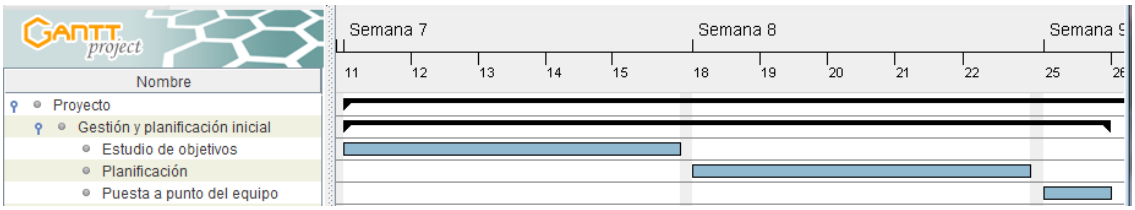

Figura 2: Diagrama Gantt: Gestión y planificación inicial

1.4.2 *Fase 2 Formación.*

Esta fase consiste en el estudio de las tecnologías y herramientas que se utilizan en el proyecto. Esto es, la lectura de documentación y realización de tutoriales para familiarizarse con el entorno de desarrollo.

Principalmente, se ha estudiado el lenguaje *Processing* y el modo en el que se pueden desarrollar las aplicaciones para el proyecto mediante la creación de ejemplos básicos.

Una vez que los conceptos ya quedaron claros se procedió al estudio del código entregado.

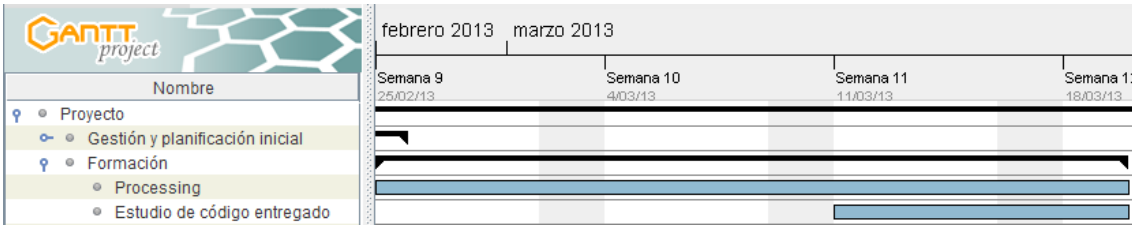

Figura 3: Diagrama Gantt: Formación

1.4.3 *Fase 3 Desarrollo.*

Esta fase consiste en el diseño, implementación y prueba del código programado.

- Diseño: diseño de la arquitectura y de los módulos necesarios para la puesta en funcionamiento de la aplicación.
- Implementación: una vez realizado un correcto diseño de la aplicación, se procede a desarrollar el código para obtener la aplicación completa.
- Validación: se prueba el correcto funcionamiento de la aplicación mediante distintas pruebas.

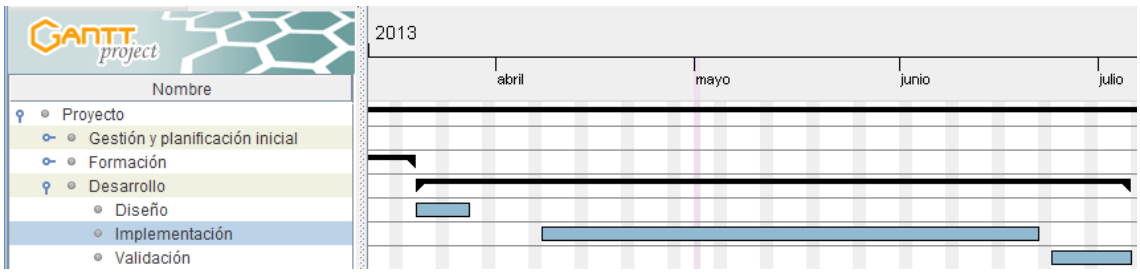

Figura 4: Diagrama Gantt: Desarrollo

1.4.4 *Fase 4 y 5 Memoria y Presentación.*

En esta fase se realiza la recopilación de toda la información obtenida, utilizada y generada así como la elaboración del documento de la memoria del proyecto.

Esto conlleva también la creación de los archivos y la preparación necesaria para la

defensa del proyecto.

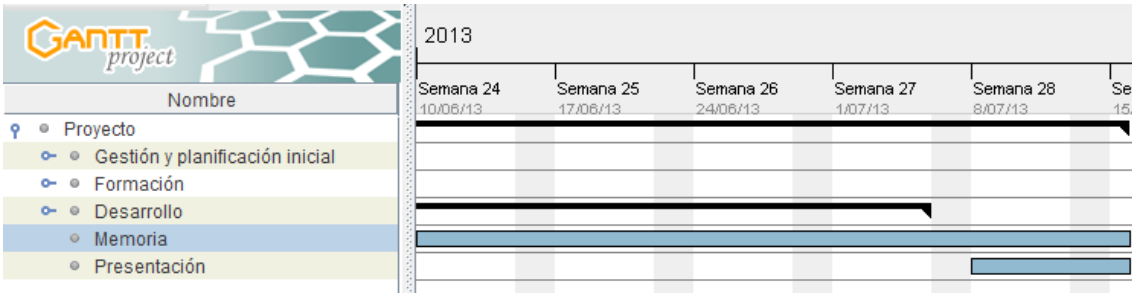

Figura 5: Diagrama Gantt: Memoria y Presentación

1.5 *Análisis de Riesgos*

A continuación, se analizarán los riesgos identificados, la probabilidad de que sucedan y cómo estos podrían afectar al proyecto. Así mismo, se describirán las medidas que se han adoptado para evitarlos y el plan de contingencia en caso de que sucediesen.

1.5.1 *Riesgos*

- **Indisponibilidad de alguna de las partes:** la ausencia del alumno o de alguno de los tutores tanto de Ultra-Lab como de Arteleku debido a enfermedades u otros motivos puede ocasionar graves retrasos o incluso la cancelación del proyecto.

- **Pérdida de la información:** La pérdida de datos o cualquier otro tipo de información debida a diferentes motivos, como averías en el equipo o borrados accidentales, pueden ocasionar graves retrasos.

- **Problemas en el desarrollo:** falta de experiencia por parte del alumno en la utilización de alguna de las herramientas y tecnologías aplicadas en el proyecto.

- **Estimaciones erróneas:** la inexperiencia en la planificación y en la gestión de proyectos puede provocar una estimación de incorrecta que afecte a la duración prevista del proyecto.

1.5.2 *Cuantificación de riesgos*

En la siguiente tabla (Tabla 1) tenemos una enumeración de los riesgos junto a una estimación de la probabilidad de que estos ocurran. También se valora el impacto que puedan tener en la realización del proyecto en caso de que alguno de ellos sucediera.

Riesgo	**Probabilidad**	**Impacto**	**Valoración**
Indisponibilidad de alguna de las partes	Baja	Alto	Medio
Pérdida de información	Baja	Alto	Alto
Problemas en el desarrollo	Alta	Medio	Medio
Estimación errónea	Media	Alta	Alta

Tabla 1: Tabla de riesgos

1.5.3 *Plan de contingencia y actuación.*

Indisponibilidad de alguna de las partes:

- Si el alumno sufre una indisponibilidad moderada, este deberá recuperar las horas perdidas.
- En caso de que la indisponibilidad se alargara demasiado en el tiempo habría que revisar la planificación de la fecha de entrega.
- Si la indisponibilidad es causada por alguno de los tutores durante un largo periodo de tiempo, se procederá a buscar un sustituto.

Pérdida de información: Para tratar de evitar esta situación se llevará a cabo una política de copias de seguridad consistente en:

- Realizar copias diarias de la aplicación y de la memoria.

- Dichas copias se almacenarán en el PC de trabajo, en un disco duro externo y en el correo personal de Google.

Problemas en el desarrollo: Para evitar este riesgo se ha tratado de reunir previamente la mayor información posible a fin de cumplir los objetivos marcados en este proyecto. Si esta información no fuera suficiente durante el desarrollo del proyecto, se procederá a buscar la información necesaria durante la fase de desarrollo o a tomar en cuenta otras alternativas.

Estimación errónea: Si se detecta esta situación, se volverá a planificar la fase en la que se haya producido y las posteriores fases del proyecto.

Capítulo 2 PROCESSING

2 *¿Qué es Processing?*

Processing en un lenguaje de programación, un entorno de desarrollo y una comunidad online desde 2001, que fusiona y genera arte visual y software. Inicialmente fue creado como un cuaderno de bocetos para enseñar los fundamentos de la programación en un entorno visual. No obstante *Processing* se desarrolló rápidamente como una herramienta para crear trabajos profesionales.

Processing fue creado por Ben Fry y Casey Reas en 2001 [3] mientras ambos eran estudiantes de John Maeda en el Laboratorio MIT *Media Lab*. No obstante, el mayor desarrollo ha tenido lugar en el *Interaction Design Institute Ivrea*, en la Universidad *Carnegie Mellon* y en la Universidad UCLA, donde Reas es director del Departamento de Diseño *Media Arts*.

Por su trabajo en Processing, Fry y Reas recibieron en el año 2008 el premio Muriel *Cooper* del Instituto *Design Management*. La comunidad de *Processing* fue premiada en el año 2005 con el premio *Prix Ars Electronica Golden Nica* y con el premio *Interactive Desing Prize* del *Tokyo Type Director's Club*.

Processing es una alternativa libre, de código abierto, a las herramientas de software con costosas licencias, siendo accesible en distintos entornos educativos. Su condición de código abierto fomenta la participación de la comunidad y la colaboración la cual es vital para el crecimiento de *Processing*.

La comunidad comparte programas, contribuye con código, resuelve dudas en el foro y crea librerías para mejorar el software. La comunidad de *Processing* ha escrito más de setenta librerías para facilitar la visión por computador, la visualización de datos, música, redes y electrónica.

Los estudiantes de cientos de escuelas de todo el mundo usan *Processing* para las clases que van desde la educación matemática en la escuela media a los grados de programación en Estudios de Bellas Artes.

- En el programa ITP de la Universidad de Nueva York, se enseña *Processing* junto a Arduino y PHP como parte del curso básico para 100 alumnos nuevos cada año.

- En la Universidad de California (Los Angeles), los estudiantes del programa *Design Media Arts* utilizan *Processing* para aprender los conceptos y las habilidades necesarias para la creación de la próxima generación de sitios web y vídeo juegos.

- En las Escuelas Públicas de Lincoln en Nebraska y en la Escuela *Phoenix Country Day* en Arizona, los maestros de escuela están experimentando con *Processing* para complementar el álgebra tradicional y las clases de geometría.

Decenas de miles de empresas, artistas, diseñadores, arquitectos e investigadores usan *Processing* para crear una gama muy diversa de proyectos.

- Empresas de diseño como *Motion Theory* proporciona gráficos en movimiento creados con *Processing* para los anuncios de televisión de empresas como *Nike*, *Budweiser* y *Hewlett-Packard*.

- Bandas como REM, *Radiohead*, y *Modest Mouse* han ofrecido animación creada con *Processing* en sus vídeos musicales.

- Publicaciones como la revista *Nature*, *New York Times*, *Seed*, y *Communications of ACM* han encargado gráficos informativos creados con *Processing*.

- El grupo de artistas HeHe ha utilizado *Processing* para producir su galardonada instalación *Nuage Vert*, una visualización pública a gran escala de los niveles de contaminación en Helsinki.

- El Laboratorio de Física Aplicada de la Universidad de Washington utiliza *Processing* para crear la visualización de un ecosistema marino costero como parte del proyecto RISE NSF.

- El Instituto *Armstrong Interactive Media Studies* en la Universidad de Miami utilizan *Processing* para la construcción de herramientas de visualización y análisis de texto digitales para la investigación en Humanidades.

Finalmente, queremos destacar que el museo *Cooper-Hewitt National Design* incluyó *Processing* en su *National Design Triennial*. Obras creadas con *Processing* ocuparon un lugar destacado en el espectáculo *Desing* y *Elastic Mind* en el Museo de Arte Moderno.

Asimismo, muchas revistas de diseño, incluyendo *Print, Eye*, y *Creativity* han incluido y destacado reseñas del software en sus publicaciones.

2.1 *Trabajos notables con Processing.*

A continuación, mostraremos una lista de trabajos notables realizados por distintos artistas y profesionales.

2.1.1 *Signals – Casey Reas*

Signals fue un mural encargado para el edificio 76 del Instituto Tecnológico de Massachusetts (MIT). Este fue creado en colaboración con Ben Fry.

En concreto, este mural presenta señales entre las proteínas de un tumor canceroso. El comportamiento celular en un tumor parece estar controlado por proteínas interconectadas que operan en una red para transmitir instrucciones de forma activa. Estas redes se vuelven disfuncionales en las células cancerosas. Como puede observarse en la imagen (Figura 6) cada cluster representa las señales entre las proteínas interconectadas en un cáncer a medida que cambian con el tiempo. Los arcos individuales son señales de una proteína a otra, y el tamaño del arco se corresponde con la magnitud de la señal. Los datos fueron proporcionados por el laboratorio del profesor Michael Yaffe.

Figura 6: Signals – Casey Reas

2.1.2 *Tom Carden - Travel time tube map*

Tom Carden ha construido un mapa del metro de Londres que muestra el tiempo que te llevaría viajar de una estación a otra.

Para utilizarlo, en el sitio web de Carden [7] en el cual podemos seleccionar una estación en el menú desplegable o haciendo *click* en una estación del mapa. La topografía se retuerce para reflejar el tiempo que tomaría llegar a un destino determinado, el cual está marcado por círculos concéntricos alrededor de la estación elegida.

Figura 7: Tube map - Tom Carden

2.1.3 *Marius Watz - GasWorks 1-3 (2004)*

Marius Watz [10] es un artista que trabaja con la abstracción visual a través de procesos de software generativo. Su trabajo se centra tanto en la síntesis de la forma como en el producto de comportamientos paramétricos. Es conocido por sus geometrías de bordes irregulares y colores vivos, y ha desarrollado una serie de trabajos que van desde obras de software, proyecciones públicas hasta la creación de objetos físicos producidos con la tecnología de fabricación digital.

Watz ha expuesto en lugares como el museo *Victoria & Albert Museum* (Londres), *Todaysart* (La Haya), *Itaú Cultural* (Sao Paulo), *Museumsquartier* (Viena), y *Galleri ROM* (Oslo).

Entre sus trabajos podemos destacar la obra titulada *GasWorks 1-3* (2004) que son tres animaciones computacionales basadas en asociaciones. Los tres diferentes sistemas muestran los principios dinámicos del movimiento generativo y su composición.

Esta obra fue creada para ser expuesta en la Galería *Gas Station* de *Atmosferas* en Lisboa (Portugal). *Atmosferas* es un laboratorio de cultura digital que organiza eventos, conferencias y exposiciones para promover un diálogo entre las artes y las ciencias.

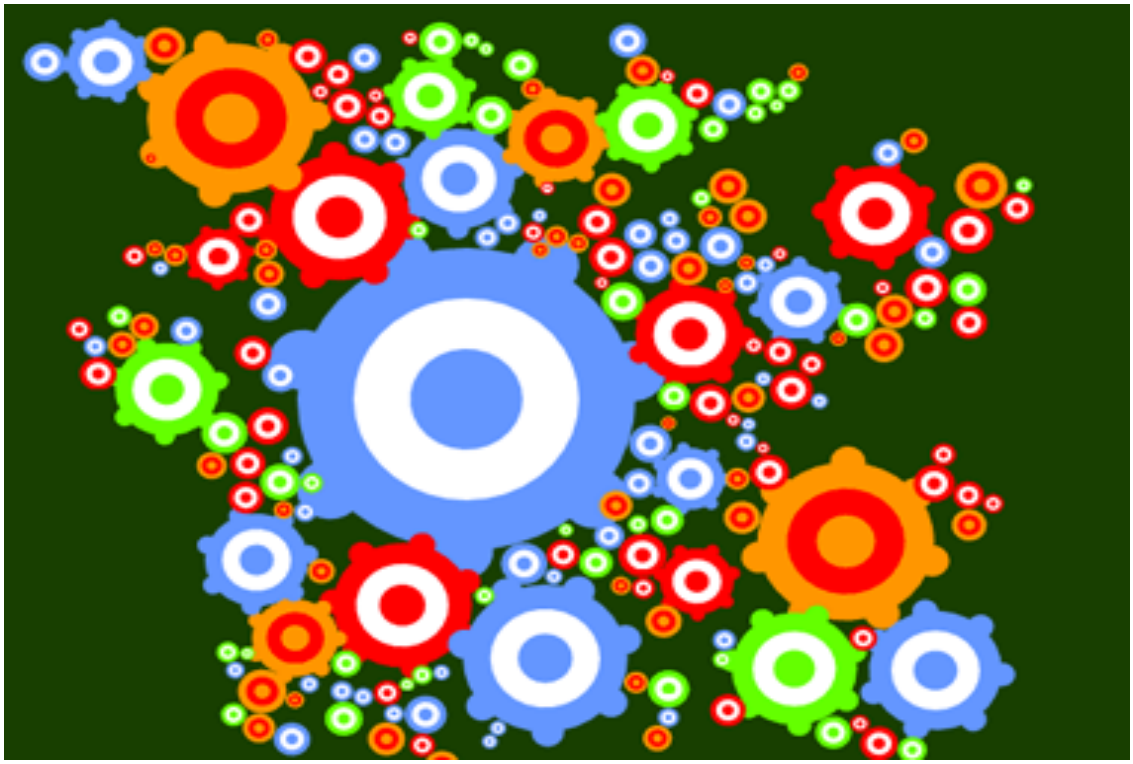

Figura 8: Gas Work - Marius Watz

2.1.4 *Jer Thorp - Just Landed*

Jer Thorp [9] es un artista y profesor de Vancouver, Canadá. Viniendo de unos antecedentes en genética, sus prácticas de arte digital exploran los límites entre ciencia, datos, arte y cultura. Recientemente su trabajo ha sido publicado en "*The Guardian*", "*Scientific American*", "*The New Yorker*", y "*Popular Science*".

El galardonado trabajo en software de Thorp se ha expuesto en Europa, Asia, América del Norte y del Sur, incluso en el Museo de Arte Moderno de Manhattan.

Figura 9: Just Landed - Jer Thorp

2.2 *Librerías de Processing utilizadas en el proyecto*

En este apartado hablaremos de las distintas librerías utilizadas durante la realización de la aplicación.

2.2.1 *ControlP5*

ControlP5 es una librería escrita por Andreas Schlegel, para el entorno de programación de *Processing*.

ControlP5 es una interfaz gráfica de usuario para *Processing*. Puede ser usado en aplicaciones y en modo *applet*. Cuenta con controladores botones, *toggles*, paneles desplazables y casillas de verificación entre otros, que pueden ser fácilmente añadidos a *Processing*.

ControlP5 ofrece una gama de controladores que permiten cambiar fácilmente y ajustar

los valores, mientras que el programa está en ejecución. Cada controlador se identifica por un nombre único asignado al crear el controlador. *ControlP5* localiza variables y funciones dentro del dibujo y une los controladores a las variables o funciones coincidentes automáticamente. Cambios realizados en el controlador pueden ser fácilmente capturados dentro del dibujo mediante la implementación de la función *ControlEvent*.

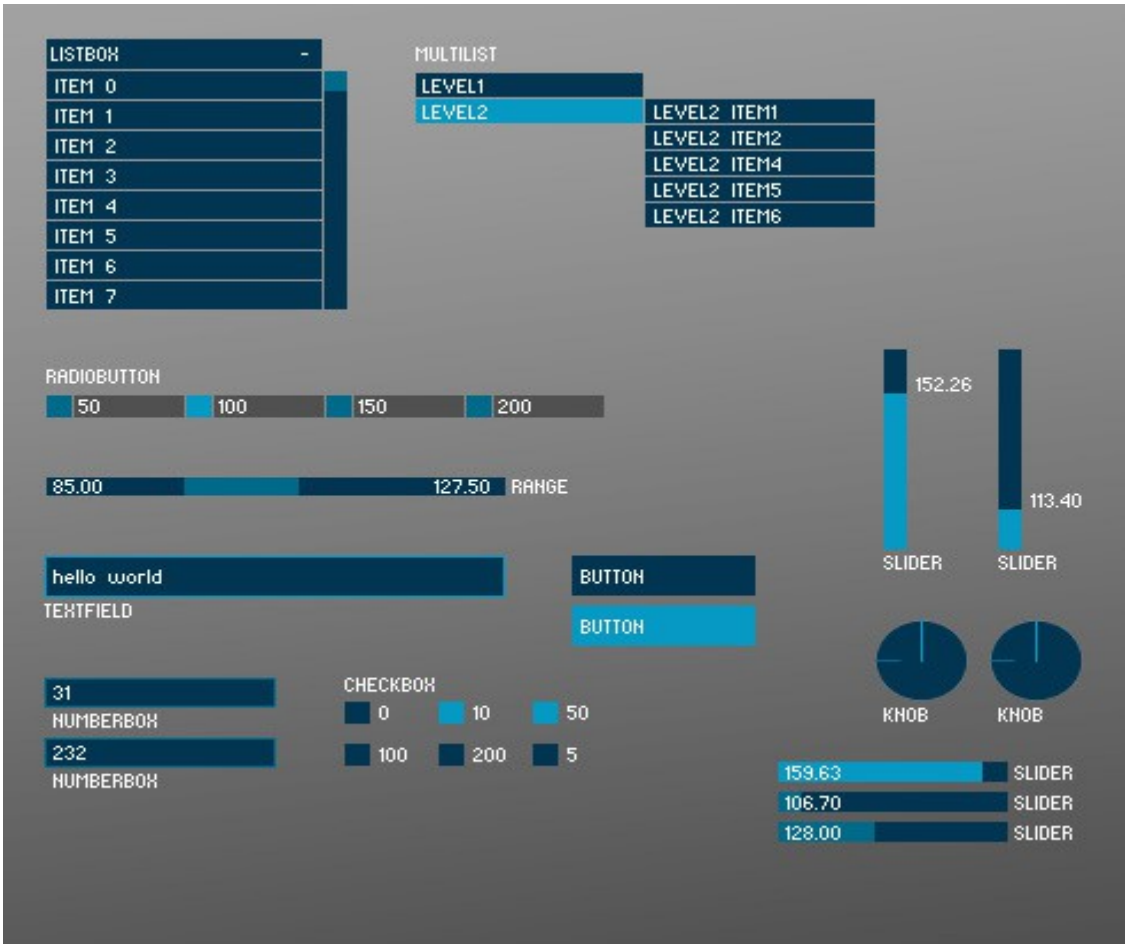

Figura 10: ControlP5

2.2.2 *Toxiclibs*

Toxiclibs [8] es una librería, de código abierto e independiente, que puede ser utilizada para tareas de diseño computacional con Java y *Processing*, desarrollada por Karsten

Schmidt. Las clases son genéricas casi en su totalidad para maximizar su reutilización en diferentes contextos, que van desde el diseño generativo, animación, diseño de interfaz, visualización de datos hasta la arquitectura y la fabricación digital. También se utiliza como herramienta de enseñanza.

Después de más de 3 años de desarrollo la colección se compone de más de 270 clases, 18 paquetes agrupados en 7 librerías y más de 25000 líneas de código.

Las librerías se han diseñado para no tener dependencias a fin de maximizar la reutilización y la flexibilidad de las mismas. A pesar de que estas bibliotecas se han desarrollado principalmente para su uso en proyectos relacionados con *Processing*, no hay dependencia explícita sobre el *PApplet* o cualquier otra clase de *Processing*. Esta característica es totalmente intencional a fin de maximizar el uso de estas librerías.

Además, debemos destacar que todo el código de este proyecto utiliza la sintaxis de Java 5 y, en breve, sólo Java 6 por lo que no funcionará con versiones anteriores de *Processing*.

Capítulo 3: DISEÑO E IMPLEMENTACIÓN

3 *Descripción del programa*

En este capítulo presentamos el diseño general de la aplicación, la descripción de las funciones correspondientes a los aspectos más importantes de la aplicación así como las imágenes de la interfaz de usuario.

3.1 *Descripción de las funciones*

El diseño del programa consta de varias partes:

- Diseño de la Pantalla Inicial
- Gestión de eventos
- Inicialización de la aplicación
- Visualización de Figuras

3.1.1 *Diseño de la Pantalla Inicial*

En este bloque se definirán la imagen inicial de la pantalla, los botones y *sliders* más importantes de la aplicación junto con una breve explicación de cada uno.

Comenzaremos con el diseño de la pantalla inicial tal cual aparece al iniciar la aplicación y, posteriormente, continuaremos con cada uno de los botones.

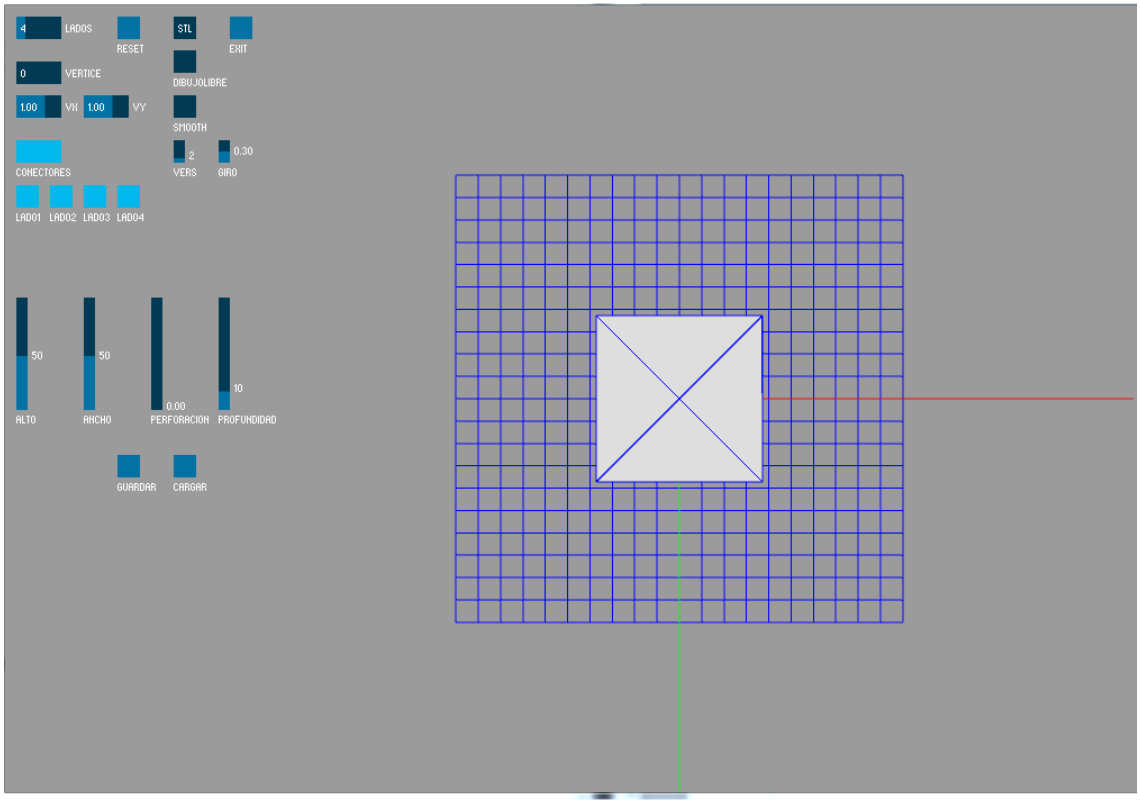

Figura 11: Pantalla inicial

Sin contar con los botones *sliders* y las casillas, podemos ver que se carga una re**j**illa **de** tamaño similar al de distintas superficies de impresión de impresoras 3D, ya que las piezas creadas con la aplicación tienen la opción de ser imprimidas.

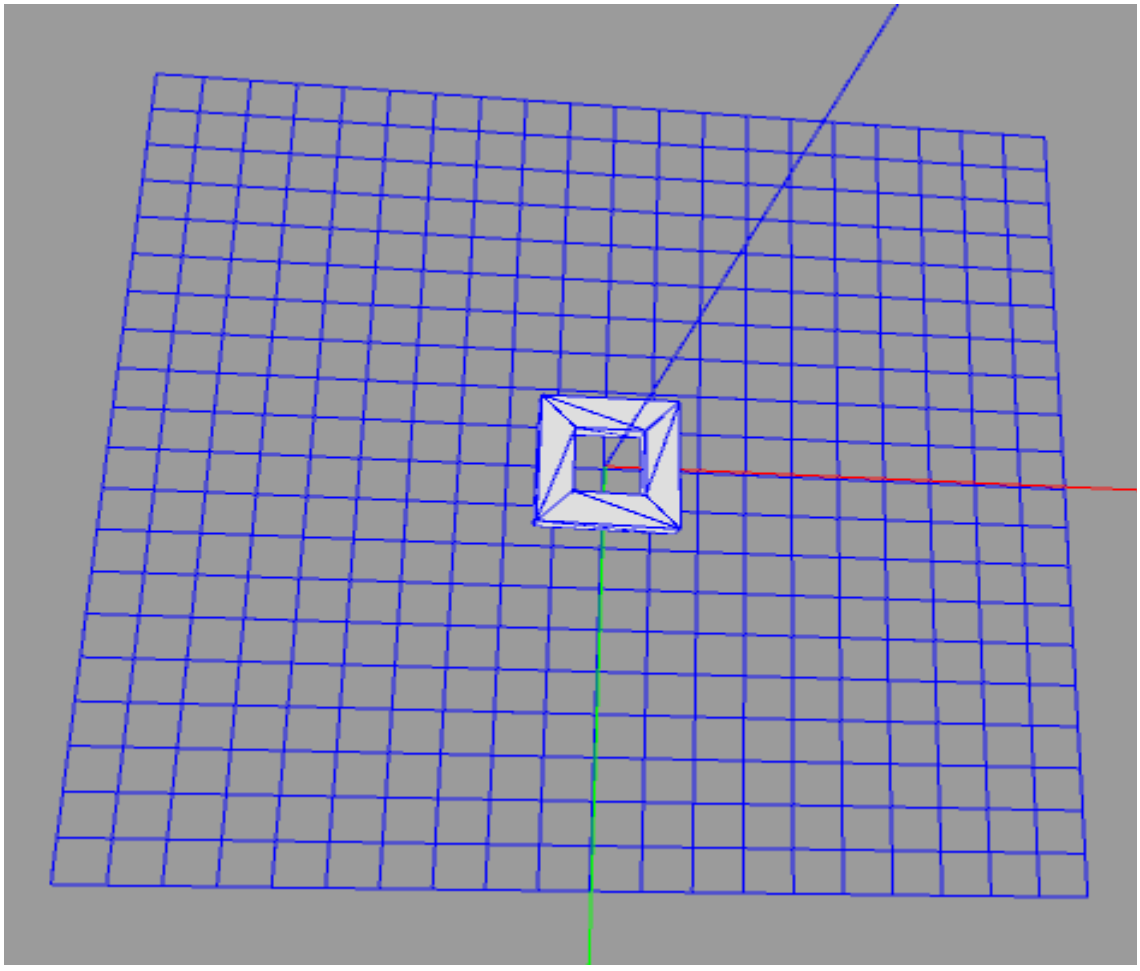

Figura 12: Rejilla

Botones, *Sliders* y Casillas:

Selector de lados de la figura: Con este selector, que podemos ver en la figura (Figura 3 subfigura a), podemos escoger la cantidad de lados del objeto, tanto para las figuras regulares como para la opción de mano alzada. La cantidad de lados a elegir va desde un mínimo de tres lados hasta un máximo de ocho lados.

Selector de Vértices y desplazamiento: Con este selector, que podemos ver en la figura (Figura 3 subfigura b), podemos escoger cada uno de los vértices de la figura y modificar su posición tanto en el eje X como en el eje Y.

Casillas de verificación de conectores: Con la casilla de verificación "Conectores" escogeremos que la figura tenga conectores o no en su totalidad. Con las casillas "Lado"

podremos escoger para cada lado concreto de la figura si deseamos que tenga conectores o no. Como podemos observar en la figura (Figura 3 subfigura c) tenemos que tener en cuenta que los conectores sólo se dibujarán en el caso que el lado tenga el tamaño necesario para poder dibujarse.

Selector de alto de la pieza: Con este selector, como puede observarse en la figura (Figura 3 subfigura d), podemos modificar el alto de la pieza entera (eje X).

Selector de ancho de la pieza: Con este selector, ver figura (Figura 3 subfigura e), podemos modificar el ancho de la pieza entera (eje Y).

Selector de profundidad de la pieza: Con este selector, que podemos ver en la figura (Figura 3 subfigura f), se nos permite modificar la profundidad de la pieza entera (eje Z).

Selector de perforación de la pieza: Con este selector, como podemos ver en la figura (Figura 3 subfigura g), se puede decidir que la pieza esté perforada o no y, en el caso de que decidamos perforarla, se permite elegir el grosor de la perforación.

Botón Guardar figura: Guardamos la configuración de la pieza que estamos actualmente dibujando mediante el botón que podemos ver en la figura (Figura 3 subfigura h).

Botón Cargar figura: Cargamos la configuración de una pieza creada previamente mediante el botón que podemos ver en la figura (Figura 3 subfigura i).

Botón para crear STL: Generamos un archivo ".STL" de la figura actualmente dibujada para poder ser impresa en una impresora 3D seleccionando el botón que podemos ver en la figura (Figura 3 subfigura j).

Botón Reset: Reinicia el estado de la pieza al estado inicial de la aplicación a partir de la pulsación del botón que podemos ver en la figura (Figura 3 subfigura k).

Botón de Dibujo Libre: Este botón nos permite comenzar una nueva figura en la modalidad de dibujo libre utilizando el botón que podemos ver en la figura (Figura 3 subfigura l).

Casilla de verificación *Smooth*: Con esta casilla, que podemos ver en la figura (Figura 3 subfigura m), decidiremos si queremos que se aplique a la pieza un suavizado de los vértices.

- Con el selector “VERS” elegiremos cuántos vértices adicionales añadiremos durante el suavizado.
- Con el selector “GIRO” elegiremos el grado de separación entre los nuevos vértices añadidos.

Botón *Exit*: Botón con el que salimos de la aplicación, ver figura (Figura 3 subfigura n). También podemos pulsar la tecla Esc.

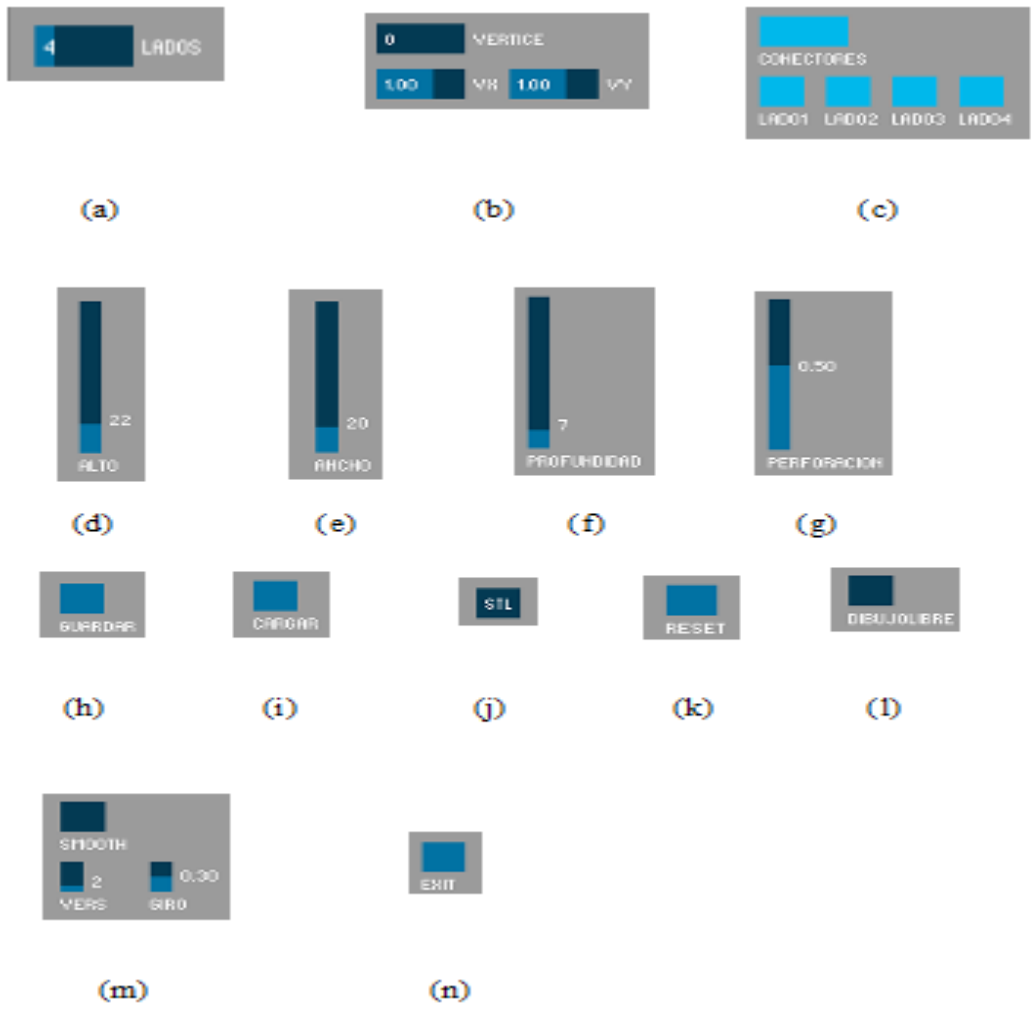

Figura 13: Botones, sliders y casillas

3.1.2 *Proceso de gestión de eventos*

En este bloque veremos cómo funciona internamente la aplicación mediante la utilización de los distintos botones, casillas y *sliders* utilizados durante la interacción.

A continuación, veremos una breve explicación de las funciones:

controlEvent: Gestiona todas las pulsaciones y modificaciones de los botones casillas y *sliders*. La mayoría de los cambios son simples modificaciones de algún valor de una variable. Más adelante veremos los casos más complejos.

modificarLados: Modifica el valor de lados del prisma al nuevo valor seleccionado. A su vez, modifica el valor del selector de vértice para evitar un conflicto con el número de vértices y muestra únicamente las casillas de **Lado** correspondientes a la cantidad de lados seleccionado.

tratarConectores: Modifica la figura para que en ella aparezcan o no conectores en todos los lados. Con esta función podemos variar el valor de todos los lados, independientemente de que tuviésemos modificado cada lado individualmente.

dibujoLibre: Nos permite entrar o salir del modo de dibujo libre, modificando para ello ciertos valores para evitar desajustes al dibujar la figura con el ratón. También inicializa la posición de la cámara para realizar un correcto dibujado y queda a la espera de las pulsaciones de ratón para dibujar la figura.

3.1.3 *Inicialización de la aplicación*

A continuación, veremos una breve explicación de las funciones:

Setup: Inicialización básica de la ventana de ejecución, con la velocidad de visión y suavizado en el visionado.

Draw: Es la función que se ejecuta continuamente para dibujar tanto menús como figuras.

CamaraP5: Una función necesaria para que los giros de cámara, que nos permiten ver la pieza desde distintos ángulos y distancias, no afecte a los menús.

MousePressed: Función que está a la espera de los *clicks* de ratón cuando estemos en modo de dibujo libre, dichos *clicks* serán los utilizados para posicionar en pantalla los vértices de la figura.

Inicializar: Función que crea los objetos de dibujado de piezas, cámara, botones, *sliders* y casillas.

3.1.4 *Visualización de figuras*

A continuación, presentamos una breve explicación de las funciones relacionadas con la tarea de "Visualización de Figuras".

dibujar_figuras: Función llamada por la función **Draw** citada anteriormente. Esta función se encarga de crear y mostrar el eje de coordenadas y la rejilla. También permite mostrar la figura en pantalla creada por la función **initMesh** que veremos a continuación.

InitMesh: Función que comprueba la manera de dibujar la figura, ya sea de manera regular, en dibujo libre o si se está cargando una figura previamente guardada. Todas estas acciones se realizan mediante llamadas a las siguientes funciones.

DibujarPoligonoRegular: Función que dibuja la figura cuando estemos en el modo de dibujado normal, que es como se inicializa la aplicación.

DibujarPoligonoDibujoLibre: Función que dibujará la figura cuando estemos en el modo de dibujado de dibujo libre.

Cargar: Función que crea la figura a partir de la información contenida en un fichero. Una vez cargada la figura se puede seguir modificando.

Guardar: Se guarda la información más relevante de una pieza en un fichero para poder utilizarla más adelante.

Posicionar y Transformar: Funciones que se utilizan en el modo de dibujo libre para

que la posición de los puntos que se van indicando con el ratón aparezcan en el sitio en el que hacemos *click*. Estas funciones las explicaremos más adelante con mayor detalle.

Lámina: Función que, una vez que tenemos todos los vértices, se encarga de crear las caras superior e inferior de la figura, teniendo en cuenta si las caras están perforadas o no.

Almena: Función que crea las tiras laterales de la figura y tiene en cuenta si se ha indicado que tenga o no conectores para generarlos en las tiras. En caso de tener conectores, se utiliza la función **Celda.** Una vez que genera la tira esta función se encarga de colocar correctamente las tiras junto con las láminas superior e inferior.

Celda: Función que crea los conectores para añadirlos posteriormente a las tiras laterales.

***Smooth*:** Función que se encarga de realizar el suavizado de los ángulos de la figura en caso de que se haya seleccionado esta opción.

3.2 *Diseño de las clases*

En este apartado explicaremos el diseño interno de objetos, atributos y métodos de las clases que utilizamos para la creación de la aplicación.

3.2.1 *Polígono*

El polígono es nuestra clase básica para poder representar figuras. Su principal cometido es el de guardar los vértices en nuestra aplicación.

3.2.1.1 *Funcionalidad: escalado*

Para la funcionalidad de escalado, se necesitan únicamente los vértices de la figura y un valor entre 0 y 1 que indica el escalado a aplicar a los vértices de la figura.

Esta funcionalidad de escalado se utiliza para realizar la perforación de la figura, ya que la perforación dentro de la figura no es otra cosa que la diferencia entre la pieza original y una copia de esta, pero de menor tamaño.

3.2.1.2 *Funcionalidad: suavizado*

Para la funcionalidad de *smooth* o suavizado se utilizan los vértices de la figura. Esta función precisa de dos parámetros adicionales: el número de vértices que se añaden a cada lado y el ángulo de giro que se aplica a cada vértice añadido de esta forma.

3.2.2 *Lámina*

El objeto **Lámina** es el encargado de crear la figura a partir de la información proporcionada por la clase polígono anteriormente comentada.

Para crear la lámina o cara de la figura se crean unos triángulos utilizando, para ello, el centro de la figura y pares de vértices. Se crean triángulos hasta que se termina de tratar todos los vértices almacenados en el objeto **Polígono**.

De esta manera se crean las caras superior e inferior de la figura. En este momento del proceso se tiene que tener en cuenta si tenemos que realizar la perforación de la pieza o no. En el caso de que la pieza requiera de una perforación, se utiliza la clase **Polígono** para crear las caras perforadas.

Una vez creadas las caras, se generan las tiras laterales con la ayuda de la clase ***QuadStrip*** que veremos a continuación. Al crear estas tiras laterales se tiene en cuenta si la pieza va a tener conectores o no, ya sea por decisión nuestra o porque, por la configuración de la pieza, determinados lados no cumplen las condiciones necesarias para poder tener conectores.

3.2.3 *QuadStrip*

El objeto ***QuadStrip*** se utiliza para crear las caras laterales. En principio este objeto es

similar a la clase **Polígono**, con la diferencia de que, en este caso, los vértices que se utilizan tienen que ser en 3 dimensiones ya que debemos tener en cuenta la profundidad de la figura.

Estas caras laterales se crean a partir de cuatro vértices, correspondientes a los dos vértices de la cara superior y los otros dos de la cara inferior. Se crean las caras de forma similar a la utilizada en la creación de los triángulos a partir de los vértices, con la diferencia que, en este caso, no se utiliza el centro de la cara sino que se forman únicamente los triángulos a partir de los vértices.

3.3 *Implementación*

Una vez realizados el análisis y el diseño de la aplicación, mostramos la implementación realizada mediante el lenguaje de programación *Processing*. En este capítulo trataremos los algoritmos más complejos e importantes de la aplicación.

3.3.1 *Creación del polígono regular*

Esta función se encarga de generar los vértices de los polígonos regulares. La función puede obtener tantos vértices como le indiquemos pero, para no saturar la pantalla de botones, hemos limitado la cantidad de lados a 8.

Aunque esta función no es muy compleja es la base de la creación de los polígonos regulares. Básicamente se limita a dividir una circunferencia en partes iguales según el número de lados indicado, y guarda estos vértices en nuestro tipo de datos **Polígono**.

Después comprueba si la opción de suavizado se encuentra seleccionada a fin de añadir los vértices necesarios al polígono para esta operación.

Y por último, se construye la figura a partir del tipo de dato **Lámina** y del polígono creado.

```
for (int i = 0; i < lados; i++){

 p.add(verticesX[i+1]*ancho*sin(radians(45+(360/lados)*i))
 ,verticesY[i+1]*alto*cos(radians(45+(360/lados)*i)));

    }
    if (smooth){

      p.smooth(vers,giro);

    }
    Lamina lam = new Lamina(p, Perforacion);
    mesh.addMesh(lam);
```

Código 1: Función que crea el polígono regular

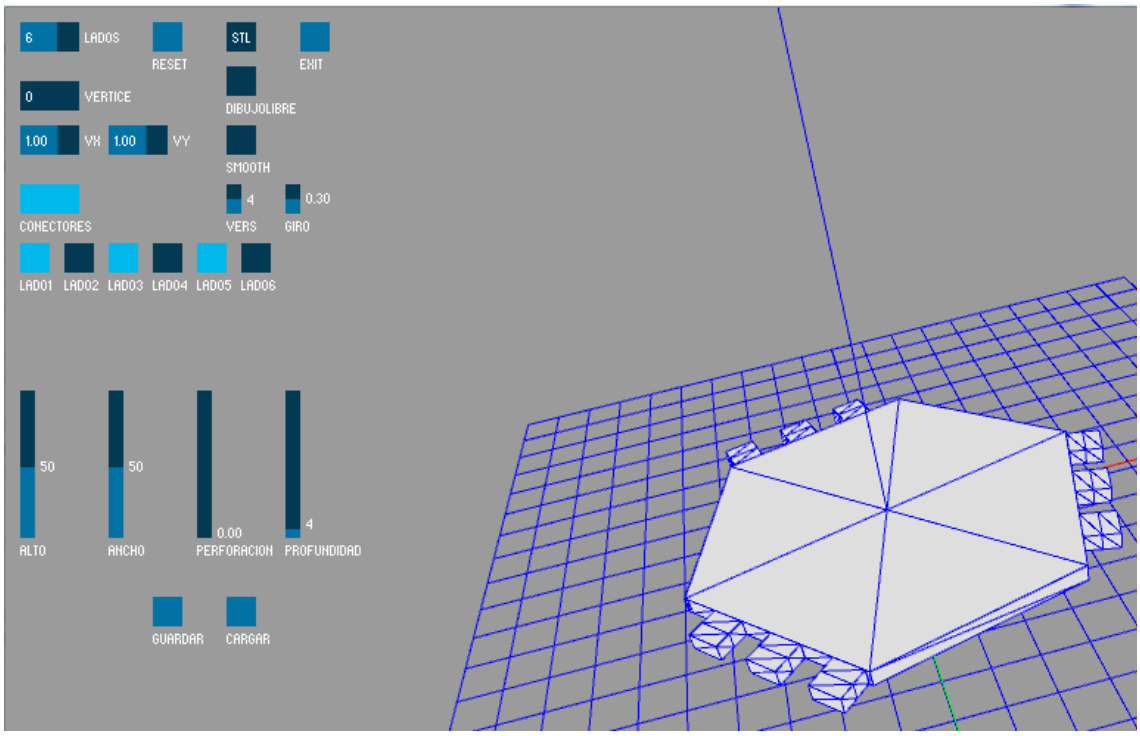

Figura 14: Polígono regular

3.3.2 *Creación de un polígono en dibujo libre*

Esta función se encarga de generar los vértices que se guardan en nuestro tipo de datos **Polígono** a partir de las pulsaciones de ratón en la pantalla. No obstante, antes de guardar esos vértices, la función nos proporciona otra utilidad en el dibujo libre como

comentaremos a continuación.

Una de estas ayudas visuales es la de ir añadiendo puntos en la pantalla allí donde hagamos *click* con el ratón.

Y, como segunda ayuda visual, una vez realizado el primer *click* se dibuja una línea desde el punto hasta donde tengamos el puntero del ratón. Y así sucesivamente para todos los *clicks* realizados. Las líneas dibujadas se mantienen dibujadas entre punto y punto no solamente entre el último punto y el puntero del ratón, lo cual nos proporciona así una previsualización de cómo va a quedar nuestra figura actual. En este caso también al igual que en el de dibujo del polígono regulares hemos limitado el número de vértices a 8.

Cuando se termina de introducir los vértices manualmente es cuando la función crea el polígono y, a partir de este momento, se posibilita la creación de la figura para verla en pantalla realizando los últimos ajustes.

Después se comprueba si está seleccionada o no la opción de *smooth* o suavizado para añadir los vértices necesarios al polígono.

Y, por último, se finaliza la construcción de la figura y se presenta en pantalla utilizando para ello el tipo de dato **Lámina** asociado al polígono recién creado.

```
  int auxvertice = 1;
  if(!crearpoligono){
     for (Vec2D pu : points) {
     //dibujar los puntos de dibujo libre
     ellipse((pu.x - width/2)*zoom,(pu.y -height/2)*zoom,5,5);
   }
   if (points.size() > 1){ // dibujar lineas entre puntos
      for (int j = 0; j < points.size()-1; j++){
        line(posicionar(points.get(j).x,"ancho"),

posicionar(points.get(j).y,"alto"),posicionar(points.get(j+1).x
,"ancho"),
        posicionar(points.get(j+1).y,"alto"));
      }
    }
    if (points.size() > 0){
      line(posicionar(points.get(points.size()-1).x,"ancho"),
      posicionar(points.get(points.size()-1).y,"alto"),
      posicionar(mouseX,"ancho"), posicionar(mouseY,"alto"));
    }
  }
  if (crearpoligono){
    auxvertice = 1;
    for (Vec2D po : points) {
      p.add(verticesX[auxvertice]*transformar(po.x,"ancho"),
      verticesY[auxvertice]*transformar(po.y,"alto"));
      auxvertice ++;
    }
  }
```

Código 2: Función para crear un polígono en dibujo libre

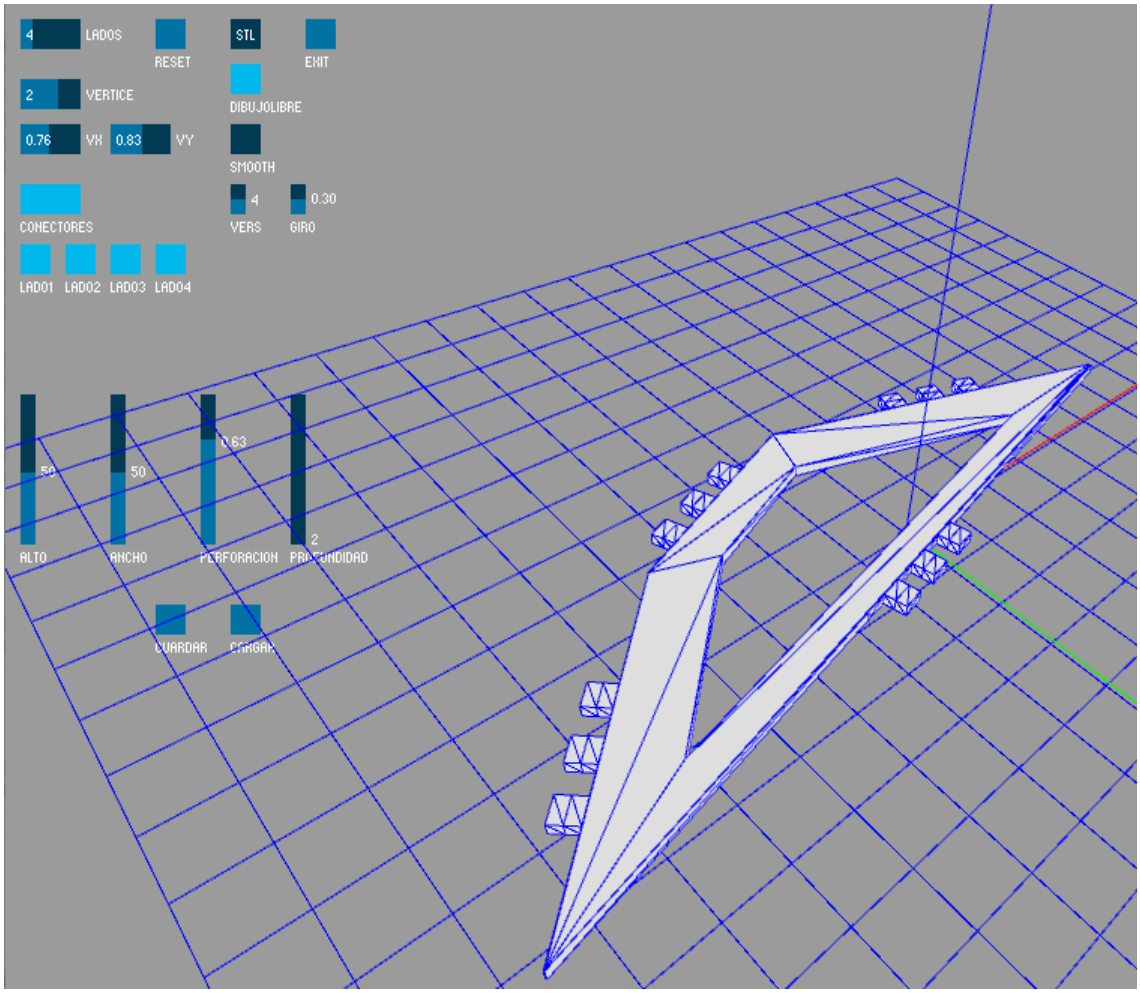

Figura 15: Polígono en dibujo libre

3.3.3 *Guardar figura*

Esta función se encarga de guardar la información necesaria para poder cargar nuevamente la figura deseada.

La información almacenada es la siguiente:

- La cantidad de lados de la figura.
- El alto de la figura.
- El ancho de la figura.

- La profundidad de la figura.
- La perforación de la figura.
- La posición de los vértices de la figura.

Toda esta información se almacena en un archivo de texto con extensión ".txt" para su posterior utilización.

Una información que nunca se almacena en el fichero es la opción de *smooth* o suavizado, ya que esta opción crea una gran cantidad de vértices adicionales que tendríamos que tratar tanto para guardar como para cargar la figura. Teniendo en cuenta que con un simple *click*, podemos activar o desactivar esta opción no tiene sentido guardar dicha información.

```
output = createWriter("positions.txt");
output.println(lados);
output.println(alto);
output.println(ancho);
output.println(Perforacion);
output.println(profundidad);
for (Vec2D ve : p.vertices) {
  output.println(ve.x+" / "+ve.y);
}
output.flush(); // Writes the remaining data to the file
output.close(); // Finishes the file
```

Código 3: Función que guarda la información en el fichero

3.3.4 *Cargar figura*

Esta función se encarga de cargar la información previamente almacenada en un fichero de texto ".txt" para dibujar piezas que hubiésemos guardado previamente.

La función abre el archivo de texto en caso de que exista y empieza a leer la información de este, línea a línea, y guarda la información para la correcta creación de la figura.

Una vez obtenidos los datos de las principales variables de la figura, previamente mencionadas en la opción de "Guardar figura", se leen los vértices de la figura y se almacenan en el tipo de datos **Polígono**.

Después, se comprueba si se está marcando la opción de *smooth* o suavizado para añadir los vértices necesarios al polígono.

Y por último, y mediante el tipo de dato **Lámina**, al cual le pasamos el polígono que acabamos de crear y el valor de perforación seleccionado, se finaliza la construcción de la figura y se presenta en pantalla.

```
    reader = createReader("positions.txt");
    try {
      line = reader.readLine();
    }
    catch (IOException e) {
      e.printStackTrace();
      line = null;
    }
    while (line != null){
      if(numlinea < 6){
        if (!cargado){
          switch(numlinea){
          case 1 :
            controlP5.getController("lados").
            setValue(float(line));
            break;
          case 2 :
            controlP5.getController("alto").
            setValue(float(line));
            altold = float(line);
            break;
          case 3 :
            controlP5.getController("ancho").
            setValue(float(line));
            anchold = float(line);
            break;
          case 4 :
            controlP5.getController("Perforacion").
            setValue(float(line));
            break;
          case 5 :
            controlP5.getController("profundidad").
            setValue(float(line));
            break;
        }
        if (numlinea == 5){cargado = true;}
      }
        try {
        line = reader.readLine();
        numlinea ++;
```

```
      }
      catch (IOException e) {
        e.printStackTrace();
        line = null;
      }

      }
      else{
      float[] nums = float(split(line, " / "));
      p.add(verticesX[numlinea-5]*nums[0]*(ancho/anchold)
      ,verticesY[numlinea-5]*nums[1]*(alto/altold));
      try {
        line = reader.readLine();
        numlinea ++;
      }
      catch (IOException e) {
        e.printStackTrace();
        line = null;
      }
      }
    }
```

Código 4: Función que carga la información del fichero

3.4 *Descripción de los problemas y errores encontrados durante la implementación y sus posibles soluciones*

En este apartado describiremos los distintos problemas encontrados durante la implementación de la aplicación y las soluciones que se han aplicado para el correcto funcionamiento de la misma.

3.4.1 *Cámara*

Al utilizar la cámara para poder visualizar de una manera libre la pieza creada, nos hemos encontrado con diversos problemas. Uno de los problemas más importantes, ha sido que, junto con la figura, también se movía todo el sistema de menús. Hecho que no era para nada deseado.

Para poder solucionarlo hemos extraído la parte del menú del redibujado de la cámara haciendo que se muestre siempre estático.

```
cam.beginHUD();
controlP5.draw();
cam.endHUD();
```

Código 5: Función que permite que el menú permanezca estático

Otro problema que hemos tenido con el uso de la cámara, es el modo de control de la misma cuando se utiliza el ratón. Cuando interaccionamos con el menú, también se producían cambios en la cámara. Nuestro objetivo es que cuando estemos en la zona de menús, el ratón sólo sirva para controlar los menús, y que cuando estemos fuera de estos el ratón sirva para controlar la cámara.

Este problema, lo hemos solucionado deshabilitando el control de la cámara mediante el ratón en la zona de los menús. De esta manera en la zona de los menús sólo modificamos los menús con el ratón.

```
cam.setMouseControlled(false);
```

Código 6: Código para evitar el control de la cámara con el ratón

3.4.2 *Zoom*

Nos hemos encontrado algunos problemas al calcular los valores de los puntos en pantalla. El error de visualización de los puntos se debía al *zoom* que se aplica en cada momento. Ya que, por ejemplo, al hacer *click* en la pantalla para colocar un vértice, el valor del punto en la pantalla depende del *zoom* de la cámara y no tiene el mismo valor que el que debiera tener el vértice realmente.

Para solucionar este problema del *zoom,* tenemos que calcular la posición del *zoom* en cada momento, aplicar este valor a los vértices a fin de que estos se vean correctamente.

```
zoom = (float) (cam.getDistance()/600);
```

Código 7: Valor que necesitamos para calcular la posición real de los vértices

3.4.3 *Creación de caras*

Una vez creada una figura, podemos desear imprimirla en una impresora 3D. No obstante, hemos detectado que ciertas caras de la figura no se imprimían correctamente. Al abrir el archivo que contiene la pieza a imprimir mediante el programa **netfabb® Studio** descubrimos el porqué del error en la impresión ya que algunas de las caras de las figuras se encontraban creadas de forma inversa. Es decir, al realizar la impresión de cada cara, la impresora diferencia entre la cara exterior y la cara interior. La cara exterior es la que se imprime y la cara interior queda dentro de la figura y es la que no se ve. En nuestro caso, descubrimos que ciertas caras se hallaban intercambiadas y por ello, el producto final no era el deseado.

Para solucionar este error, debemos asegurarnos que las llamadas de creación de caras se realicen en el orden correcto para que las caras no queden invertidas.

3.4.4 *Profundidad de las figuras*

En la creación de las figuras, tuvimos problemas con la profundidad de las figuras (eje Z). Sobre todo al colocar las tiras laterales, ya que estas no quedaban en el lugar en el que queríamos. Inicialmente detectamos que parte del problema se debía al problema anteriormente comentado sobre el *zoom,* pero también nos percatamos que no sólo este era el problema, ya que al modificarse la profundidad no quedaban bien encajadas las caras superior e inferior mediante las tiras laterales. Esto era debido a cómo afectaba la profundidad, tanto en las caras superior e inferior como en las tiras laterales ya que estando posicionadas cada una a un nivel distinto del eje Z sufrían distintas modificaciones.

Para solucionar este error, debemos ver cómo afectaba a cada paso la modificación en profundidad y corregirla progresivamente, tanto en las caras superior e inferior y las tiras laterales.

Capítulo 4: RESULTADOS OBTENIDOS

En este capítulo se presentan algunos de los resultados obtenidos de la realización de este proyecto. Para ello mostramos distintas visualizaciones correspondientes a las diversas funciones realizadas que ilustran las principales características de la aplicación

4 *Pantalla Inicial*

Inicialmente se muestra la pantalla de carga de la aplicación como vemos en la siguiente imagen (Figura 16). Puede observarse que aparecen todas las opciones descritas anteriormente.

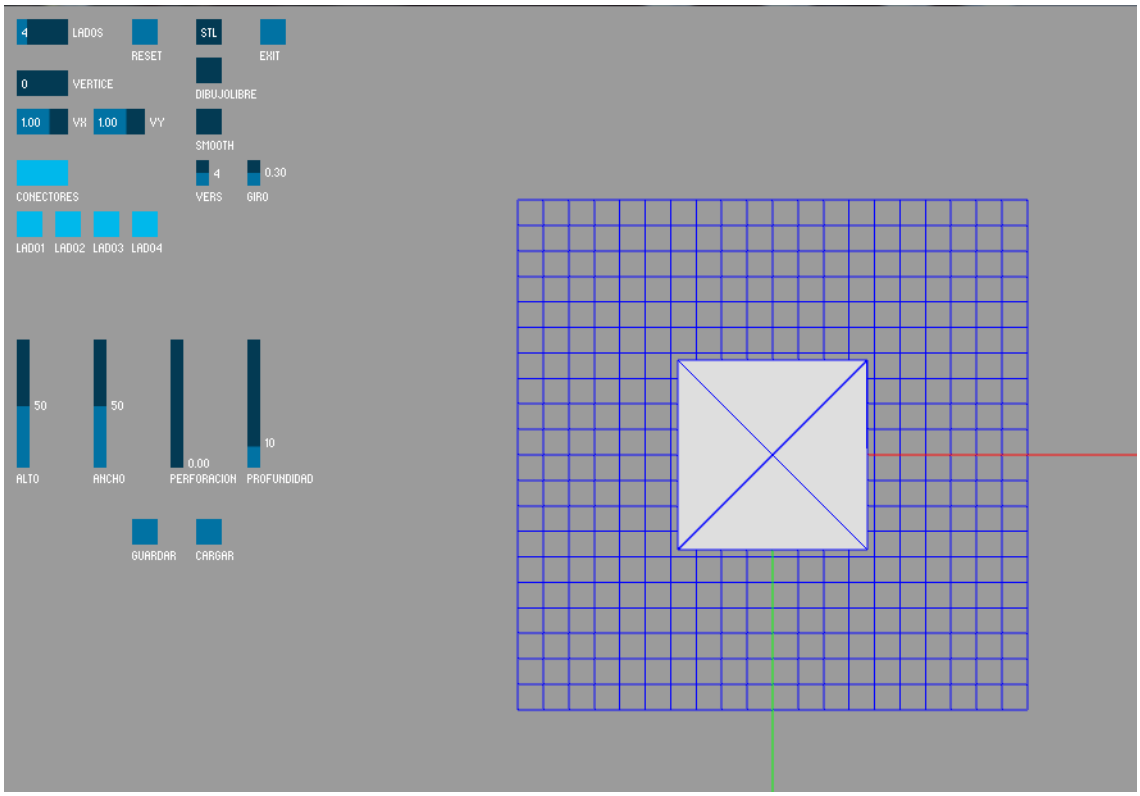

Figura 16: Pantalla inicial

4.1 *Polígono regular*

En la siguiente figura (Figura 17) podemos ver la capacidad de cración de figuras regulares y las distintas opciones de la aplicación.

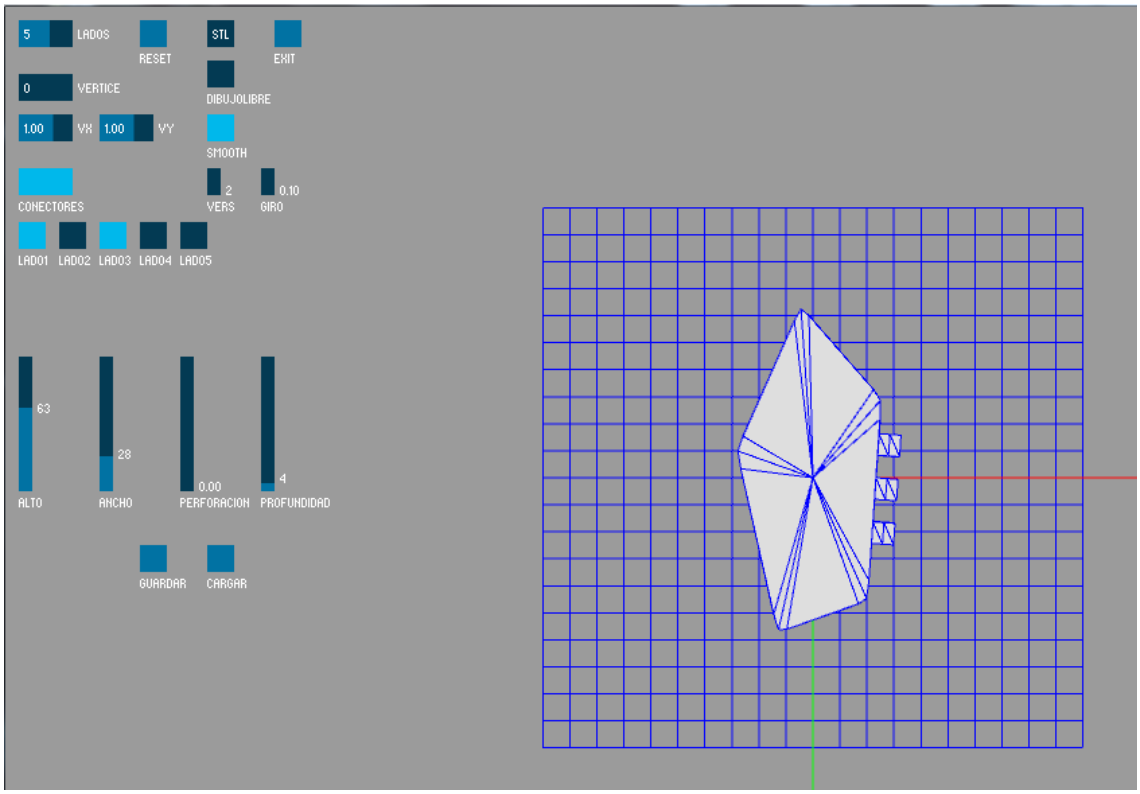

Figura 17: Creación de polígono regular

4.2 *Polígono en dibujo libre*

A continuación mostramos la creación de figuras en modo libre (Figura 18).

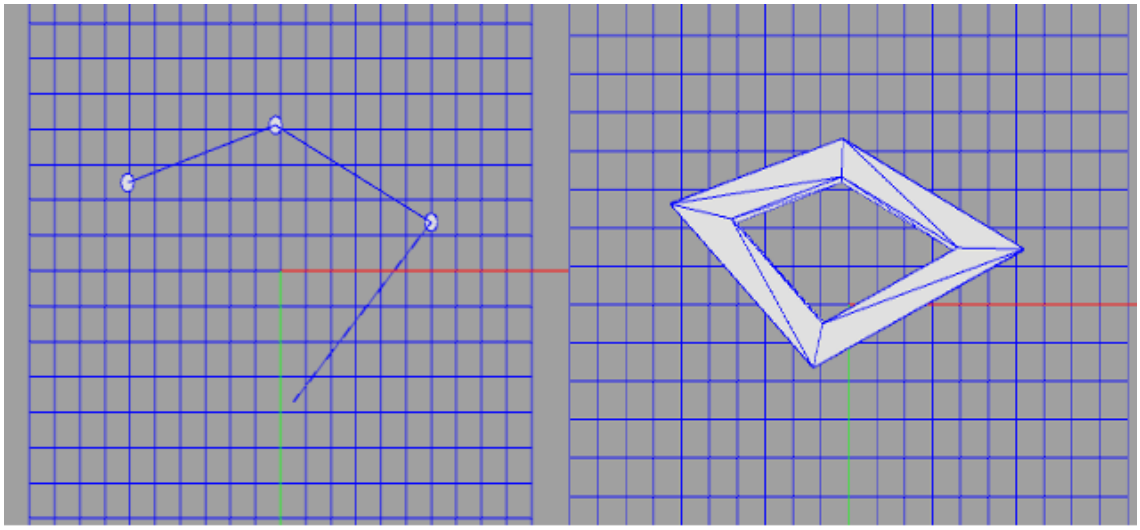

Figura 18: Creación de polígono en dibujo libre

4.3 *Creación de archivo ".STL"*

En la siguiente imagen (Figura 19) mostramos el polígono anteriormente creado, mediante el visor de archivos ".STL".

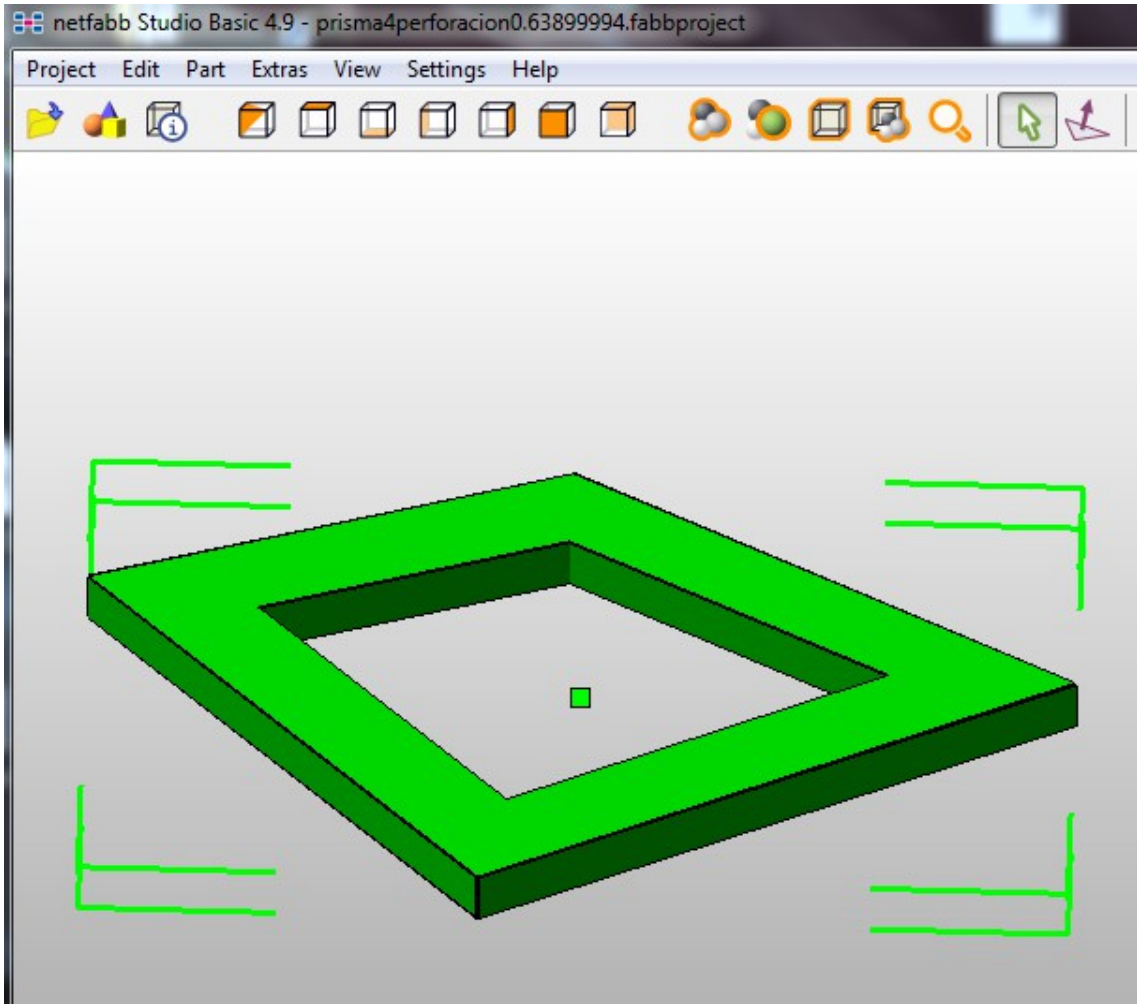

Figura 19: Polígono visto en Netfabb Studio

4.4 *Guardar y cargar polígonos*

Por último mostraremos las distintas posibilidades de la aplicación a la hora de guardar y cargar un polígono, mostrando el paso de la pantalla inicial al polígono creado previamente en dibujo libre en la siguiente imagen (Figura 20).

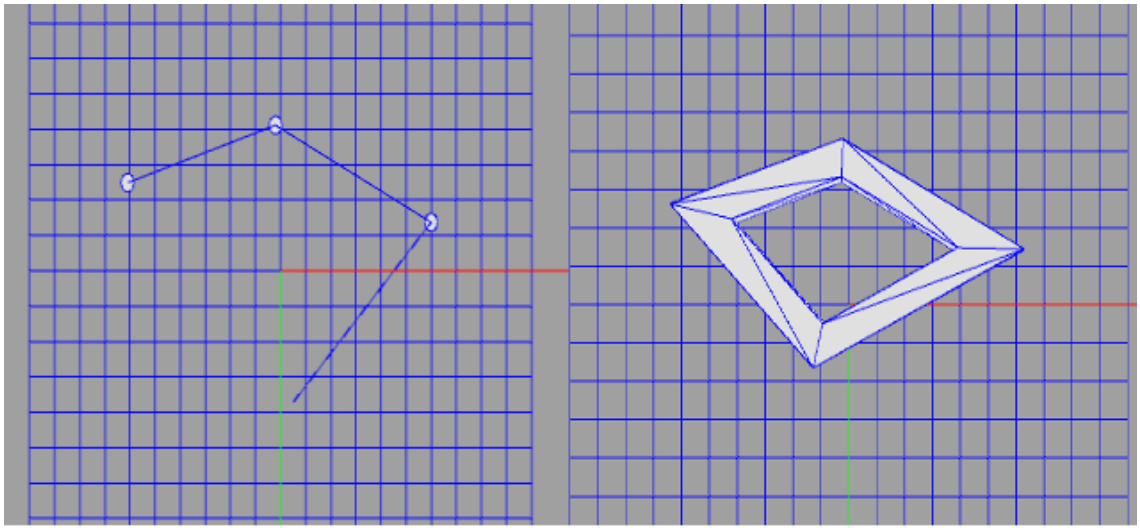

Figura 20: Guardar y cargar polígonos

Capítulo 5: CONCLUSIONES Y LÍNEAS FUTURAS

En este capítulo de la memoria, presentaremos la concordancia entre los objetivos planteados al principio del proyecto y los resultados obtenidos una vez finalizado. Por otra parte, se exponen las conclusiones finales del proyecto y un resumen de las futuras líneas de trabajo que se podrían adoptar para la mejora y ampliación del proyecto.

5 *Concordancia entre objetivos y resultados obtenidos.*

El objetivo principal del proyecto era la elaboración de un interfaz visual, que permitiera a usuarios inexpertos en programación la creación de figuras sencillas para su posterior impresión en una impresora 3D.

Podemos asegurar que este objetivo principal y más importante se ha cumplido, puesto que la creación de figuras, tanto regulares como de dibujo libre, funcionan correctamente cómo hemos visto en el capítulo anterior.

Para poder imprimir estas figuras creadas, era necesario pasar esta información a un tipo de archivo, que pudiesen leer las impresoras. El tipo de archivo seleccionado fue el ".STL". Y por lo tanto, podemos decir que este objetivo también ha sido cumplido, permitiendo imprimir las figuras creadas correctamente.

Por último, se requería un sistema de archivos para la apertura y guardado de figuras realizadas. Esta última tarea también se ha sido realizada correctamente.

5.1 *Conclusiones*

Después de haber finalizado la aplicación, y de haber analizado los resultados, se han obtenido las siguientes conclusiones a nivel profesional e individual.

A nivel profesional, cabe destacar que el cumplimiento de los principales objetivos del proyecto, aunque estos surgieron de forma espontánea en el transcurso de un taller y los objetivos no estaban completamente desarrollados, han sido aceptados satisfactoriamente por parte de la empresa.

A nivel personal, debo destacar el estudio de un nuevo lenguaje de programación, y el acercamiento a las tecnologías de las impresoras en 3D.

5.2 *Líneas futuras*

En este apartado expondremos las posibles líneas futuras que la empresa podría adoptar para mejorar o incrementar la funcionalidad de la aplicación:

1. Se puede introducir un mecanismo de modificación de vértices adicional al ya creado, que se realice con el ratón en vez de con el uso de *sliders*.

2. Se pueden añadir nuevas funciones de modificación de las figuras una vez dibujadas como por ejemplo la inserción o eliminación de vértices.

3. Se puede modificar la creación del polígono para que este no se cree desde el centro de los vértices introducidos, sino que la creación del polígono sea en base a los propios vértices solamente.

4. Se puede modificar el aspecto visual para sacar el máximo provecho a la creación de polígono y superar ampliamente el límite de 8 vértices dibujables.

5. Se pueden añadir tanto botones de ayuda para un mejor entendimiento de la aplicación incluso algún pequeño tutorial o vídeo explicativo.

Capítulo 6: BIBLIOGRAFÍA

[1] Larry Page, Sergey Brin (1998). Google INC. www.google.com

[2] Jimmy Wales y Larry Sanger (2001). Fundación Wikimedia. www.wikipedia.com

[3] Ben Fry, Casey Reas (2001). Processing. www.processing.org

[4] Jonathan Feinberg. Peasycam

http://mrfeinberg.com/peasycam/

[5] Daniel Shiffman (2008). Learnning Processing. www.learningprocessing.com

[6] Andreas Schlegel (2000). Sojamo www.sojamo.de

http://www.sojamo.de/libraries/controlP5/

[7] Tom Cardem. Tube Map.

http://www.tom-carden.co.uk/p5/tube_map_travel_times/applet/

[8] Karsten Schmidt (2012). Toxiclibs http://toxiclibs.org/

[9] Jer Thorp. Just Landed.

http://blog.blprnt.com/blog/blprnt/just-landed-processing-twitter-metacarta-hidden-data

[10] Marius Watz. Gas Works 1-3.

http://www.unlekker.net/proj/gasworks/