UNIVERSIDAD DE SANTIAGO DE CHILE

FACULTAD DE INGENIERÍA

Departamento de Ingeniería Informática

Herramienta de diseño de objetos y visualización de resultados para simulación de campo magnético

Juan Pablo Verdejo Jorquera

Profesor guía: Fernando Rannou, Ph.D.

Trabajo de titulación presentado
en conformidad a los requisitos para
obtener el título de Ingeniero de
Ejecución en Computación e Informática

Santiago – Chile
2015

TABLA DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

Escribir acá el resumen en español.

Palabras Claves: escribirKeyword1; escribirKeyword2; escribirKeyword3; etc

ABSTRACT

Write the abstract here.

Keywords: writeKeyword1; writeKeyword2; writeKeyword3; etc

CAPÍTULO 1. INTRODUCCIÓN

1.1 ANTECEDENTES Y MOTIVACIÓN

Un grupo de científicos del Departamento de Física de la Universidad de Santiago de Chile trabaja en la simulación de los efectos del campo magnético en los átomos de distintos objetos, tomando en cuenta su forma, su material y su distribución atómica, entre otras características, usando una aplicación escrita en C empleando el Método de Monte Carlo (Metropolis & Ulam, 1949).

No obstante lo anterior, uno de los grandes problemas de los científicos es que el proceso previo y posterior a la simulación es “manual”, para definir el objeto deben hacer un dibujo en Microsoft Paint®, el cual es analizado por un *script* de Matlab® que debe ser modificado para reflejar las características del objeto en específico que se quiere simular. Para generar imágenes para, por ejemplo, una publicación, también deben ejecutar ciertos *scripts*, sin embargo esto es aún más complicado, ya que deben hacer ensayo y error hasta conseguir que la imagen sea representativa del resultado, puesto que muchas veces tienen errores de visualización que no reflejan el estado real. Estos dos sub-procesos hacen que el proceso de simulación sea tedioso, quitando mucho tiempo que podría ser usado en analizar los resultados.

Es aquí donde la informática puede contribuir, creando aplicaciones que mejoren estos procesos, que valoricen el tiempo de los científicos. Por consiguiente esta es una oportunidad única de ayudar a la obtención de conocimientos que permitan entender el entorno y de mejorar los procesos que permiten avanzar como sociedad hacia la comprensión del universo.

1.2 DESCRIPCIÓN DEL PROBLEMA

Actualmente no existe un método estándar de diseño de objetos para aplicar la simulación Monte Carlo, que permita definir características físicas y geométricas y crear un archivo de entrada, que describa cada uno de los átomos, para la simulación, por lo que cada nuevo investigador en general usa como base las herramientas generadas por antiguos tesistas y lo van modificando para aplicarlo a sus investigaciones. Tampoco existe una solución para la visualización de los resultados entregados y la posterior exportación de estos para publicaciones y conferencias.

1.3 PROCESO ACTUAL

1.3.1 Diseño de objetos para simular

Actualmente los científicos no tienen un proceso definido y formal de creación de objetos sobre los cuales se van a simular, ya que existen diversas soluciones implementadas a través del tiempo, desde *scripts* que fueron creados especialmente para diseñar un objeto en específico hasta uno más flexible, pero igualmente engorroso, usando como base un mapa de bits.

1.3.1.1 Diseño programático de objetos

La primera opción para diseñar objetos sobre los cuales se correra la simulación es de manera programática, es decir, escribiendo un *script* que defina todas las partículas de un objeto y sus vecindades, para alguien con experiencia en esta área puede ser relativamente sencillo diseñar algunos objetos regulares,

como por ejemplo un cilindro, no obstante esto se complica si se quieren diseñar estructuras irregulares. Dado que la visualización del objeto diseñado es de muy baja calidad con este sistema es fácil cometer errores que eventualmente pueden invalidar los resultados de una simulación.

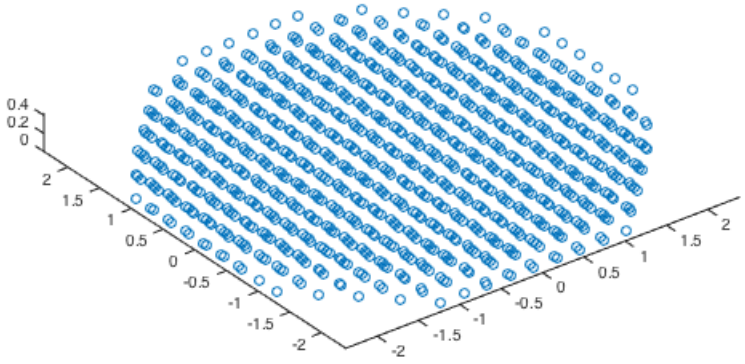

FIGURA 1.1: *Ejemplo de imagen salida de un* script *de generación de objeto*

1.3.1.2 Diseño en base a mapa de bits

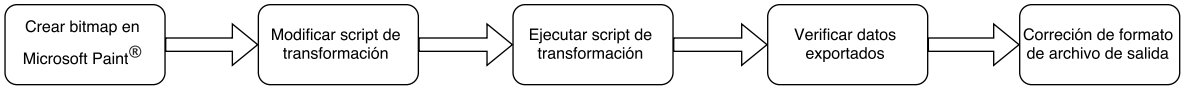

FIGURA 1.2: *Flujo para creación de diseño en base a mapa de bits*

Para flexibilizar el diseño de objetos y bajar la posibilidad de errores se creó un sistema que permite hacer este proceso basándose en un mapa de bits, de tal forma que ya no fuese necesario programar todo desde cero, si no que bastaba con crear una imagen que representara la primera capa del objeto y usar el script correcto en base a los parámetro físicos de este.

Para iniciar este proceso es necesario tener un editor de imágenes que permita crear archivos .bmp, como por ejemplo Microsoft Paint®, luego crear una imagen del tamaño deseado, la cual usualmente no excede los 50px x 50px, y ampliarla al máximo de forma de poder trabajar a nivel de pixeles. Luego se deben pintar, con color negro, los pixeles que representen la primera capa de átomos del objeto, estos pixeles luego serán identificados por un *script* de *Matlab*.

En la siguiente imagen se muestra un ejemplo de estos mapas, usando una imagen de 10px x 10px, con una cruz centrada.

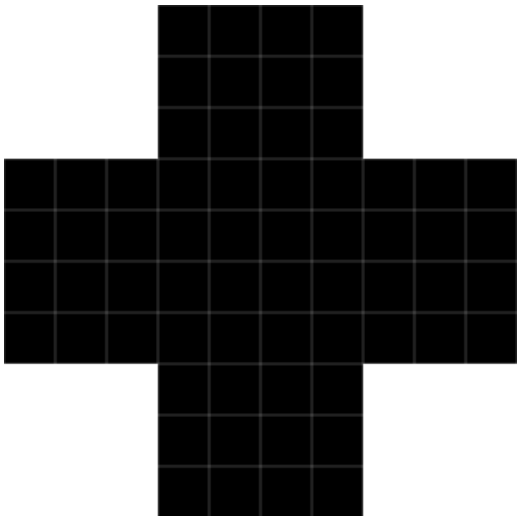

FIGURA 1.3: *Mapa de bits de 10px x 10px*

Luego deben modificar un *script* de *Matlab*, el cual tiene múltiples versiones según el tipo de estructura cristalina que se quiera diseñar, de tal forma de ingresar los parámetros físicos del objeto. Algunas de las opciones que se deben configurar antes de iniciar la ejecución son:

Número de capas

Cuantas veces se repite la primera capa para formar un objeto en 3D.

Coeficiente de escalamiento

Es un coeficiente usado para dar al objeto las medidas deseadas para ejecutar la simulación.

Constante de red

Esta es la dimensión de la arista de una celda unitaria

Todos estos parámetros deben ser modificados directamente en el código, en distintas partes de este, por lo que es muy fácil cometer errores, los que por supuesto invalidan cualquier simulación hecha en base a estos.

Una vez configurado el script debe ser ejecutado, este leerá el mapa de bits ingresado como parámetro analizando cada uno de los pixeles, creando un arreglo binario de 2 dimensiones, donde un pixel negro es representado por 1 y un pixel blanco es representado por un 0, lo que creará la primera capa de partículas; luego, según las configuraciones ingresadas, generará el resto de partículas buscadas y sus vecindades, provocando como salida un archivo con las posiciones de los átomos y sus vecinos, además de dos imágenes que representan las distintas partículas generadas en base a los archivos de entrada, estas imágenes no son lo suficientemente claras como para poder identificar errores en el diseño esperado, por lo que la posibilidad de equivocarse y simular con una premisa inválida es algo mucho más común que lo esperado.

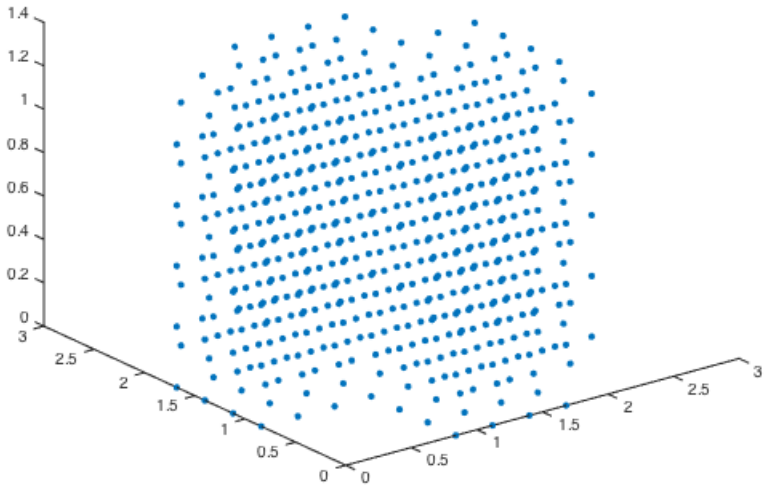

FIGURA 1.4: *Vista 3D de los átomos encontrados*

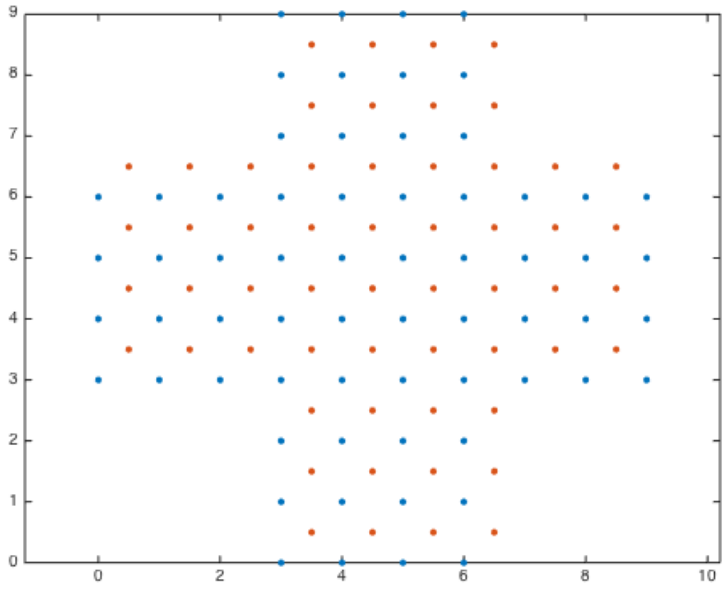

FIGURA 1.5: *Vista 2D de la primera capa de átomos*

Una vez decidido que el objeto diseñado se usará para una simulación es necesario ejecutar un nuevo software, escrito en C, que se encargará de modificar el archivo de salida del script de *Matlab* de tal forma de que pueda ser usado como entrada para la simulación Monte Carlo. Cabe notar que este software está compilado y no se cuentan con el código de fuente de este, de tal forma que si en algún momento se necesitara modificar el formato de la entrada de la simulación este software se debería re-escribir.

```
1        5.4700000e+02    1.4000000e-01    1.5400000e+00    1.2600000e+00    4.0000000e+00
             4.8300000e+02    4.8400000e+02    4.8700000e+02    4.8800000e+02    0.0000000e+00
             0.0000000e+00    0.0000000e+00    0.0000000e+00
```

Formato de salida generado por el script de matlab

```
1        547    0.14    1.54    1.26    4    483    484    487    488    0    0    0    0
```

Formato de entrada necesario para la simulación

Además de modificar el formato de todas las líneas del archivo generado este software agrega una línea en el inicio del archivo que contiene distintos parámetros usados por el software de simulación, como el número de partículas y la constante de escalamiento.

1.3.2 Visualización de resultados

Si bien el proceso de diseño de objetos es precario este no se compara con el de visualización y exportación de resultados, tanto en forma de imágenes como videos. Dentro de las limitaciones que tienen es la imposibilidad de generar videos con colores que permitan distinguir una tercera dimensión no clara al ser una representación en 2D.

Actualmente se usa la funcionalidad *Quiver plot* de matlab, que permite dibujar rápidamente vectores en base a archivos de entrada con cierto formato, no obstante este proceso requiere de tiempo para obtener representaciones que realmente reflejen lo deseado, siendo un proceso de ensayo y error que gasta tiempo de manera innecesaria para los científicos.

1.4 PROPÓSITOS DE LA SOLUCIÓN

1. Mejorar el proceso de preparación de la simulación.
2. Mejorar el proceso de exportación de imágenes para publicaciones.

1.5 ALCANCES O LIMITACIONES DE LA SOLUCIÓN

- El software se encargará del diseño de objetos para la simulación entregando la entrada para ésta y posteriormente de la visualización de los resultados, y de la exportación de estos para publicaciones, mas **NO** se encargará de la simulación en sí y queda fuera del alcance de la solución.
- La aplicación estará disponible para sistema operativo MAC OS X.
- El diseño de objeto será por capas, es decir, se define la “vista superior” y la cantidad de veces que se repetirá hacia abajo.

1.6 OBJETIVOS Y ALCANCE DEL PROYECTO

1.6.1 Objetivo general

Crear un software que permita diseñar objetos en 3D, con características físicas y geométricas específicas, sobre los cuáles se aplicarán simulaciones de campo magnético a nivel atómico y analizar los resultados de forma visual, permitiendo la exportación de imágenes para publicaciones.

1.6.2 Objetivos específicos

Para la consecución del objetivo general, se plantean las siguientes metas intermedias para el software:

1. Modelar el proceso de simulación del efecto de campo magnético en átomos efectuado por los científicos del Departamento de Física de la Universidad de Santiago de Chile.
2. Diseñar la herramienta informática de ayuda para la investigación antes mencionada.
3. Crear el software.
4. Validar el cumplimiento de los requerimientos.

1.7 CARACTERÍSTICAS DE LA SOLUCIÓN

Como solución se propone la creación de un software que facilite el trabajo de los científicos. Esta aplicación se divide en dos funcionalidades:

1.7.1 Diseño del objeto sobre el cuál se hará la solución:

El software debe permitir crear visualmente un objeto en 3D, con ciertas características físicas como la distribución cúbica (ver anexo A). Luego de la creación y configuración del objeto la aplicación debe exportar un archivo de texto que sirva como entrada para el software que hará la simulación. Este archivo tiene un formato ya definido y debe describir la posición de cada átomo del objeto.

Este proceso se divide en tres características:

1. Creación de objeto en base a un mapa de *bits* binario, con características pre-definidas, y exportación de archivo de entrada para la simulación.
2. Permitir la entrada de características del objeto, como número de capas y ordenamiento de los átomos.
3. Visualización atómica en 3D del objeto sobre el cuál se hará la simulación.

1.7.2 Visualización de resultados de la simulación:

El software debe tomar la totalidad de archivos de salida de la simulación como entrada y debe ser capaz de mostrar visualmente el estado magnético de cada átomo en un tiempo *t*, también debe ser posible ver la simulación animada a través del tiempo, como un video.

La salida de la visualización serán imágenes en 2D del estado de la simulación en un tiempo *t*, estas imágenes deben tener colores que permitan al lector entender el resultado a pesar de la dimensión faltante, por ejemplo, usando la proyección del vector en uno de los ejes y asignando un color según la intensidad de éste.

Este proceso se divide en tres características:

1. Permitir ver el estado de la simulación en 3D en un tiempo *t*.

2. Permitir ver el cambio de estado de la simulación en 3D de forma animada.
3. Permitir exportar el estado de la simulación en un tiempo *t* en 2D para publicaciones.

1.8 PROPÓSITOS DE LA SOLUCIÓN

1. Mejorar el proceso de preparación de la simulación.
2. Mejorar el proceso de exportación de imágenes para publicaciones.

1.9 ALCANCES O LIMITACIONES DE LA SOLUCIÓN

- El software se encargará del diseño de objetos para la simulación entregando la entrada para ésta y posteriormente de la visualización de los resultados, y de la exportación de estos para publicaciones, mas **NO** se encargará de la simulación en sí y queda fuera del alcance de la solución.
- La aplicación estará disponible para sistema operativo MAC OS X.
- El diseño de objeto será por capas, es decir, se define la “vista superior” y la cantidad de veces que se repetirá hacia abajo.

1.10 METODOLOGÍA Y HERRAMIENTAS UTILIZADAS

1.10.1 Metodología

Dado que se tiene conocimiento de los requerimientos mayores, pero pueden existir detalles al trabajar en un ámbito tan específico como simulaciones físicas, se decidió usar una modificación de Scrum (Beedle et al., 1999).

Scrum está pensado para trabajar en equipos con varios desarrolladores, además de los cargos más de gestión, para esto se tienen 3 roles (Scrum Alliance, 2015):

Product Owner

Es el encargado de maximizar el valor del trabajo el equipo, para esto tiene un alto conocimiento del producto mediante un contacto directo con los *stakeholders* y facilita la comunicación de estos con el equipo de desarrollo, debido a esto es el responsable de decidir qué se va a construir, pero no el cómo. Para este proyecto el *Product Owner* será el profesor Fernando Rannou, quién tiene contacto constante con los *stakeholders* por proyectos paralelos que se están desarrollando.

Scrum Master

Es el líder del equipo de desarrollo y debe tener un buen conocimiento de la metodología scrum, el cual debe traspasar al equipo de desarrolladores. Sus 3 principales tareas son: Guíar al equipo teniendo un conocimiento tanto del producto como de las tecnologías a utilizar, mantener al equipo avanzando eliminando toda dificultad que puedan tener durante el desarrollo, estas dificultades pueden ser tanto internas como externas, y enseñar a metodología *scrum* al equipo. En este caso como solo un desarrollador trabajará en el proyecto y este tiene un gran conocimiento de la metodología gracias a sus años de experiencia laboral usandola, no se usará este rol.

Desarrollador

El desarrollador es el encargado de entregar los incrementales del producto, para eso se basa en la lista de tareas definidas por el *Product Owner* al inicio de un *sprint*. En este proyecto solo trabajará un programador.

Otras modificaciones hechas a la metodología fue modificar alguna de sus ceremonias, cambiando la reunión diaria (*Daily Scrum*) por una semanal, entre el profesor y el desarrollador, las reuniones retrospectivas al finalizar cada *sprint* se unió con la de planificación del siguiente periodo de desarrollo. Además, como es recomendado, se tuvo una reunión con los *stakeholders* luego de cada

sprint.

Estas modificaciones fueron necesarias para poder usar la metodología en un proyecto con solo un desarrollador y optimizando al máximo el tiempo usado en reuniones, debido al poco espacio en las agendas tanto del desarrollador como del *Product Owner*.

1.10.2 Herramientas de desarrollo

1.10.3 Modelado 3D: OpenGL

Se usará OpenGL como biblioteca de modelado 3D, por ser el estado del arte en este ámbito, entre sus ventajas está el ser multi-plataforma, lo que eventualmente permitiría una rápida portación a otro sistema operativo, y el ser la más usada actualmente, lo que permite que tenga una amplia comunidad de usuarios que la soportan y documentan.

1.10.4 Lenguaje de programación: Python

Para el desarrollo se usará el lenguaje de programación Python con la biblioteca wxPython para Intefaz de Usuario. Esta biblioteca tiene soporte para la API OpenGL. Python, al ser un lenguaje multiplataforma, permitiría una rápida portación a otro sistema operativo en el futuro.

1.10.5 Control de versiones: GIT

Para el versionamiento del código se usará GIT, manteniendo un respaldo del repositorio con el código y la documentación en una máquina virtual

con Linux ubicada en Estados Unidos.

1.11 AMBIENTE DE DESARROLLO

Para el desarrollo se usará el siguiente ambiente de desarrollo:

- Computador marca Apple, con una tarjeta gráfica que soporte OpenGL 3.2+ y sistema operativo Mac OS X para el desarrollo.
- Una máquina virtual con Linux, ubicada en Estados Unidos, para mantener un respaldo del código y de la documentación.

CAPÍTULO 2. DOMINIO DE PROBLEMA

2.1 PROBLEMA

Alumnos de doctorado del Departamento de Física de la Universidad de Santiago de Chile se encuentran estudiando los efectos de aplicar ciertos campos magnéticos a nano-estructuras cristalinas, con propiedades físicas definidas, cuyos resultados esperan puedan ser aplicados a la producción de nuevos dispositivos magnéticos o para el grabado de datos magnético de alta densidad.

2.2 ESTRUCTURAS CRISTALINAS

Las nano-estructuras estudiadas son de tipo cristalinas, las que pueden ser divididas en celdas unitarias de distintos tipos según la distribución de sus partículas. Las 3 estructuras más usadas son el cubo simple (*SC* o *Simple Cubic*), cubo centrado en su cuerpo (*BCC* o *Body-centered Cubic*) y cubo centrado en sus caras (*FCC* o *Face-centered cubic*).

En las estructuras cristalinas la dimensión física de las aristas está dada por el parámetro llamado *Lattice Parameter* o Parámetro de Red, como en este caso solo se trabajará con estructuras cúbicas las aristas tendrán todas la misma dimensión, por lo que nos referiremos a este parámetro como *Lattice Constant* o Constante de Red.

Para cada átomo de una estructura cristalina se puede identificar un conjunto de átomos que compondrán la vecindad, es decir, son las partículas que están a menor distancia de este. Dependiendo del tipo de celdas unitarias se define el algoritmo para encontrarlos y se identifica el número máximo de partículas que pueden ser encontradas.

Dependiendo del tipo de estructura se pueden encontrar patrones de

periodicidad entre ellas, es decir, que cada cierto número de capas se repita la misma distribución de átomos.

2.2.1 Cubo simple (SC)

Esta es la estructura cristalina más simple, donde en cada vértice podemos encontrar una partícula, debido a esto la distancia entre dos átomos vecinos es siempre la constante de red. Como en cada vértice de la estructura se unen 8 celdas unitarias, cada cubo tiene $1/8$ de partícula, luego cada celda tiene $1/8 \cdot 8 = 1$ átomo en total.

En el caso del cubo simple la vecindad de cada átomo está compuesta a lo más por otros 6 átomos, los que están en dirección a cada una de las aristas que componen el vértice donde se encuentra la partícula.

El cubo simple tiene una periodicidad de 1, es decir, todas las capas tienen la misma distribución de átomos.

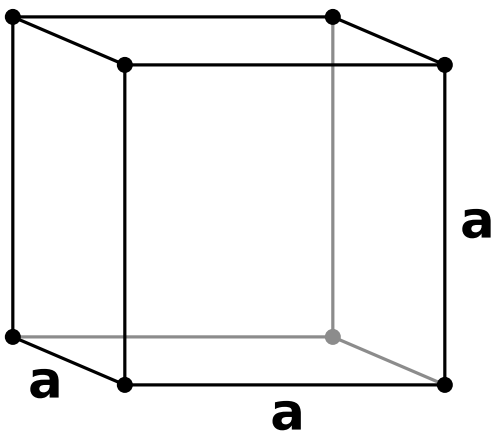

FIGURA 2.1: *Cubo simple*

2.2.2 Cubo centrado en su cuerpo (BCC)

Una estructura cristalina de cubos centrados en su cuerpo se caracteriza porque en cada una de sus celdas unitarias se puede encontrar, además de los átomos en sus vértices, un átomo en su centro, de esta forma cada una de sus celdas tiene $1/8$ de átomo en cada uno de sus vértices más uno en su centro, por lo que tiene $1/8 \cdot 8 + 1 = 2$ átomos en total.

A diferencia de un cubo simple, la vecindad de un átomo está compuesta por:

- Las partículas centrales de cada uno de los cubos que componen un vértice, para el caso de un átomo ubicado en ese punto
- Las partículas ubicadas en cada uno de los vértices, para el caso de un átomo ubicado en el centro de una celda unitaria

En ambos casos el número máximo de partículas en una vecindad es de 8.

La periodicidad de capas de una estructura compuesta por cubos centrados en su cuerpo es de 2, es decir, cada 2 capas se repetirá la distribución de átomos.

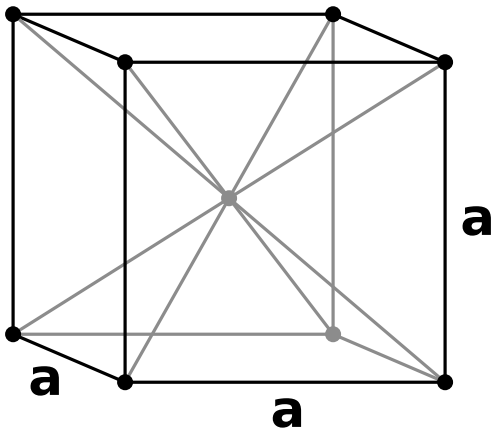

FIGURA 2.2: *Cubo centrado en su cuerpo*

2.2.3 Cubo centrado en sus caras (FCC)

En el caso de las estructuras cristalinas compuesta por cubos centrados en sus caras se pueden distinguir átomos en cada uno de sus vértices y átomos en cada una de las caras de cada cubo. Como el átomo de una caras es compartido por 2 cubos, cada uno de ellos contiene $1/2$ de este, por lo tanto cada celda unitaria contiene $1/8 \cdot 8 + 1/2 \cdot 6 = 4$ átomos en total.

Para la identificación de la vecindad se usa la siguiente regla:

- Las partículas de cada cara que esté compuesta por una de las aristas que conforman el vértice, para el caso de un átomo ubicado en ese punto
- En el caso de las partículas que se encuentran en una cara, todos los átomos ubicados en las caras formados por las aristas que componen la cara del átomo inicial (2 caras por arista), además de los átomos que están en los vértices que componen la cara inicial

De las reglas anteriores se puede inferir que el tamaño máximo de una vecindad es de 12 átomos.

Para las estructuras formadas por cubos centrados en sus caras se identifica una periodicidad de 2, es decir, al igual que en el caso de los cubos centrados en su cuerpo, cada 2 capas se repite a distribución atómica.

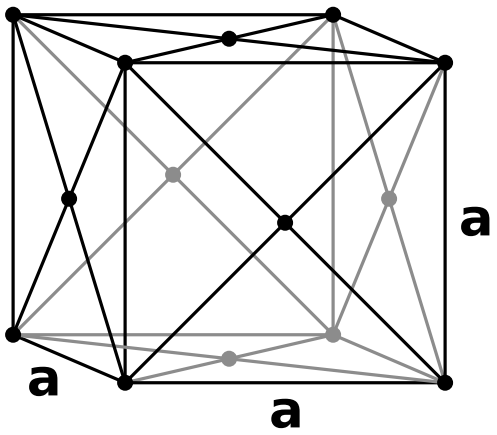

FIGURA 2.3: *Cubo centrado en sus caras*

2.3 ESCALAMIENTO

A pesar de la gran capacidad de procesamiento de los computadores actuales aún resulta inconveniente simular los sistemas con sus características reales, debido a la gran cantidad de átomos que deberían ser simulados. Para esto los científicos deben primero escalar el sistema, usando un sistema propuesto en *Scaling relations for magnetic nanoparticles* (Landeros et al., 2005) donde analizaron un diagrama de fase magnético para un cilindro con ciertas características materiales y físicas, en función de su diámetro y altura, con esto notaron que el diagrama $J' = J \cdot X$, con la constante de escalamiento $X < 1$ se puede obtener modificando las dimenssones del cilindro en base a la siguiente relación:

$$Longitud' = Longitud \cdot X^{\eta}, \text{ con } \eta = 0.55$$

Con esta relación de escalamiento las relaciones magnéticas entre los átomos se mantienen intactas.

En el caso de la solución creada en esta memoria se recibe como entrada de parte del usuario el tamaño original del objeto a simular, la cantidad de celdas unitarias que componen el objeto escalado, la constante de red y la constante η, de tal forma que la constante de escalamiento X se obtiene mediante:

$$X = \sqrt[\eta]{(Altura'/Altura)}$$

$$Altura' = \frac{Capas * Constante\ de\ Red}{Periodicidad\ de\ la\ estructura\ cristalina}$$

2.4 MÉTODO MONTE CARLO

El Método Monte Carlo es un método estadístico (no determinista) propuesto por Nicholas Metropolis et al en 1949, mientras trabajaba en el desarrollo de la Bomba Atómica en el Laboratorio nacional de Los Alamos en EEUU, que permite obtener soluciones aproximadas a problemas muy difíciles de solucionar de forma matemática (determinista) debido a su magnitud o complejidad, para esto se basa en la aleatoriedad. Primero se debe determinar que condiciones se deben cumplir para que cierto caso a probar sea válido, estos pueden ser, por ejemplo, un set de ecuaciones con múltiples parámetros o resultados de interacciones entre un cierto grupo de partículas, luego genera sets de prueba aleatorios que pueden cumplir o no las condiciones definidas, luego de una gran cantidad de iteraciones se pueden usar los resultados válidos para obtener conclusiones. De cierta forma se experimenta teóricamente las distintas combinaciones de factores que afectan un problema definido.

Dependiendo de donde se desea aplicar el método es posible que no sea necesario que los números generados para las pruebas no sean realmente aleatorios, si no que, en general, solo se necesita que sean lo suficientemente aleatorios, para esto existen diversas pruebas estadísticas con las que se puede validar, como que con una gran cantidad de elementos generados no exista un patrón claro de generación y que estos estén distribuidos de forma uniforme.

2.5 APLICACIÓN DEL MÉTODO MONTE CARLO AL PROBLEMA

Para esta investigación los científicos aplicarán el Método Monte Carlo usando las siguientes condiciones (Allende, 2008):

- Se selecciona una partícula i al azar y se calcula su energía $E_1 = E(\vec{m}_i)$
- Se modifica la dirección del momento magnético de la partícula i al azar y se calcula la nueva energía $E_2 = E(\vec{m}_i')$

- Se considera válido el cambio de $\vec{m}_i$ a $\vec{m}'_i$ si se cumple alguna de las siguientes condiciones
  - E_2 es menor que E_1
  - E_2 es mayor que E_1 y se cumple con la relación $\exp(-\beta(E_2 - E_1)) > \varepsilon$, donde $\beta = (kT)^{-}1$ siendo k la constante de Boltzmann, T la temperatura (Newman & Barkema, 1999) y ε un número aleatorio entre 0 y 1

En caso de que las condiciones no se satisfagan se considera inválido el cambio y se rechaza, es decir, se conserva $\vec{m}_i$.

Analizaremos los parámetros de simulación de uno de los casos estudiado por los científicos usando el Método Monte Carlo (Vargas et al., 2011), para esto trabajaremos con un *dot* circular, con diámetro d = 80 nm y altura h = 20 nm. Como fue explicado anteriormente resulta prácticamente imposible analizar estos objetos en tamaño real, por lo que deben ser escalados lo suficiente para poder ejecutar los cálculos usando la tecnología disponible actualmente sin perder sus características magnéticas como el desarrollo de *vortex* magnéticos, para esto se usará un factor de escalamiento $X = 0.01 - 0.001$, esto se logra usando $\eta \approx 0.55 - 0.57$ y por supuesto escalando las dimensiones iniciales de forma $d' = dx^{\eta}$ y $h' = hx^{\eta}$.

Para elegir la nueva orientación del campo magnético se usará un generador aleatorio con una probabilidad $p = min[1, \exp(-\Delta E/k_B T')]$, donde ΔE es el cambio de energía debido a la reorientación del spin, k_B es la constante de Boltzmann y $T' = Tx$, con $T = 10K$.

Inicialmente se aplica un campo hacia el eje X de magnitud $H = 5.5kOe$ y se analizaran pasos de $\Delta H = 0.1kOe$, es decir, para completar una curva de histéresis son necesarios 110 pasos de ΔH. En cada uno de estos pasos se analizará el campo magnético del *dot*, ejecutando 3.500 pasos Monte Carlo por cada ΔH, por lo que la cantidad de pasos para esta simulación es de $110 \cdot 3500 = 385000$ de forma de completar la curva de histéresis.

En cada uno de estos pasos es necesario hacer un cálculo de energía del *dot*.

Este proceso completo se repite varias veces con distintas semillas para el generador aleatorio de números, para así validar el resultado.

2.6 CÁLCULO DE ENERGÍA

Para calcular la energía total E_{tot} de un *dot* se usa la siguiente ecuación:

$$E_{tot} = \frac{1}{2}\sum_{i \neq j}(E_{ij} - J_{ij}\hat{\mu}_i \cdot \hat{\mu}_j) + E_H$$

donde E_{ij} es la energía dipolar dada por

$$E_{ij} = [\vec{\mu}_i \cdot \hat{\mu}_j - 3(\vec{\mu}_i \cdot \hat{n}_{ij})(\vec{\mu}_j \cdot \hat{n}_{ij})]/r_{ij}^3$$

con r_{ij} la distancia entre los momentos magnéticos $\vec{\mu}_i$ y $\vec{\mu}_j$ y $\hat{n}_{ij}$ el vector unitario en la dirección que conecta los dos momentos magnéticos. $\hat{\mu}_i$ es un vector unitario en la dirección de $\vec{\mu}_i$ y $E_H = -\sum_i \vec{\mu}_i \cdot \vec{H}$ representa la energía Zeeman para un campo $\vec{H}$ aplicado hacia el eje x.

J_{ij} es la interacción de canje magnético entre i y j, es decir, como sus campos magnéticos internos se afectan entre ellos, para este caso se asumirá que $J_{ij} = 0$ para dos átomos que no son vecinos. Como en el caso de dos partículas vecinas es necesario calcular esta interacción, al momento de ejecutar la simulación es necesario saber, para cada átomo, quienes componen su vecindad.

CAPÍTULO 3. ARQUITECTURA DE LA SOLUCIÓN

3.1 ARQUITECTURA

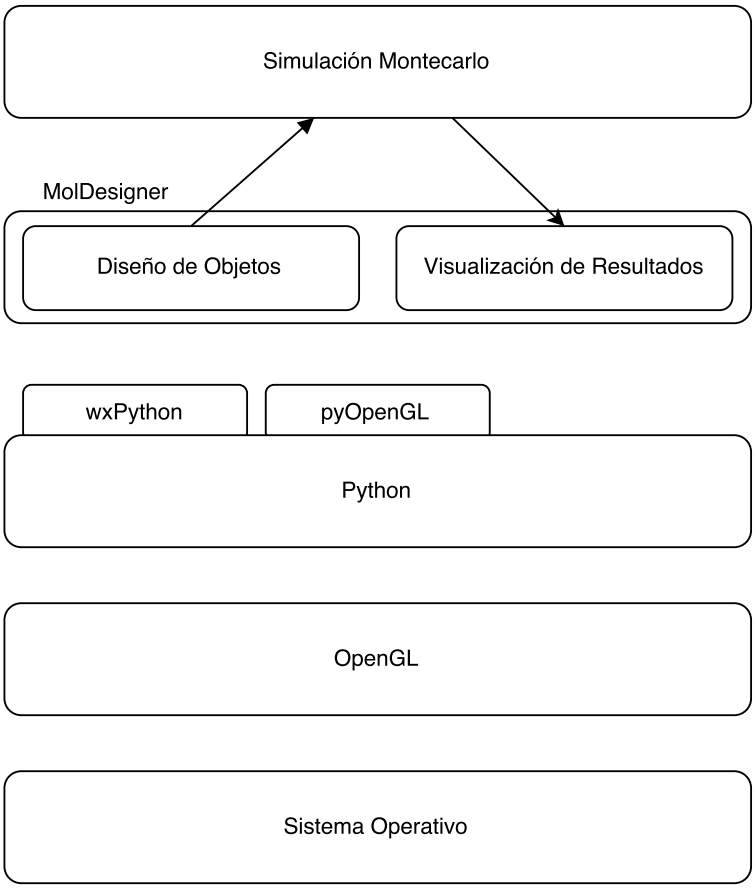

FIGURA 3.1: *Diagrama de la arquitectura de la solución.*

3.1.1 Sistema Operativo

Si bien la aplicación fue desarrollada íntegramente en Mac OS X, y es por lo tanto el sistema operativo oficial de la presente memoria, todo el sistema fue desarrollado pensando en ser eventualmente multi-plataforma, sin que esta portación a distintos Sistemas Operativos tenga mayores complicaciones, más allá de los problemas que uno pueda encontrar por la distinta estructura de estos. Para esto tanto la elección del lenguaje de programación como de las

distintas bibliotecas usadas fueron hechas teniendo el cuenta que deben poder ser ejecutados en los 3 sistemas operativos más usados, OS X, Windows y Linux.

3.1.2 OpenGL

Como parte importante del software es su capacidad de representar gráficamente en 3D tanto los átomos de un objeto diseñado como los resultados de las simulaciones era necesario buscar una biblioteca de procesamiento gráfico 3D que fuera realmente multi-plataforma, tanto a nivel de sistema operativo como de tarjetas gráficas. Debido a esto se eligió el estándar OpenGL, el cuál tiene implementaciones tanto para Windows, OS X y Linux, además de ser el estándar de la industria para gráficos 2D y 3D (The Khronos Group, 2015c), esto permite tener una gran comunidad activa lo que a su vez se traduce en una gran cantidad de documentación al respecto.

La especificación del estándar OpenGL era dirigido por el consorcio independiente *OpenGL Architecture Review Board* hasta el año 2006, cuando se decidió transferir esta responsabilidad al *Khronos Group* (The Khronos Group, 2015b), un consorcio formado por distintas organizaciones, tanto empresariales como académicas, quienes manejan múltiples estándares de la industria como *OpenGL ES*, *OpenCL* y *WebGL* (The Khronos Group, 2015a).

3.1.3 Python

Para el lenguaje de programación se barajó inicialmente la opción de C++ por las ventajas que conlleva trabajar a bajo nivel, no obstante debido a su inclinada curva de aprendizaje y complejidad se decidió usar Python, ya que era un lenguage que también cumple con las características necesarias para este desarrollo, como el ser multi-plataforma, y tener a disposición bibliotecas

de manejo gráfico como su compatibilidad con la API OpenGL, de tal forma de centrar la complejidad del proyecto en las representaciones 3D.

Python es un lenguaje de programación de alto nivel, interpretado y orientado a objetos, desarrollado durante el año 1990 por Guido van Rossum, aunque actualmente es de código abierto y mantenido por su comunidad, la que es liderada por la *Python Software Foundation*, quienes además resguardan los derechos de este.

Algunas de las características más conocidas de Python es su fácil sintaxis y su gran biblioteca estándar, la que cubre áreas como Protocolos de comunicación, Ingeniería de software e Interfaces de Sistema Operativo (Python Software Foundation, 2015).

Durante el desarrollo se la aplicación se usaron distintas bibliotecas, siendo las dos más importantes wxPython y pyOpenGL.

3.1.3.1 wxPython

wxPython es una biblioteca de python que permite usar de forma nativa wxWidgets, un set de herramientas de código abierto, escrita en C++ y inicialmente escrito por Julian Smart y actualmente mantenida por la comunidad que permiten crear interfaces gráficas en distintas plataformas como Windows, OS X, iOS, Linux, entre otros. En la memoria presentado wxPython maneja todas las interacciones de los usuarios, además de casi todas las interfaces gráficas con excepción de los *canvas* de OpenGL.

3.1.3.2 pyOpenGL

pyOpenGL es una biblioteca que permite la programación en OpenGL directamente desde python, es multiplataforma y totalmente compatible con la biblioteca de interfaz gráfica usada (wxPython). Uno de los sub-paquetes de

pyOpenGL usado en esta memoria es GLUT, la que permite crear fácilmente ciertos objetos en OpenGL, como esferas o conos, de tal forma de no tener que programar estos a partir de triángulos, como sería si se usara OpenGL puro.

3.2 MOLDESIGNER

MolDesigner es el software desarrollado en esta memoria, el cual permite a científicos generar objetos sobre los cuales se simulará la aplicación de campos electromagnéticos y la visualización de resultados generados por la simulación. Esta herramienta está pensada para ser usada por estudiantes del doctorado de física de la Universidad de Santiago de Chile, no obstante la aplicación que ejecuta la simulación es usada por diversas organizaciones académicas, por lo que este software pretende ser un aporte a la comunidad científica en general.

Durante el desarrollo se puso especial énfasis en la experiencia de usuario, de tal forma que cualquier científico que tenga acceso a esta aplicación sea capaz de usarlo sin la necesidad de un entrenamiento, por este motivo el software tiene todos sus textos en inglés.

La aplicación tiene dos funcionalidades claramente definidas, el diseño de objetos y la visualización de resultados.

3.2.1 Diseño de objetos

Para el desarrollo de esta funcionalidad se usó como base el sistema de diseño de objetos actual, basado en un mapa de bits, explicado en la sección 1.3.1.2 de esta memoria, pero con la finalidad de que el proceso completo sea ejecutado en la misma aplicación disminuyendo con esto la probabilidad de errores que se puedan producir.

Dentro de las entradas de esta funcionalidad están tanto el mapa de bits, mediante una grilla binaria, como los distintos parámetros físicos asociados al objeto, como el tipo de estructura cristalina, la constante de red y el número de capas, entre otros. Además se le ofrece al usuario mucha información dinámica, es decir, que va cambiando en base a los parámetros de entrada, como por ejemplo la constante de escalamiento.

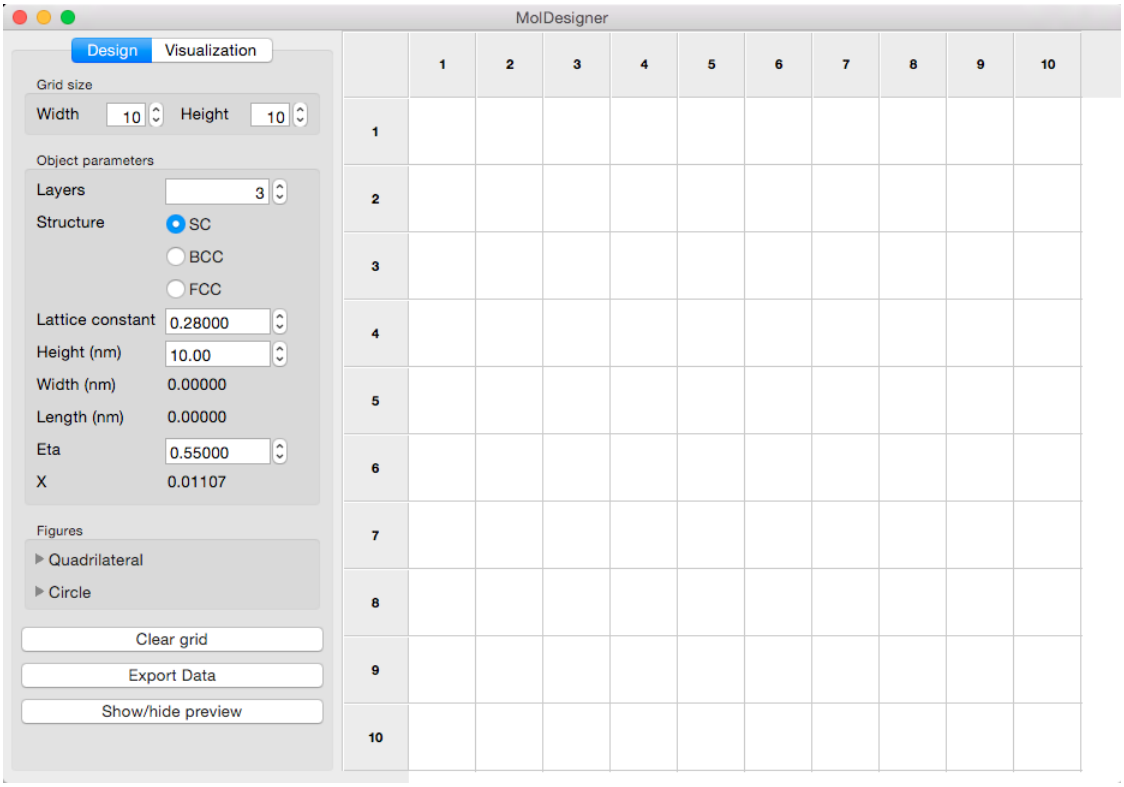

FIGURA 3.2: *Vista de diseño de objetos del software*

Una de las características del diseño es la posibilidad de crear imágenes pre-definidas en base a dimensiones entregadas por el usuario, de esta forma se pueden crear cuadriláteros definiendo su ancho y alto o circunferencias con un radio específico, una vez ingresados los parámetros solo es necesario hacer click en la posición de la grilla donde se quiere dibujar.

El software permite también al usuario ver de forma inmediata la visualización 3D del objeto que se está creando con colores identificando sus distintos tipos de partículas, permitiendo comprobar de manera rápida y fácil si efectivamente el diseño es el deseado.

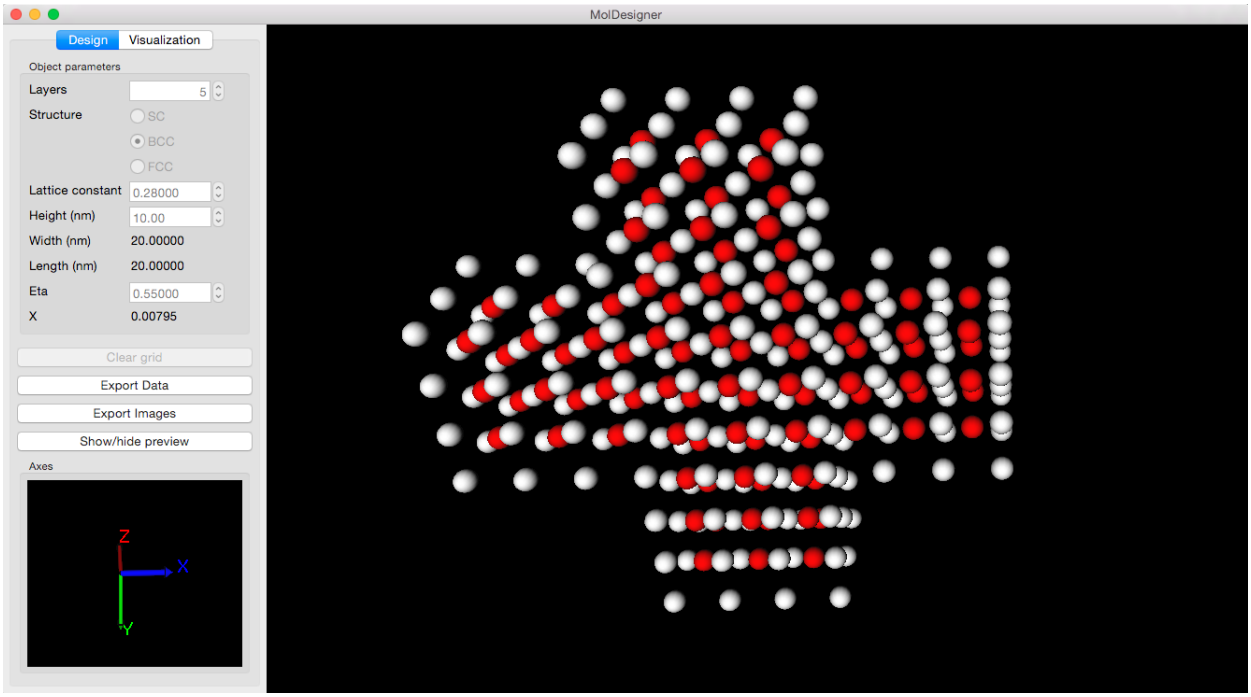

FIGURA 3.3: *Previsualización de un objeto diseñado*

Una vez confirmado que el objeto diseñado es el deseado es posible exportar directamente las imágenes del diseño y los ejes coordenados para referencia, como el archivo que servirá como entrada para el software de simulación. A diferencia del proceso actual, donde el archivo exportado debe ser modificado para poder ser usado, en este caso los datos exportados puedes ser usados inmediatamente para ejecutar una simulación.

3.2.2 Visualización de resultados

La segunda funcionalidad se basa específicamente en los requerimientos de los *stakeholders*, sin tener un proceso actual como base, ya que esta es la gran debilidad que tienen actualmente.

Dentro de las características de esta funcionalidad está el ver y exportar el estado de la simulación en un tiempo t, esto incluye los vectores de campo magnético, la curva de histéresis del campo magnético y los

ejes coordenados para referencia, de forma de poder publicar sus resultados fácilmente.

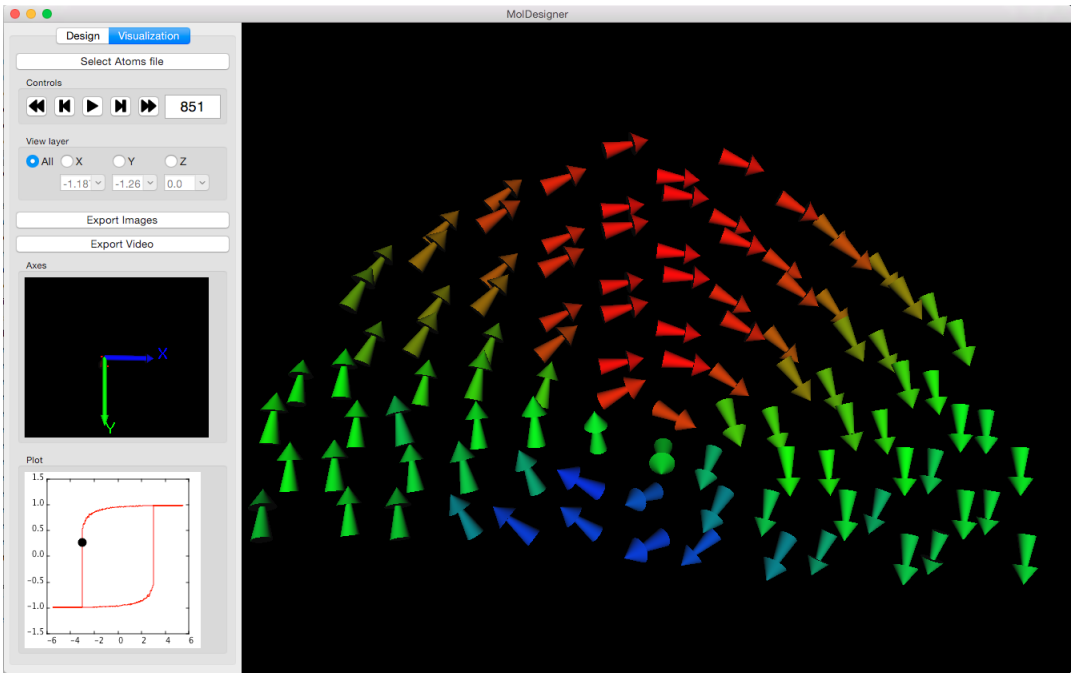

FIGURA 3.4: *Pestaña de visualización de resultados en t = 851*

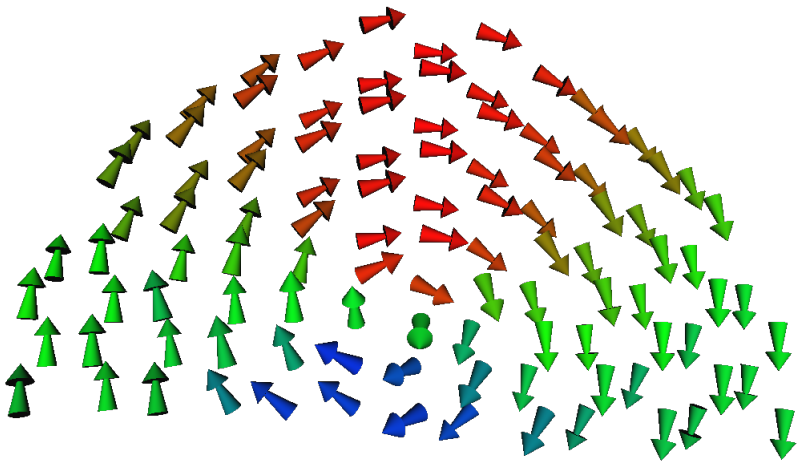

FIGURA 3.5: *Imagen de vectores exportada para publicación*

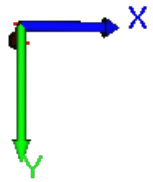

FIGURA 3.6: *Imagen de ejes de referencia exportada para publicación*

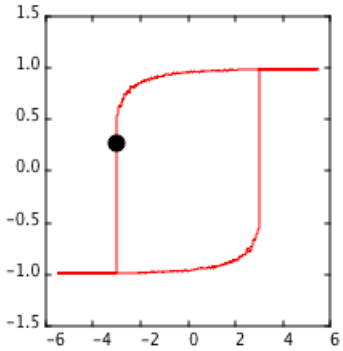

FIGURA 3.7: *Imagen de curva de histéresis exportada para publicación*

Otras características disponibles es la asignación de colores para cada vector según uno de sus componentes, usando una escala de colores de azul a rojo, en el caso de la imagen 3.5 el máximo valor de $\hat{i}$ será rojo y el mínimo será azul, esto permite identificar rápidamente una tercera dimensión en una imagen 2D y también los vortex que se generan.

También es posible ver la variación de los vectores a través del tiempo como video y luego exportarlo de tal forma de que este pueda ser usado fácilmente en conferencias donde se divulguen los resultados.

Cabe notar que todas las características anteriores pueden ser ejecutadas mientras se visualiza solo una de las múltiples capas en la que se puede dividir el objeto, en base a cualquier eje.

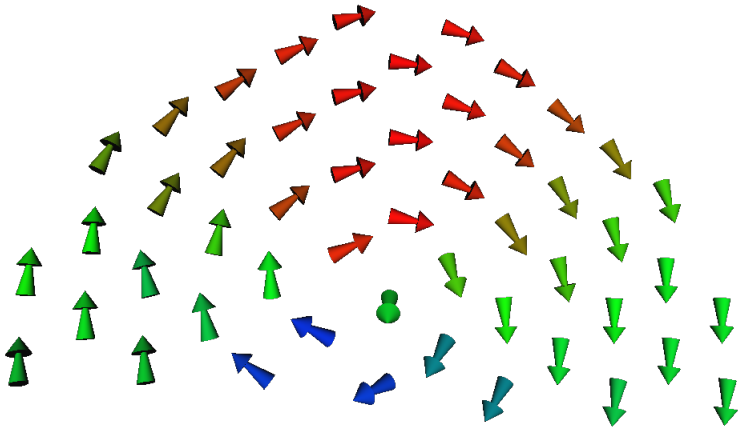

FIGURA 3.8: *Estado de la simulación para t = 851 solo para la capa Z = 0*

FIGURA 3.9: *Estado de la simulación para t = 851 solo para la capa X = -0.395*

3.3 CLASES

Para el desarrollo de la aplicación se usó el lenguaje Python, dividiendo la aplicación en clases que se pueden categorizar de la siguiente forma:

3.3.1 Clases visuales

3.3.1.1 MolDesigner

Esta es la clase principal del programa, ya que es la encargada de iniciar el programa, cargando todas las dependencias necesarias para su ejecución, pero principalmente tiene la lógica de la parte visual del *software*, es decir, de la creación y distribución en pantalla de los distintos elementos visuales, como ventanas, botones, campos de texto, etc., además de la interacción del usuario con estos. Cada reacción a una acción ejecutada sobre estos elementos es orquestada por esta clase.

3.3.1.2 BitmapGrid

La clase BitmapGrid es la encargada de manejar la grilla con la cuales los científicos diseñarán los objetos sobre los cuales se simulará, para esto se basa en la biblioteca *wx.grid* de *wxPython*, una implementación de una planilla de cálculos tipo *excel*, la cual es modificada para no poder ser editada y que las distintas acciones del *mouse* sobre esta (click, seleccionar fila o columna, seleccionar un rango de celdas, etc.) generen cambios en el color de fondo de cada celda, pudiendo este ser negro o blanco, transformando esta planilla de cálculos en un mapa de bits binario.

Entre las características de este mapa de bits se encuentra la capacidad de crear figuras predefinidas rápidamente, por ejemplo, se puede dibujar un cuadrilátero indicando el ancho, el alto y seleccionando la esquina superior derecha de esta figura. También es posible crear un círculo indicando el radio que tendrá este y su centro.

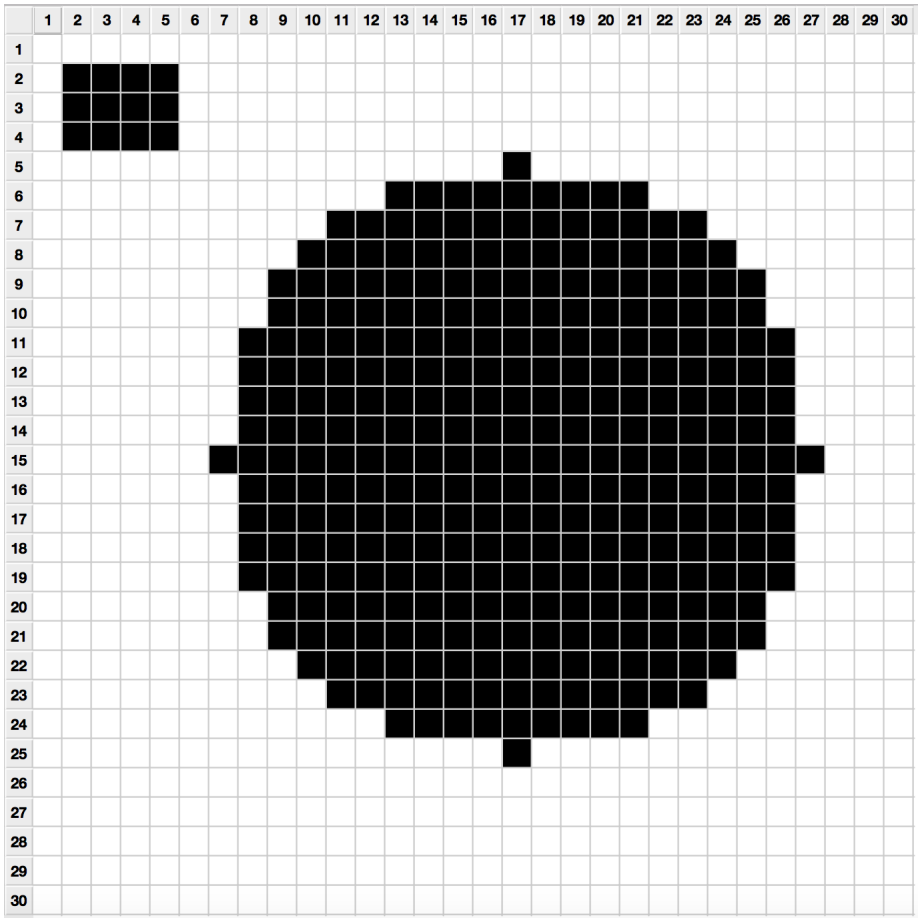

FIGURA 3.10: *Mapa de bits binario con 2 figuras pre-diseñadas.*

3.3.2 Clases 3D

Estas clases son las encargadas de manejar los distintos *canvas* que se usan en el *software*, tanto para la visualización del diseño y del resultado de la simulación, como para ayudas referenciales para los científicos.

3.3.2.1 AtomCanvas

La clase AtomCanvas es la más importante con respecto a la visualización 3D, ya que es la encargada de mostrar en pantalla tanto el diseño de los objetos sobre los cuales se correrá la simulación como los resultados de estas usando OpenGL. En el desarrollo de esta se puso énfasis en la optimización, pudiendo mostrar sin mayores demoras más de 20.000 átomos, para tener una idea un sistema promedio analizado por los científicos usa 3.000 átomos (TODO: CONFIRMAR).

Otra de las tareas de esta clase es manejar las distintas interacciones del usuario, tanto con el teclado como con el *mouse* con las representaciones en 3D, como rotaciones, movimientos y *zoom*.

Para la visualización del diseño se usan esferas de distintos colores, representando cada uno de los distintos tipos de átomos que pueden haber según la estructura cúbica elegida. En las siguientes imágenes se representan una estructura de 5x5, con 3 capas, siendo solo diferente la estructura cúbica elegida.

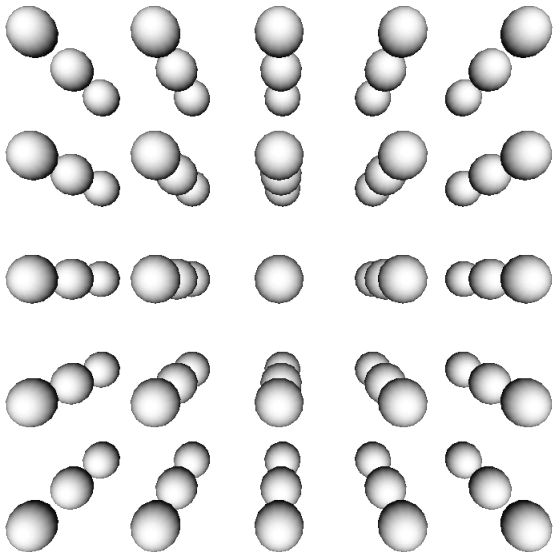

FIGURA 3.11: *En un SC todos los átomos son blancos.*

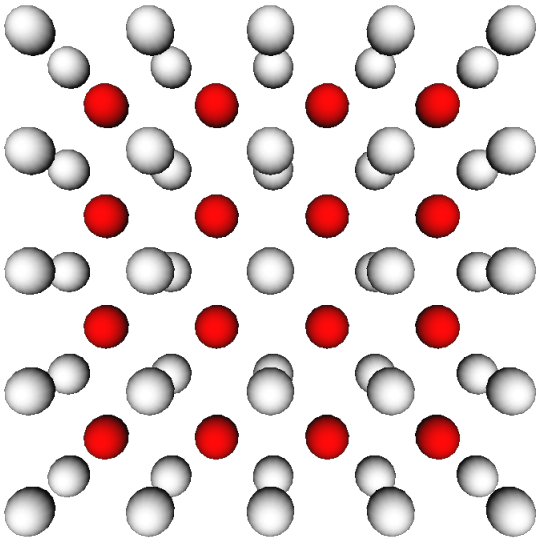

FIGURA 3.12: *En un BCC los átomos centrales son rojos.*

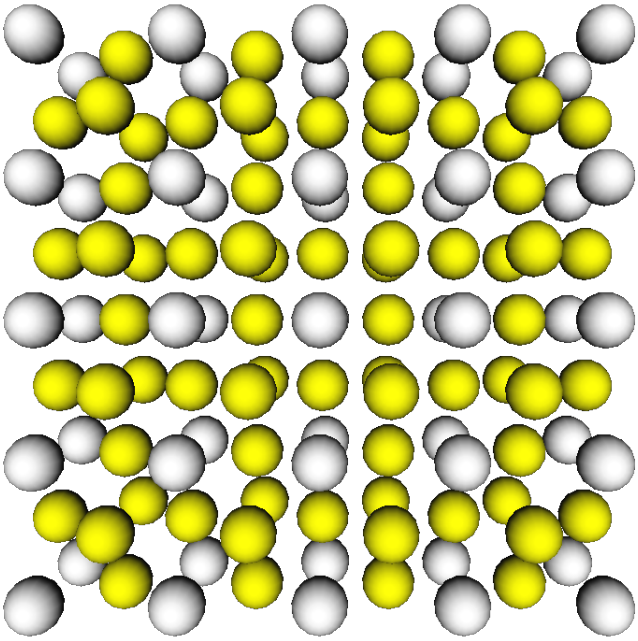

FIGURA 3.13: *En un FCC los átomos de las caras son amarillos.*

En el caso de la visualización de resultados se usan flechas de colores como representación. Para asignar un color a una flecha se parte de la premisa que en tiempo t=0 el campo magnético está cargado hacia un eje, es decir, todos los vectores serán iguales, teniendo la misma magnitud, sentido y dirección, estando este paralelo a uno de los ejes del plano coordenado; además se sabe

que en ese momento su magnitud es máxima. Si inicialmente todos los vectores son paralelos al eje A, se usará la componente â de cada vector para definir el color, si la componente es 0 será de color verde, si la magnitud es máxima en sentido contrario a los vectores iniciales el color será azul, si la magnitud es máxima en el mismo sentido de los vectores iniciales el color será rojo. Como se puede inferir la escala va desde azul a rojo, siendo este último color el inicial para todos los vectores.

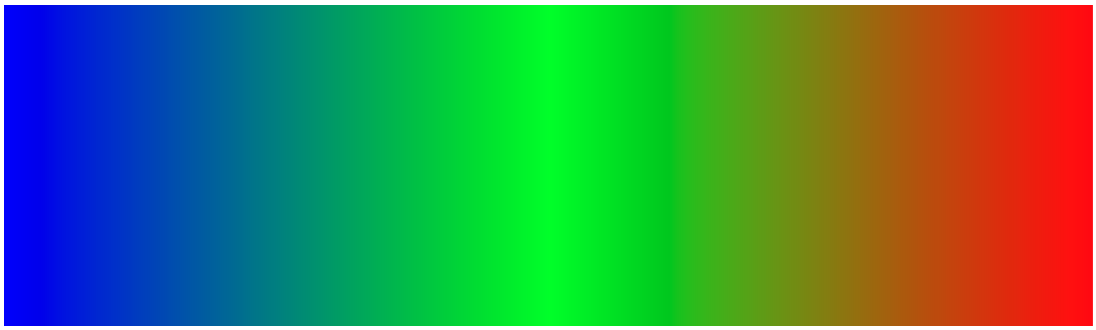

FIGURA 3.14: *Escala de colores de azul a rojo.*

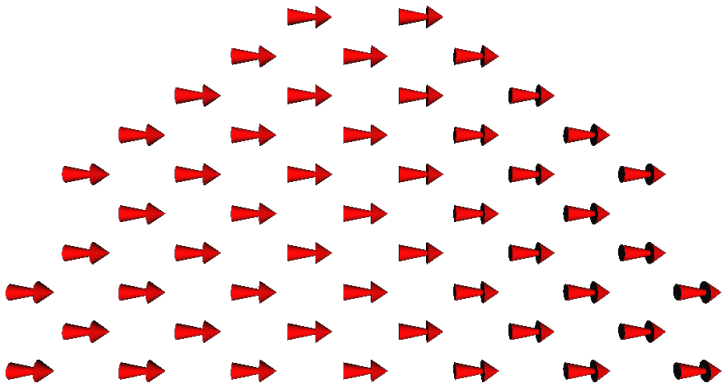

FIGURA 3.15: *Estado inicial de la visualización, con todos los vectores rojos cargados al eje X.*

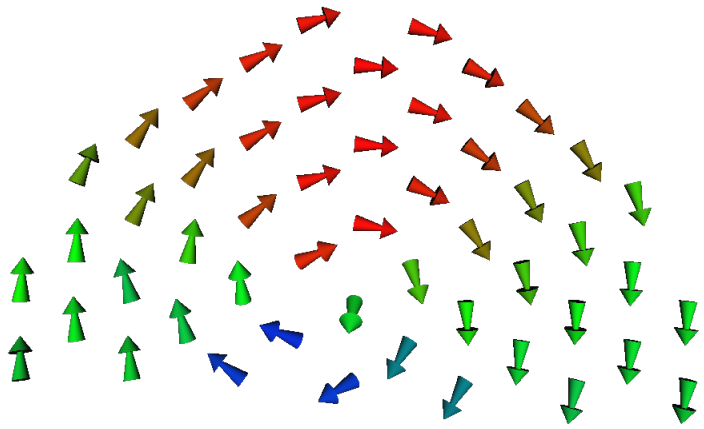

FIGURA 3.16: *Vectores de distintos colores según la componente* $\hat{i}$.

3.3.2.2 Axes

La clase Axes es la encargada de mostrar los ejes coordenados de los distintos canvas, tanto del de diseño de objetos como el de visualización de resultados, usando Open GL. Cada eje se representa con su propio color, usando azul, rojo y verde para los ejes X, Y y Z respectivamente, y una etiqueta con el mismo color, de forma de hacerlo fácil de visualizar para el usuario. Debido al diseño del *software*, donde las distintas funciones del programa (diseño y visualización) se seleccionan mediante pestañas, esta clase debe ser instanciada 2 veces, ya que no es posible usar la misma instancia en ambas secciones. Estas se comunican directamente con AtomCanvas para obtener los distintos parámetros de rotación de forma que los ejes sean coherentes a la imagen mostrada.

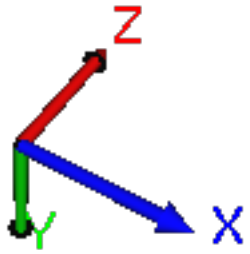

FIGURA 3.17: *Representación visual de los ejes coordenados.*

3.3.3 Clases de cubos

Las clases de cubos son 3 clases hermanas que calculan los átomos de un objeto que se está diseñando, todas tienen solo 2 métodos: *calculate* y *find_neighborhood*, el primero se encarga de identificar todos los átomos que aplican dados los parámetros físicos como la estructura de la primera capa, luego el segundo busca, para cada átomo, todos sus vecinos inmediatos, estos parámetros son necesarios para exportar el archivo que luego servirá de entrada para la simulación.

3.3.3.1 SC

SC es la clase que maneja los cubos simples (*Simple Cubic* o *SC*), estas estructuras cúbicas se caracterizan por tener un átomo en cada uno de sus vértices, por lo que el cálculo de sus átomos se reduce a simplemente repetir la capa superior tantas veces como sea indicado en la entrada de propiedades

físicas. Para encontrar el vecindario es necesario buscar todos los átomos que estén en las siguientes posiciones relativas [-1,0,0], [1,0,0], [0,-1,0], [0,1,0], [0,0,-1] y [0,0,1], por lo que el tamaño máximo de su vecindad es de 6 átomos.

3.3.3.2 BCC

BCC es la clase que maneja los cubos centrados en el cuerpo (*Body Centered Cubic* o *BCC*), que son las estructuras cúbicas que además de tener un átomo en cada vértice de los cubos tienen uno en el centro de cada uno de estos, de tal forma que en el cálculo de átomos se debe trabajar con una capa intermedia que contendrá los centros de cada cubo, de tal forma que para una estructura de 5 capas quedaría así:

#	Capa
1	Primaria
2	Intermedia
3	Primaria
4	Intermedia
5	Primaria

La regla para agregar un átomo central es que debe tener un cubo de átomos a su alrededor, en caso de que el cubo no esté completo simplemente se usarán las capas primarias:

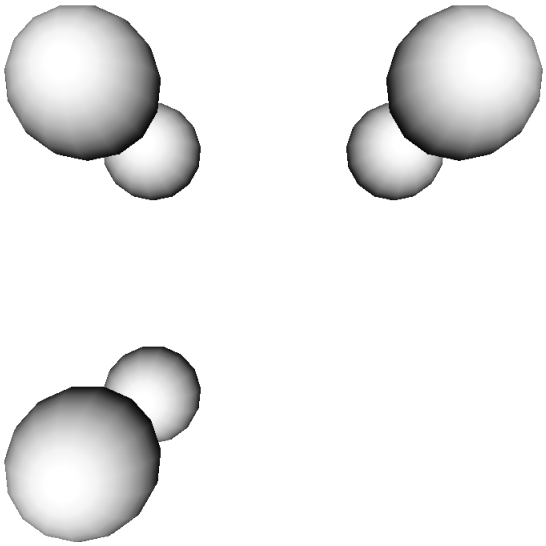

FIGURA 3.18: *Cubo BCC incompleto, sin átomo central*

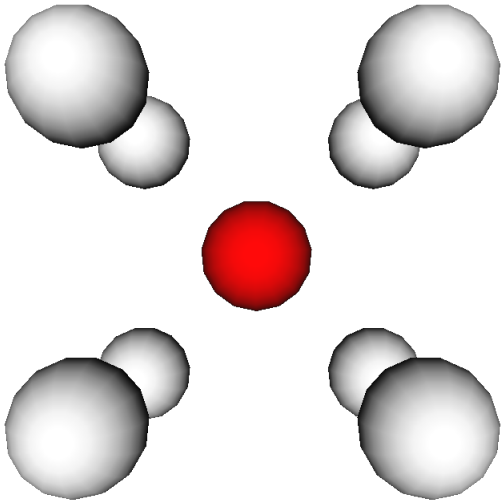

FIGURA 3.19: *Cubo BCC completo, con átomo central*

En el caso de los BCC los átomos que conforman la vecindad siempre estarán en las posiciones relativas $[\pm 0.5, \pm 0.5, \pm 0.5]$, es decir, cada átomo puede tener una vecindad compuesta por hasta 8 átomos.

3.3.3.3 FCC

FCC es la clase que maneja los cubos centrados en las caras (*Face Centered Cubic* o *FCC*), los cuales se caracterizan por tener un átomo extra por cara además de uno en cada uno de sus vértices, por lo que además de tener que crear una capa intermedia es necesario modificar la capa primaria, es decir, la que crea el usuario usando el mapa de bits. La regla para agregar estos átomos en las caras es que esté en la diagonal creada por otros 2 átomos, en cualquier dirección. En la siguiente imagen se ve una estructura cúbica de 1x2, como en una de sus caras se forma una diagonal entre 2 átomos se agrega uno extra en una capa intermedia.

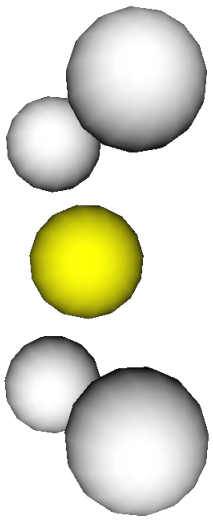

FIGURA 3.20: *Estructura cúbica FCC, con átomo en una de sus caras*

La vecindad de estas estructuras cúbicas está dada por la posición relativa dada por $[\alpha, \beta, \gamma]$, donde:

$$(\alpha = \pm 0.5; \beta = \pm 0.5; \gamma = 0) \vee (\alpha = \pm 0.5; \beta = 0; \gamma = \pm 0.5) \vee (\alpha = 0; \beta = \pm 0.5; \gamma = \pm 0.5)$$

Lo que en su combinatoria resulta 12 posiciones, siendo este el número máximo de átomos en una vecindad.

REFERENCIAS

Allende, S. (2008). *Propiedades magnéticas de los nanohilos de niquel*. Ph.D. thesis, Universidad de Santiago de Chile.

Beedle, M., Devos, M., Sharon, Y., Schwaber, K., & Sutherland, J. (1999). Scrum - a pattern language for software development. *Pattern Languages of Program Design*, *4*, 637–651.

Landeros, P., Escrig, J., Altbir, D., Laroze, D., d'Albuquerque e Castro, J., & Vargas, P. (2005). Scaling relations for magnetic nanoparticles. *Phys. Rev. B*, *71*, 094435.
URL http://link.aps.org/doi/10.1103/PhysRevB.71.094435

Metropolis, N., & Ulam, S. (1949). The Monte Carlo Method. *Journal of the American Statistical Association*, *44*(247), 335–341.

Newman, M. E. J., & Barkema, G. T. (1999). *Monte Carlo methods in statistical physics*. Oxford: Clarendon Press.

Python Software Foundation (2015). General python faq¶.
URL https://docs.python.org/3/faq/general.html#what-is-python-good-for

Scrum Alliance (2015). Core scrum.
URL https://www.scrumalliance.org/why-scrum/core-scrum-values-roles

The Khronos Group (2015a). About the khronos group.
URL https://www.khronos.org/about/

The Khronos Group (2015b). About the opengl arb 'architecture review board'.
URL https://www.opengl.org/archives/about/arb/

The Khronos Group (2015c). Opengl overview.
URL https://www.opengl.org/about/#1

Vargas, N. M., Allende, S., Leighton, B., Escrig, J., Mejía-López, J., Altbir, D., & Schuller, I. K. (2011). Asymmetric magnetic dots: A way to control magnetic

properties. *Journal of Applied Physics*, *109*(7), –.

URL `http://scitation.aip.org/content/aip/journal/jap/109/7/10.1063/1.3561483`