UNIVERSIDAD DE QUINTANA ROO
DIVISIÓN DE CIENCIAS E INGENIERÍA

TESIS

RECONSTRUCCIÓN DE MATERIALES ESTOCÁSTICOS PARA APLICACIONES DE ENERGÍA

TESIS
PARA OBTENER EL GRADO DE

MAESTRO EN MECATRÓNICA

PRESENTA
LIC. ABRAHAM MISRAIM RIOS CANO

DIRECTOR
DR. JAIME SILVERIO ORTEGÓN AGUILAR

ASESORES
DR. GLISERIO ROMELI BARBOSA POOL
DR. JAVIER VÁZQUEZ CASTILLO
DR. ABIMAEL RODRÍGUEZ SÁNCHEZ
DRA. BEATRIZ ESCOBAR MORALES

CONACYT
Consejo Nacional de Ciencia y Tecnología

CHETUMAL QUINTANA ROO, MÉXICO, FEBRERO DE 2019

UNIVERSIDAD DE QUINTANA ROO
DIVISIÓN DE CIENCIAS E INGENIERÍA

TRABAJO DE TESIS BAJO LA SUPERVISIÓN DEL COMITÉ DEL PROGRAMA DE MAESTRÍA Y APROBADA COMO REQUISITO PARA OBTENER EL GRADO DE:

MAESTRO EN MECATRÓNICA

COMITÉ DE TESIS

DIRECTOR: _____
DR. JAIME SILVERIO ORTEGÓN AGUILAR

ASESOR: _____
DR. GLISERIO ROMELI BARBOSA POOL

ASESOR: _____
DR. JAVIER VÁZQUEZ CASTILLO

ASESOR: _____
DR. ABIMAEL RODRÍGUEZ SÁNCHEZ

ASESOR: _____
DRA. BEATRIZ ESCOBAR MORALES

CHETUMAL, QUINTANA ROO, MÉXICO, FEBRERO DE 2019.

Agradecimientos

Al Consejo Nacional de Ciencia y Tecnología (CONACyT), por el patrocinio económico brindado a lo largo de 2 años.

A la Universidad de Quintana Roo, por ser la institución de alta calidad que me permitió desarrollar mis aptitudes escolares y científicas.

Al director de la tesis, el Dr. Jaime Silverio Ortegón Aguilar, por la acertada orientación y el conocimiento invaluable que me compartió, pues me abrieron paso a entender cada detalle de la tesis, y por su paciencia para no dejar de confiar en la culminación de este trabajo.

A los asesores de la tesis, el Dr. Gliserio Romeli Barbosa Pool, el Dr. Javier Vázquez Castillo, el Dr. Abimael Rodríguez Sánchez y la Dra. Beatriz Escobar Morales, por sus valiosas contribuciones y fungir como jurado en la defensa de mi tesis.

A mis padres, por ser aquellas grandes personas que me inspiran, por su apoyo incondicional, por la motivación constante, por ser el ejemplo de perseverancia y disciplina que guía mi vida, y sobre todo, por su inigualable amor.

A mis demás profesores y compañeros de aula, que estuvieron siempre guiando mi aprendizaje con mucha dedicación y pasión a lo largo de la maestría.

A mis amigos y allegados, que siempre tuvieron palabras de aliento acerca de mi vocación para la ciencia.

Resumen

La reconstrucción de materiales estocásticos permite el estudio de las propiedades de los materiales a partir de imágenes bidimensionales de los mismos. Diversos autores han presentado metodologías e implementaciones para la reconstrucción de materiales, pero la eficiencia y el consumo de tiempo y recursos todavía es objeto de estudio por parte de la comunidad científica. Los distintos elementos de los materiales se dividen en fases para su estudio, en esta tesis se consideran, únicamente, dos fases: sólido y vacío. El objetivo consiste en la creación de una herramienta informática que reconstruya de forma tridimensional materiales estocásticos, utilizados en aplicaciones de energía, a partir de imágenes bidimensionales. Se inició con la clasificación de los píxeles de una imagen, generada por un microscopio electrónico de barrido o SEM (por sus siglas en inglés: "Scanning Electron Microscope"), en las mencionadas fases. Se empleó un conjunto de 3 funciones de correlación para caracterizar la morfología del material a partir de una imagen de referencia: tamaño de poro (TP), dos puntos (DP) y camino lineal (CL). El proceso de reconstrucción parte de un volumen generado con esferas basadas en TP; continuando con un proceso de recocido simulado para minimizar la diferencia de DP y CL de cada fase, con respecto a las obtenidas de las imágenes 2D. Una de las ventajas del sistema es el proceso de diezmado, previo a la reconstrucción, que reduce la imagen preservando el 98 % de la información contenida en las funciones de correlación. Adicionalmente, la temperatura mínima, considerada para el recocido simulado, es establecida en función del tamaño de la imagen después del diezmado. En cada iteración del algoritmo de optimización se recalcula el error general mediante el intercambio de dos píxeles de fases distintas. La actualización de las funciones de correlación se hace únicamente para las filas, columnas, y capas de vóxeles que son afectadas por el intercambio. La implementación del diezmado, la actualización de las funciones y la selección de la temperatura mínima lograron una reducción en el tiempo de procesamiento de hasta un 99 %, y también la generación de un volumen, a partir de una imagen, manteniendo una precisión del 98 % con respecto a la imagen original.

Tabla de Contenidos

Agradecimientos **3**

Resumen **4**

1. Introducción **7**
1.1. Estado del Arte 8
1.2. Planteamiento del Problema 15
1.3. Objetivo General 15
1.4. Objetivos Particulares 15
1.5. Alcance 16
1.6. Justificación 16
1.7. Estructura de la Tesis 16

2. Microestructuras de Materiales Estocásticos y Descriptores **17**
2.1. Microscopía de Materiales 18
2.1.1. Microscopio Electrónico de Barrido 18
2.2. Identificación de Fases 20
2.3. Binarización de Imágenes Digitales 21
2.4. Funciones de Correlación 22
2.4.1. Función de Dos Puntos 22
2.4.2. Función de Camino Lineal 23
2.4.3. Función por Tamaño de Poro 23
2.5. Recocido Simulado 24

3. Marco de Trabajo de Software **25**
3.1. Lenguaje de Programación C# 26
3.2. Librería Gráfica GTK+ 26
3.3. Generador de Interfaces Xamarin Studio 27

3.4. Metodología de Desarrollo de Software 28

4. Algoritmos para la Reconstrucción 29
4.1. Binarizado Digital 30
4.2. Funciones de Correlación: Camino Lineal y Dos Puntos . 31
4.3. Función de Correlación: Tamaño de Poro 33
4.4. Reconstrucción con Recocido Simulado 36

5. Algoritmos para Acelerar la Reconstrucción 41
5.1. Diezmado 42
5.2. Generación del Volumen Inicial 45

6. Pruebas y Resultados del Sistema 49
6.1. Pruebas de Funcionamiento y Desempeño 50
6.1.1. Metodología de Aplicación de Pruebas 51
6.2. Resultados 52
6.2.1. Electrodo de Platino sobre Carbón (9kX_CCM3) . 52
6.2.2. Electrodo de Titanio Puro (800X_C2H2O4) 58
6.2.3. Electrodo de Platino sobre Carbón (15kX_HP1) . . 64
6.2.4. Electrodo de Platino sobre Carbón (15kX_HP3) . . 70
6.3. Discusión de Resultados 76

7. Conclusiones y Trabajo Futuro 77
7.1. Conclusiones 78
7.2. Trabajo Futuro 79

Bibliografía 80

Anexo A: Código del Sistema de Reconstrucción 87

Capítulo 1

Introducción

Los materiales heterogéneos se encuentran presentes en prácticamente todo el entorno físico que rodea al planeta, y se pueden encontrar coexistiendo tanto en la naturaleza como en los materiales artificiales. Su gran presencia toma interés científico gracias a las distintas aplicaciones que tienen en la ingeniería, tales como: suelos, concretos y electrodos de celdas de combustible. Algunas de estas aplicaciones, como las celdas de combustible, son tecnologías en desarrollo que cuentan con el potencial para ser prometedoras en términos de energía sostenible y sustentable, mismas que coadyuvan a la situación energética mundial, donde las reservas de combustible fósil están por agotarse [1, 2].

Particularmente, el estudio de los electrodos se lleva a cabo de acuerdo a los elementos que los conforman y a la forma en la que se organizan microestructuralmente. Es por ello que toma especial interés el desarrollo de técnicas de reconstrucción para dichos materiales, y así, obtener información que permita mejorar su eficiencia, como por ejemplo, el coeficiente efectivo de transporte [3].

La reconstrucción tridimensional presenta un paradigma en relación al esfuerzo computacional que conlleva la obtención de resultados. Es decir, la eficiencia computacional se vuelve difícil de alcanzar debido a la numerosa información que se necesita procesar. Por consiguiente, el propósito general de la presente tesis es el desarrollo de un sistema computacional de reconstrucción de materiales estocásticos que, mediante la optimización de procesos y algoritmos basados en funciones de correlación, obtenga reconstrucciones con una reducción sustancial en el tiempo de procesamiento mediante una técnica de diezmado y otras heurísticas.

1.1. Estado del Arte

Para comenzar, se presentan trabajos que describen las ideas principales que dieron pie a formular métodos teóricos sobre el estudio de materiales.

Coker y Torquato realizaron un análisis del efecto de la digitalización, a una resolución moderada, sobre la determinación de diversas magnitudes morfológicas para un modelo de medios digitalizados tridimensionales [4]. En las prácticas vistas en la literatura, se obtiene información morfológica de orden inferior, ya sea experimentalmente o teóricamente, y con ésta, se construyen límites rigurosos sobre una variedad de propiedades efectivas de los medios aleatorios [5, 6, 7, 8, 9, 10]. Mediante la capacidad para caracterizar cuantitativamente la microestructura y morfología de un medio aleatorio, se llega a la comprensión fundamental del transporte efectivo, propiedades electromagnéticas y mecánicas de los materiales heterogéneos aleatorios, tales como los materiales porosos y los compuestos [11, 12, 13, 14]. También, los autores lograron obtener alta resolución bidimensional y tridimensional en las muestras gracias a los avances experimentales en el campo de la microscopía. Asimismo, ellos describen la importancia de comprender la relación entre una función de correlación para un material real y aquella que es extraída a partir de una representación digitalizada, debido a que los datos experimentales y digitalizados fueron de resolución finita. El objeto de análisis se establece en estudiar sistemas de esferas digitalizadas superpuestas y demostrar qué cuantificaciones morfológicas son más sensibles al proceso de digitalización y resolución. Además, los autores respondieron afirmativamente a la pregunta acerca de si es necesario un conjunto de datos tridimensionales para la obtención de resultados fiables con respecto a la estructura de los materiales [4].

Patelli y Schuëller propusieron una técnica de optimización que se basa en algoritmos genéticos (GA, por sus siglas en inglés: "Genetic Algorithm") para la reconstrucción de microestructuras a partir de información reducida de las mismas [15]. Ellos mencionan que la generación digital de muestras es vista como un problema de optimización donde un conjunto de funciones de correlación están prescritas, basándose en información microestructural limitada, y los algoritmos de reconstrucción proceden a encontrar soluciones que mejor coincidan con dicho conjunto de funciones. Tuvieron entonces, como objetivo para la técnica propuesta, la identificación de un método eficiente para la reconstrucción de

microestructuras de materiales estocásticos heterogéneos. Los autores compararon el enfoque propuesto con el libro de Torquato sobre los materiales heterogéneos aleatorios, así como también, con la publicación de Sankaran y Zabaras, que se enfoca en el método de máxima entropía [16, 17]. También, realizaron ejemplos numéricos que cuantificaron el desempeño de estas técnicas de reconstrucción en términos de precisión de las soluciones, estabilidad de los métodos y esfuerzos computacionales. Entonces, los autores propusieron el algoritmo de optimización híbrido estocástico que se basa en la sinergia entre el GA y el recocido simulado. En particular, aplicaron ambos algoritmos para funciones específicas pero lógicamente relacionadas, es decir, implantaron el GA para la identificación eficiente de configuraciones de microestructura que satisfacen a los descriptores seleccionados de los materiales y utilizaron el recocido simulado para el refinamiento de soluciones identificadas por medio del GA y con esto aumentaron la precisión del algoritmo en general. Finalmente, los autores presentaron el enfoque híbrido para la reconstrucción de microestructuras, que combina la robustez del algoritmo genético con la precisión del recocido simulado.

Barbosa, Andaverde, Escobar y Cano utilizaron un método alternativo para la reconstrucción estocástica de capas catalizadoras (CL, por sus siglas en inglés: "Catalyst Layer") que propone una técnica de escalado para determinar propiedades de transporte efectivas en celdas de combustible (PEMFC, por sus siglas en inglés: "Proton Exchange Membrane Fuel Cell") [3]. La técnica de escalamiento permite una reducción de recursos computacionales necesarios para la simulación. Definieron las escalas mediante diferentes estructuras observables en escalas de magnitud de orden diferente. El método alternativo de reconstrucción permite la utilización de la información disponible antes y después de la CL. Las estructuras que se reconstruyen de esta forma se caracterizan por una función de correlación de dos puntos y una de distribución de tamaño de poro, es decir, las estructuras que se generan de esta forma se caracterizan estadísticamente por funciones de correlación. De igual manera, los autores estudiaron variables como ejemplos de estructuras para la CL, estas variables son: la fracción volumétrica del electrolito disperso, la porosidad total de la CL y la distribución de tamaño de poro.

Pant, Mitra y Secanell presentaron una metodología de reconstrucción que consiste en el intercambio de píxeles por el método de vecinos de fase diferente (DPN, por sus siglas en inglés: "Different Phase Neigh-

bor") y en el recocido jerárquico multimalla [18]. En la literatura, se identifican las desventajas del intercambio de píxeles de forma aleatoria y se comparan distintos métodos de intercambio de píxeles sesgados [19, 20, 21]. Debido a ello, la metodología de los autores, da prioridad a los píxeles que están rodeados por más píxeles de diferente fase. La reconstrucción se realiza iniciando en imágenes gruesas y sucesivamente las refina, de este modo, la información que recaba la utiliza en las etapas de refinamiento para congelar los píxeles interiores de estructuras preformadas, y así, conserva las estructuras a gran escala en las imágenes refinadas reduciendo el número de píxeles a intercambiar. Esto da como resultado una disminución en el tiempo computacional necesario para llegar a las soluciones. Asimismo, ellos demostraron que la metodología, en comparación con el recocido simulado de malla simple tradicional, reduce el tiempo de cálculo requerido para lograr las reconstrucciones alrededor de un factor de 70-90 [18]. Otros autores, de igual manera, demostraron que la utilización de métodos por DPN reduce significativamente las etapas en el proceso de reconstrucción y también mejora la precisión puesto que elimina el número de píxeles aislados [20, 21].

Xu, Gao y Li presentaron un nuevo descriptor microestructural que se basa en el análisis de componentes principales (PCA, por sus siglas en inglés: "Principal Component Analysis") para el diseño de materiales heterogéneos a partir de una base de datos de microestructuras de materiales [22]. Ellos compararon técnicas mediante descriptores de primer y segundo orden para la identificación de desventajas [23, 24]. Indicaron que, dentro de las desventajas más relevantes, el costo computacional es una de ellas. También, los autores identifican la pérdida de información, puesto que solamente información estadística es la que se capta e información clave que determina el rendimiento de los materiales se pierde, y por último, hacen referencia a la carga computacional extenuante que se presenta en términos de reconstrucción. A partir de estas desventajas, ellos formularon las bases para el descriptor, mismo que logra capturar las características geométricas de la base de datos y proporciona una representación compacta del espacio de diseño, así como, también, facilita la extracción de la microestructura de los materiales, generación de la reconstrucción de las microestructuras y preserva mejor las características en comparación con los descriptores mencionados anteriormente. Por último, los autores comprobaron el rendimiento del descriptor con un problema de análisis de elasticidad lineal [22].

Asimismo, se presentan a continuación, trabajos que fueron de suma importancia para la presente tesis, debido a que los autores incluyeron, dentro de su proceso de reconstrucción de materiales, el método de recocido simulado. Dicho método, fue implementado y optimizado mediante el análisis de sus virtudes e individualizando sus características enfocándolo a materiales estocásticos heterogéneos.

Sundararaghavan y Zabaras aplicaron una teoría de aprendizaje automático para la creación de una biblioteca entrenada con imágenes instantáneas de microestructuras obtenidas experimentalmente o computacionalmente [25]. Inicialmente, mediante el estudio de métodos de clasificación plasmados en la literatura, para la descripción de microestructuras, los autores analizaron las implementaciones de los métodos mencionados anteriormente, y llegaron a la conclusión de que los descriptores utilizados en dichas implementaciones entregan como resultado un conjunto de singularidades que se atribuyen a la ausencia de información morfológica completa [26, 27, 28]. Entonces, a partir del trabajo de análisis anterior, los autores plantearon la idea sobre lo fundamental que resulta la información microestructural para determinar propiedades críticas de materiales de alto rendimiento, así como también, identificaron la necesidad de una representación que cuantifique todos los elementos microestructurales mediante el análisis de imágenes digitalizadas. A través de estas ideas, ellos formularon la teoría de aprendizaje automático propuesta. Dicha teoría se constituye por dos fases principales para el proceso de representación y clasificación; la primera fase consiste en la implementación de un algoritmo de aprendizaje estadístico que emplea máquinas de vectores de soporte (SVM, por sus siglas en inglés: "Support Vector Machine") para la clasificación de imágenes instantáneas de microestructuras. El objetivo de la clasificación es agrupar microestructuras similares dentro de una clase donde se lleva a cabo la segunda fase del proceso. Esta segunda fase consiste en un método de análisis de componentes principales que actualiza dinámicamente la biblioteca de microestructuras y cuantifica numéricamente las características microestructurales de las mismas.

También, Sundararaghavan y Zabaras, plantearon una solución al problema de reconocimiento de patrones, que se presenta en la reconstrucción tridimensional, mediante una base de datos de microestructuras con máquinas de vectores de soporte para predecir el proceso de reconstrucción usando información estadística limitada disponible a partir de

imágenes planas [29]. La sensibilidad de la aproximación, para estudiar la relación microestructura-propiedad, se muestra mediante la comparación de las propiedades calculadas de microestructuras reconstruidas con los resultados experimentales. Los autores emplearon una base de datos como biblioteca de microestructuras 3D, y realizaron la reconstrucción planteando el problema como reconocimiento de patrones en lugar de un problema de optimización. La biblioteca se genera utilizando modelos de Monte-Carlo, a partir de la evolución de microestructuras, y se utiliza como el conjunto de datos de los cuales se seleccionan las microestructuras 3D que tengan características coincidentes [30]. Esto se logra mediante el aprendizaje automático y los medios de clasificación, utilizando descriptores de imagen plana como características. La técnica de reconocimiento de patrones utiliza firmas bidimensionales de microestructuras para generar realizaciones en 3D casi en tiempo real, acelerando la predicción de las propiedades del material. También, los autores demostraron la eficacia de la combinación de la metodología de clasificación y el análisis de componentes principales para la representación de orden reducido de microestructuras 3D, y de esta manera, contribuyeron al desarrollo de los materiales por diseño [29].

Sadati, Zamani y Mahdavian propusieron dos esquemas híbridos bajo el algoritmo de optimización de enjambre de partículas (PSO, por sus siglas en inglés: "Particle Swarm Optimization") [31]. Los algoritmos híbridos propuestos, se basan en el uso de las técnicas de optimización de enjambre de partículas en combinación con el enfoque de recocido simulado. La literatura menciona que los esquemas basados en PSO son distintos de los algoritmos evolutivos (EA, por sus siglas en inglés: "Evolutionary Algorithm"), puesto que no aplican operadores sobre la población para generar nuevas soluciones prometedoras, en cambio, cada individuo (denominado como partícula), de cada población (llamada enjambre), ajusta la trayectoria hacia la mejor posición anterior que se alcanza por cualquier miembro de los vecindarios topológicos [32, 33, 34, 35, 36, 37]. En la variante global de PSO, todo el enjambre se considera como el vecindario. Así, el intercambio global de información tiene lugar y las partículas se benefician de los descubrimientos y experiencias de los demás miembros durante la búsqueda de regiones prometedoras. El esquema propuesto presenta, en principio, una deficiencia durante la búsqueda de los puntos mínimos o máximos globales, pero gracias al recocido simulado, encuentra el mínimo global utilizando

la tecnología de búsqueda estocástica a partir de los medios de probabilidad y asegura un mínimo global cuando el espacio de parámetro se muestra infinitamente varias veces durante el período de recocido. Mediante la simulación de tres funciones de prueba diferentes, los autores demostraron cómo los algoritmos híbridos propuestos ofrecen una capacidad de convergencia hacia los puntos mínimos o máximos globales, así como, también, indicaron que, mediante el análisis de los resultados de las simulaciones, los enfoques de recocido simulado basados en los algoritmos híbridos, tienen características de convergencia más altas que los métodos de PSO puros [31].

Jiang, Chen y Burkhart desarrollaron dos conjuntos de métodos para reconstruir microestructuras 3D a partir de imágenes 2D; el primer método caracteriza la microestructura bajo ciertos descriptores estadísticos, típicamente función de correlación de dos puntos y funciones de correlación de clúster, y seguidamente, realiza un proceso de optimización para construir una estructura 3D que coincida con los descriptores estadísticos; el segundo método modela las microestructuras usando modelos estocásticos como, por ejemplo, un campo aleatorio gaussiano (GRF, por sus siglas en inglés: "Gaussian Random Field"), y entonces genera una estructura 3D directamente de la función [38]. Mediante el estudio de literatura afín a los temas presentados anteriormente, los autores fundamentaron los métodos propuestos [39, 40, 41, 42, 43, 44]. El primer método obtiene una microestructura 3D relativamente precisa, pero computacionalmente, el proceso de optimización es muy intenso, especialmente para problemas con imágenes de gran tamaño, ya que implementa un algoritmo de búsqueda estocástico, como el recocido simulado, y adopta una estrategia de movimiento vóxel por vóxel. El segundo método, genera una microestructura 3D rápidamente, pero sacrifica la precisión debido a problemas en las implementaciones numéricas, dado que únicamente representa la información de correlación de orden inferior. Entonces, a partir de las cualidades de cada método, los autores propusieron un enfoque híbrido de optimización para el modelado de microestructuras porosas 3D de materiales bifásicos isotrópicos aleatorios mediante un GRF, que combina los dos conjuntos de métodos y, por lo tanto, mantiene la precisión del método basado en correlación con una eficiencia mejorada a través del GRF. La idea original viene de Talukdar, Torsaeter, Ioannidis y Howard, que abordan la combinación de los dos enfoques, pero no profundizan el desarrollo a detalle [45].

Olchawa, Piasecki, Wiśniowski y Frączek propusieron una reconstrucción aproximada de microestructuras heterogéneas aleatorias, mediante la ley de potencias de dos exponentes (TEPL, por sus siglas en inglés: "Two-Exponent Power-Law") [46]. Ellos afirman que el punto principal es darse cuenta de que algunas de las propiedades efectivas de los materiales estocásticos aleatorios pueden correlacionarse con características estructurales espaciales elegidas. Torquato menciona que esta conexión puede denominarse una relación estructura-propiedad [16]. La idea básica del enfoque propuesto por los autores, es utilizar el modelo de esferas superpuestas (OSM, por sus siglas en inglés) con un sólo parámetro, es decir, el radio de la esfera, pero aunque la OSM es un modelo estructural flexible, existen materiales particulares que apenas se pueden modelar de esta manera, por ejemplo, compuestos reforzados con partículas. Sin embargo, para tamaños de partículas suficientemente grandes y concentraciones bajas de carga, el método es aplicable. La primera etapa, consiste en la reconstrucción de una microestructura a partir de una configuración aleatoria de vóxeles, manteniendo las concentraciones de fase. Para acelerar el proceso, el OSM prepara rápidamente un conjunto de microestructuras al azar aleatorias aproximadas. Si es necesario, la segunda etapa entra en juego. La microestructura aproximada resultante puede utilizarse de nuevo como la configuración de partida por uno de los métodos orientados a los detalles como, por ejemplo, el recocido simulado. De este modo, se obtiene una reconstrucción con mayor precisión, conservando algunas características geométricas importantes, así como, también, se obtiene una reducción sustancial en el costo computacional del proceso general de reconstrucción. Para algunos materiales, la primera etapa ofrece una aproximación aceptable de las microestructuras, y mediante el estudio de la literatura bajo este mismo contexto, se pueden incluir los materiales cuyas propiedades efectivas, como la conductividad efectiva y los módulos elásticos, son menos sensibles a los detalles estructurales [16]. Los autores probaron el método sugerido con muestras sustitutivas de cerámica y carbonato, y en cada uno de los casos, cincuenta ensayos de bajo costo proporcionaron algunos candidatos lo suficientemente aceptables para una selección de la reconstrucción óptima y, finalmente, mencionaron que cuando se planifica una mayor precisión, las reconstrucciones finales pueden servir como configuraciones de partida [46].

1.2. Planteamiento del Problema

Existen diversos estudios en la literatura, previamente explicados en la sección anterior, que hacen la reconstrucción tridimensional de materiales a partir de imágenes de microscopio. Dichos trabajos se basan en funciones que describen la estructura de los materiales, pero mediante procesos que conllevan un esfuerzo computacional muy grande. Por consiguiente, el objetivo de esta tesis, es aplicar un algoritmo metaheurístico de búsqueda mediante la optimización global. El propósito general de esta clase de algoritmos es hallar una solución que se aproxime al valor óptimo de una función, con un rango de búsqueda amplio en un tiempo de resolución relativamente corto.

1.3. Objetivo General

Desarrollar un sistema de cómputo que reconstruya, a partir de imágenes bidimensionales, materiales estocásticos de forma tridimensional utilizados en aplicaciones de energía mediante la implementación de un algoritmo de simulación numérica.

1.4. Objetivos Particulares

- Recuperar y comprender los estudios y algoritmos previos referentes a la reconstrucción de materiales estocásticos desarrollados en la Universidad de Quintana Roo.
- Implementar la reconstrucción del material mediante el recocido simulado.
- Mejorar la implementación del recocido simulado en términos de precisión y velocidad, haciendo adecuaciones en la inicialización del material y en la actualización de la evaluación de la función objetivo (Funciones de correlación de la imagen SEM-2D).
- Integrar un sistema de cómputo con los módulos desarrollados para la reconstrucción de materiales estocásticos.

1.5. Alcance

La herramienta computacional planteada en la presente tesis, permitirá la obtención de información para profundizar los conocimientos científicos sobre la reconstrucción de materiales estocásticos, además de ofrecer una visión integral sobre los métodos de optimización para eficientar procesos computacionales. También, el beneficio se extenderá a futuras generaciones quienes se interesen en el tema, debido a que el diseño y la codificación tendrán una presentación explicada que permitirá modificar u optimizar el sistema en su totalidad.

1.6. Justificación

La débil relación de la eficiencia computacional con la reconstrucción de materiales es un paradigma presente que ha causado impacto en el campo de la ciencia de los materiales, debido a que es una relación difícil de fortalecer en la práctica. Para solucionar dicho paradigma, la presente tesis se enfocará en estudiar metodologías de reconstrucción de materiales para desarrollar algoritmos que tendrán como fin común la creación de una herramienta informática, que permitirá llevar a cabo el trabajo de reconstrucción con una reducción sustancial en el tiempo de procesamiento de hasta un 99 %.

1.7. Estructura de la Tesis

La estructura de la tesis continúa en el capítulo dos, que consiste en la introducción a los temas que influyen antes y durante el proceso de reconstrucción, abarcando desde la microscopía de materiales hasta llegar al recocido simulado. Seguidamente, en el capítulo tres, se aborda el marco de trabajo de software, describiendo las tecnologías utilizadas y su proyección dentro del desarrollo del proyecto. Continuando con la secuencia, en los capítulos cuatro y cinco, se explica el funcionamiento de los algoritmos utilizados en el sistema. Después, en el capitulo seis, se presentan las pruebas realizadas y los resultados obtenidos en términos de funcionamiento y desempeño. En el capítulo siete, se escribe la conclusión de la tesis y se aportan ideas sobre el trabajo a futuro de la misma. Por último, se presenta la bibliografía consultada, así como el código del sistema plasmado en el apartado de anexo.

Capítulo 2

Microestructuras de Materiales Estocásticos y Descriptores

La reconstrucción de materiales estocásticos para aplicaciones de energía tiene gran interés, tanto teórico como práctico, en varios campos que van desde la ciencia de materiales hasta la geofísica. Una de las motivaciones de esta tesis es la comprensión de la relación entre las microestructuras y las propiedades físicas de los materiales. En virtud del desarrollo de métodos efectivos y eficientes para generar modelos tridimensionales de la estructura de los materiales, debido a las dificultades en el análisis de la geometría y topología del espacio poroso, se intenta establecer un sistema estocástico de reconstrucción de la estructura de los materiales mediante un modelo meta-heurístico. Por el interés que entraña el avance para los sistemas de energía, para delimitar la complejidad del problema, la presente tesis se enfoca en concretos, suelos y electrodos para celdas de combustible.

En la práctica, la información acerca de la microestructura de medios porosos está limitada a imágenes bidimensionales. Se propone un modelo estocástico para generar la estructura tridimensional del espacio a través de información espacial obtenida de imágenes en dos dimensiones. Toda esta terminología se explica a continuación para seguidamente abordar la metodología fundamental de desarrollo.

2.1. Microscopía de Materiales

La microscopía de materiales es el primer eslabón en la cadena de procesos necesarios para la realización de la reconstrucción de materiales estocásticos, debido a ello, se considera como una de las herramientas más importantes al momento de estudiar materiales. A simple vista se puede obtener una percepción superficial de la estructura de un material, sin embargo, las propiedades y los detalles de la estructura atómica se aprecian exclusivamente en la morfología del mismo.

Gracias al avance tecnológico se han desarrollado nuevas técnicas dentro del análisis microscópico, tales técnicas de microscopía, como por ejemplo la técnica de sonda, han aumentado las posibilidades de llevar acabo un análisis superficial más preciso de los materiales con resolución atómica. El análisis microscópico tiene como fin la obtención de precisión gracias a la capacidad aumentada de observación. Para lograr dicha capacidad es fundamental el estudio de imágenes de materiales a mayor resolución espacial que la que percibe el ojo humano, mismas imágenes que sólo pueden ser obtenidas con instrumentos especializados conocidos como microscopios [47].

2.1.1. Microscopio Electrónico de Barrido

Para el desarrollo de esta tesis se utilizaron particularmente imágenes generadas mediante un microscopio electrónico de barrido o SEM (por sus siglas en inglés: "Scanning Electron Microscope"). De esta forma las imágenes que proporcionó el microscopio otorgan altos niveles de detalle morfológicos. Estas imágenes se generan mediante un proceso que se abordará a fin de tener un panorama general del SEM y su funcionamiento.

El SEM tiene la capacidad de generar imágenes de alta resolución de la extensión superficial de un material. La principal característica del microscopio es la ausencia de botones o mando manual, todo se controla directamente por una computadora. Estructuralmente está conformado por una columna donde se encuentra la cámara de muestras y los detectores de señal. Debajo de la columna se encuentra la bomba de vacío, y el resto del equipo está conformado por una serie de unidades de procesamiento de datos para el control del microscopio y las imágenes. En la parte superior de la columna se encuentra el cañón de electrones, compuesto por un filamento metálico sometido a alta tensión y en el cual se

genera la emisión de un haz de electrones. El haz se condensa y focaliza mediante lentes electromagnéticas a su paso por la columna hasta llegar a una última lente objetivo. Encima de la lente objetivo hay una serie de bobinas de barrido que harán que el haz, que se enfoca en la muestra, barra línea por línea. Para que una muestra sea analizada tiene que cumplir una serie de características; una de ellas es la capacidad de mantener la estabilidad al bombardeo de electrones a la cual el haz la someterá, así como también debe ser conductiva a la electricidad [48].

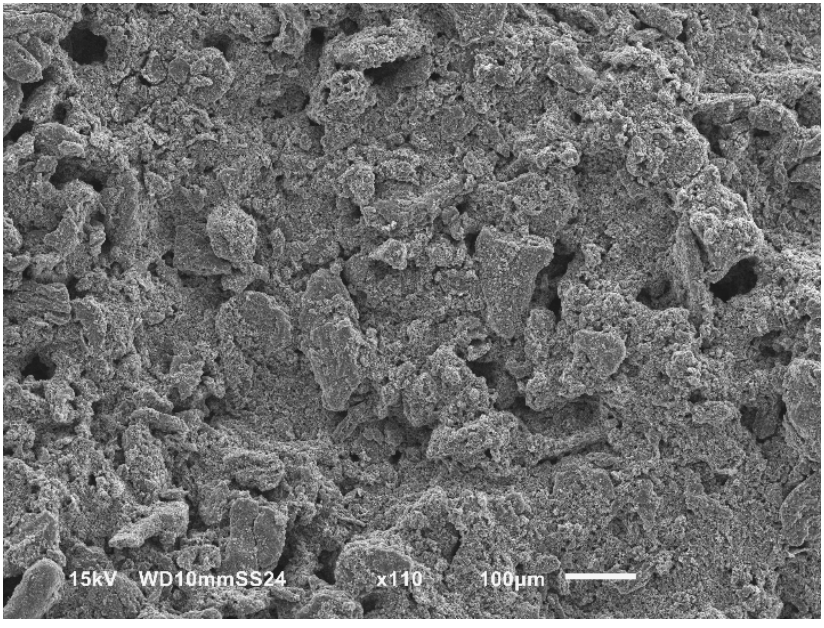


Figura 2.1: Imagen SEM

En la Figura 2.1, se aprecia una imagen resultante del SEM sobre Oro, con un contraste a escala de grises. El contraste en la imagen, se genera mediante la cuantificación de los electrones que llegan al detector y su acoplamiento a una escala de grises. Esta cuantificación está en función de la probabilidad de rebote de los electrones sobre la superficie de la muestra, donde las zonas que están más alzadas tienen mayor probabilidad de rebote.

2.2. Identificación de Fases

Continuando con la serie de procesos clave para la reconstrucción de materiales estocásticos, se presenta la identificación de fases. Este proceso consiste en la caracterización probabilística de las zonas sólidas y porosas a partir de una imagen binarizada de un material. La existencia de zonas identificables es posible gracias a que la profundidad superficial se representa en binario. La profundidad superficial, como se aprecia en la Figura 2.2, se presenta mediante la interacción espacial de los colores negro y blanco, es decir, el color negro se encuentra en los poros, mientras que el color blanco indica una fase sólida en la superficie del material [49].

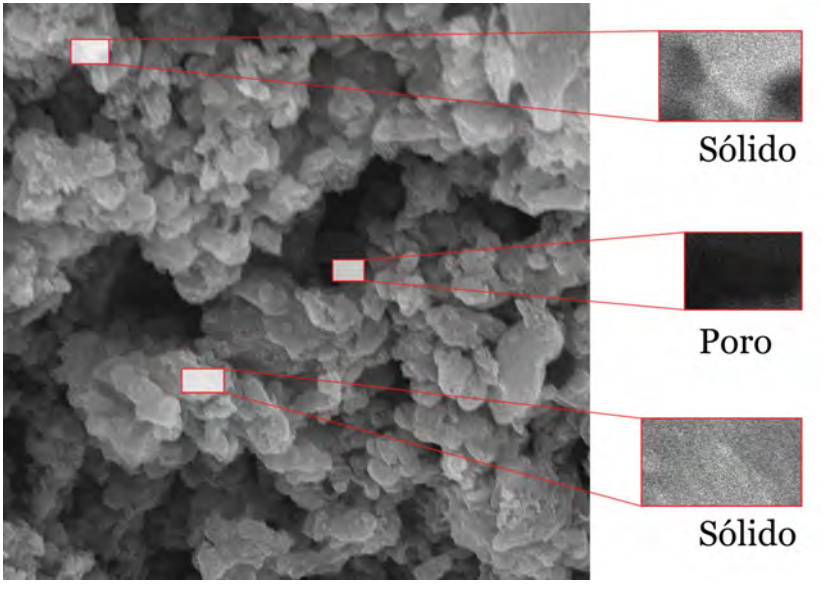


Figura 2.2: Fases Sólidas y Porosas

En la Figura 2.2 también se observa la identificación visual de fases, misma que se puede realizar mediante un análisis óptico sencillo, sin embargo, la importancia de este proceso no se basa en la información que se obtiene por una localización física, sino, en la capacidad de entregar información indispensable acerca de la distribución probabilística de las fases. Misma información que se recaba, ordena y cuantifica por 3 funciones de correlación.

2.3. Binarización de Imágenes Digitales

La binarización de imágenes digitales es el proceso que continúa en la cadena de realizaciones para la reconstrucción de materiales estocásticos que se presenta en esta tesis. En principio, la binarización consiste en la transformación de información a un plano bivalente, es decir, se obtiene una reducción de la información debido a que únicamente coexisten dos valores; verdadero y falso.

En el contexto de imágenes digitales, la binarización es el proceso de convertir una imagen a una versión reducida de la misma, donde los dos únicos colores resultantes posibles son el blanco y el negro. Para llevar a cabo esta asignación de colores se realiza un análisis individual píxel por píxel. En este análisis se encuentra un proceso contiguo llamado umbralización, que consiste en establecer un valor límite que sirve para decidir si el píxel, dependiendo de su tonalidad de color, se volverá blanco o negro. En la Figura 2.3 se aprecia el ejemplo de una imagen de la superficie de un material y su estado binario resultante en función de la escala de grises de la imagen original.

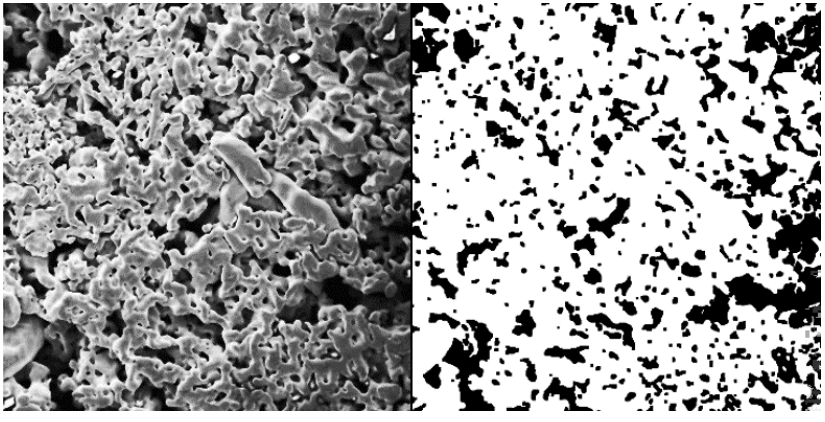


Figura 2.3: Izquierda: Imagen Original y Derecha: Imagen Binarizada

En términos de algoritmia, una imagen digital, se define como una matriz de n píxeles de alto por n píxeles de ancho, por lo tanto, para que se lleve a cabo un recorrido completo de cualquier imagen se debe aplicar una función cíclica. De esta manera se consigue que durante cada iteración se trabaje exclusivamente con un píxel [50].

2.4. Funciones de Correlación

Como se aprecia en la literatura, la utilización de una única función estadística no caracteriza de manera suficiente la microestructura de un material, esto se debe a que la cuantificación de cada clase de función de correlación tiene una naturaleza correlativa espacial diferente. La implementación de 3 funciones de correlación fue imprescindible para que el sistema lleve a cabo la caracterización cuantitativa de la superficie de los materiales. Tales funciones se desempeñan como descriptores que capturan el grado de correlación espacial entre las distintas ubicaciones en términos probabilísticos. De igual forma, sirven de referencia para la generación del volumen que se reconstruye a partir de la imagen de la microestructura del material, así como también, para que el recocido simulado recalcule el error cuadrático medio mediante la información de las funciones, pero en su versión de 3 dimensiones para el volumen reconstruido [51, 52, 53, 54, 55].

2.4.1. Función de Dos Puntos

La función de correlación de dos puntos, o también conocida como función de autocorrelación, cuantifica la probabilidad de encontrar píxeles, ubicados en los extremos de un segmento, que estén en la misma fase [4, 21, 44]. Puede ser definida de la siguiente manera:

$$S_j^{(2)}(\boldsymbol{X}_a, \boldsymbol{X}_b) \equiv \mathcal{P}\Big\{\mathcal{I}_j(\boldsymbol{X}_a)\mathcal{I}_j(\boldsymbol{X}_b) = 1\Big\} \qquad \text{(Ecu. I)}$$

Un ejemplo visual de la cuantificación mencionada anteriormente se aprecia en la Figura 2.4 donde los extremos, representados por $\boldsymbol{X}_a$ y $\boldsymbol{X}_b$, del segmento de recta con longitud r comparten la misma fase j_1.

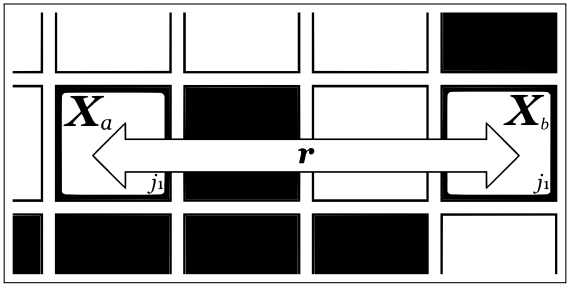


Figura 2.4: Función de Dos Puntos

2.4.2. Función de Camino Lineal

La función de correlación de camino lineal contiene la información de la conectividad de los píxeles mediante la cuantificación de la probabilidad de encontrar un segmento de línea que esté completamente en la misma fase [4, 21, 52]. Se define de la siguiente forma:

$$L_j(\boldsymbol{X}_a, \boldsymbol{X}_b) \equiv \mathcal{P}\left\{ \int_0^r \mathcal{I}_j(\boldsymbol{X}) dr = r \right\} \qquad \text{(Ecu. II)}$$

La representación visual de la cuantificación de dicha función se observa en la Figura 2.5, en la cual el segmento de píxeles de longitud r, delimitado por $\boldsymbol{X}_a$ y $\boldsymbol{X}_b$, se encuentra completamente en la fase j_1.

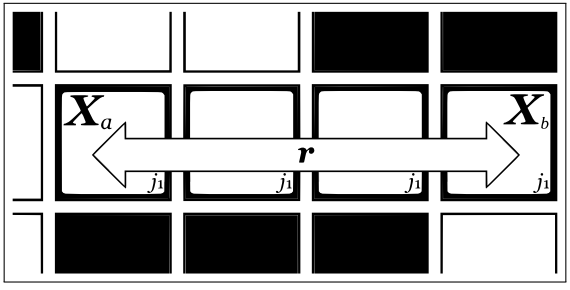


Figura 2.5: Función de Camino Lineal

2.4.3. Función por Tamaño de Poro

La función de distribución de tamaño de poro cuantifica la probabilidad de que un punto elegido aleatoriamente en la fase j se encuentre a una distancia r del punto más cercano de la interfaz j_i. En otras palabras, mide la probabilidad de encontrar círculos (o esferas, en el caso de tres dimensiones) con un radio de longitud r que estén en la misma fase j_i [4, 56]. Matemáticamente se define de la siguiente manera:

$$P_j(r) \equiv \mathcal{P}\left\{ \int_{\mathcal{V}_r} \nabla \mathcal{I}_j(\boldsymbol{y}) d\mathcal{V}_r = 1 \right\} \qquad \text{(Ecu. III)}$$

El resultado de esta función de correlación le brinda al sistema la pauta para llevar a cabo la generación del volumen inicial, mediante la inserción aleatoria de esferas con todos los radios posibles a partir de la imagen de la microestructura del material. Dicha lista de radios se genera para las fases porosas y vacías.

2.5. Recocido Simulado

El recocido simulado es un método de optimización metaheurístico que se basa en el proceso físico de enfriamiento aplicado a materiales cerámicos o metálicos, y al análisis del comportamiento de la microestructura de los mismos a medida que se van enfriando [57]. Este proceso físico comienza con el incremento en la temperatura del material, generando así el aumento de energía en sus átomos y, por consiguiente, se desplazan de sus posiciones iniciales. Seguidamente se aplica el enfriamiento, causando así la pérdida de movilidad en las moléculas agitadas, esto permite que se realice una alineación cristalina. Cuando la estructura se alinea completamente se considera que es el estado de energía mínimo. La velocidad de reducción de la temperatura debe ser lo suficientemente lenta para asegurar una alineación cristalina deseada.

El recocido simulado es una implementación algorítmica del proceso físico explicado anteriormente, y se utiliza para encontrar la solución óptima de una función objetivo. Dicha solución es hallada mediante la utilización de una estrategia de búsqueda aleatoria, misma que permite no sólo nuevas posiciones que disminuyan la función objetivo, sino que también acepta posiciones que incrementen su valor [58, 59, 60, 61].

Algoritmo: Recocido Simulado

```
Se crea la solución inicial, x(0);
Se establece la temperatura inicial, T(0);
t = 0;
repetir
    Se genera una nueva solución, x;
    Se determina la calidad de la nueva solución, f(x);
    Se calcula la probabilidad de aceptación;
    si U(0,1) ⩽ probabilidad de aceptación entonces
        x(t) = x;
    fin
hasta que Mientras la condición de parada sea verdadera;
Se regresa x(t) como la solución óptima;
```

El algoritmo del recocido simulado compara la calidad del estado actual con la del nuevo estado y, mediante una probabilidad de aceptación, decide si realizar o no la transición al nuevo estado [62].

Capítulo 3

Marco de Trabajo de Software

En este capítulo se presentan las tecnologías computacionales que fueron el ambiente donde el sistema se desarrolló, que son: el lenguaje de programación, la librería gráfica utilizada y el entorno de desarrollo integrado (IDE, por sus siglas en inglés: "Integrated Development Environment"). Por último, se menciona la metodología de desarrollo de software que se estableció y siguió rigurosamente para definir los parámetros de programación y entrega de los avances en las distintas versiones del sistema.

3.1. Lenguaje de Programación C#

Gracias al avance tecnológico, que se muestra progresivo al paso del tiempo, las ciencias computacionales se han visto beneficiadas en términos de variedad y eficiencia. Una de las ventajas en el campo de la computación, es la amplia gama de lenguajes de programación que coexisten actualmente. Debido a ello, la selección del lenguaje indicado para el desarrollo del sistema permitió vislumbrar factores importantes como requisitos de construcción del software, mismos que fueron cubiertos por el lenguaje de programación C#.

El lenguaje de programación C#, desarrollado por Microsoft en el año 2000, se creó para suplir las deficiencias que presenta la plataforma *Visual Basic*. Estas deficiencias se basan en la pobreza del lenguaje en general y se debe a que *Visual Basic* fue planeado desde su diseño inicial como un lenguaje sencillo y fácil de aprender para programadores principiantes.

En particular, el lenguaje de programación C# permite un desarrollo robusto y efectivo en términos de estructura de datos de alto nivel. Así también, cuenta con un enfoque simple y eficaz para la programación orientada a objetos. Por otra parte, el ahorro en el tiempo de programación que presenta la plataforma se obtiene gracias a que cuenta con una librería de clases muy bien diseñada y completa. Todo este conjunto de cualidades aseguraron los requisitos necesarios para ir por buen camino al éxito del desarrollo, ya que la función principal del sistema es coadyuvar en la obtención de resultados de calidad científica mediante métodos matemáticos programados, y es precisamente ahí donde la robustez y la eficiencia del lenguaje juegan un papel importante [63, 64].

3.2. Librería Gráfica GTK+

Uno de los objetivos modernos para el desarrollo de cualquier sistema o aplicación informática en general es el alcance funcional sobre la mayoría de los principales sistemas operativos. Este objetivo no fue la excepción para el sistema desarrollado en la presente tesis. En otras palabras, la capacidad integral de ser un sistema multiplataforma formó parte de los requisitos previamente establecidos, y fue logrado mediante el uso de la librería gráfica GTK+.

La librería gráfica GTK+ está constituida por un conjunto de archi-

vos elaborados por el equipo de desarrollo de GNOME, y es utilizada para la creación de interfaces gráficas de usuario o GUI (por sus siglas en inglés: "Graphical User Interface"). Gracias al basto conjunto de widgets que contiene, la pertinencia de su utilización se vuelve adecuada para el desarrollo de proyectos que van desde herramientas sencillas hasta la posibilidad de crear paquetes completos de aplicaciones.

El sistema de reconstrucción desarrollado en esta tesis se declara multiplataforma, debido a que GTK+ utiliza GDK de manera interna para llevar a cabo la visualización de los widgets. GDK se presenta como una interfaz de programación de aplicaciones o API (por sus siglas en inglés: "Application Programming Interface"), misma que se encuentra ubicada sobre la API gráfica nativa del sistema operativo, y es gracias a dicha ubicación de trabajo que la integración de GDK permite que las aplicaciones construidas con GTK+ puedan utilizarse en otras plataformas [65, 66].

3.3. Generador de Interfaces Xamarin Studio

Para llevar a cabo una correcta selección del ambiente de desarrollo de software o IDE, es importante tomar en cuenta las necesidades presentes a la hora de decidir la manera más efectiva de solucionar el paradigma computacional. Tomar en cuenta dichas necesidades le permite al programador acelerar el aprendizaje en dirección a la solución mientras se adapta a una forma particular de trabajo, por ejemplo, identificar si la usabilidad de la herramienta a desarrollar será para fines académicos o, de lo contrario, para un ambiente profesional, de esta forma se aclara la idea sobre si es necesaria una IDE intuitiva o una especializada.

Para el caso particular del sistema de reconstrucción contenido en esta tesis, se optó por el uso de Xamarin Studio en su versión 6.3, la más reciente hasta la fecha. Como se ha mencionado antes, la importancia de identificar las necesidades de desarrollo fueron pieza clave para la selección de dicha IDE, y esto se debe a que, de manera intuitiva, Xamarin permite un escritura integral de programación con C# en conjunto con GTK+, permitiendo así que la robustez del lenguaje de programación pueda ejecutarse en armonía con la capacidad multiplataforma que otorga la librería gráfica. También cuenta con versiones tanto para Windows como para Mac OS X [67].

3.4. Metodología de Desarrollo de Software

El trabajo de creación y desarrollo de una herramienta informática eficiente, que consiga llegar a los requerimientos establecidos en su totalidad, es una labor compleja de cumplir. Sin embargo, existen metodologías para el desarrollo de software que permiten al programador seguir un camino o marco de trabajo no sólo para obtener un proceso de desarrollo estructurado, planificado y controlado, sino también un aumento en la calidad del software que se producirá. El fin principal de las metodologías es imponer un proceso disciplinado por parte del desarrollador y eficientar todas las fases inmersas en el desarrollo del proyecto.

En la actualidad, existe una variedad de metodologías de desarrollo que permiten su selección conforme las características presentes en cada proyecto informático. Dicha selección está a disposición de la adaptación a ciertas cualidades en los proyectos, como por ejemplo, el equipo de desarrollo, recursos informáticos y tiempo de entrega. A pesar de existir una amplia gama de metodologías se pueden clasificar en dos grupos conforme a sus características y objetivos: ágiles y robustas.

A partir de las características identificadas durante el proceso de diseño inicial del sistema de reconstrucción, se decidió que el modelo incremental sería la opción adecuada para alcanzar los objetivos. Este modelo es el resultado de la combinación de los modelos de cascada y prototipos, y es gracias a esta combinación que logra mantener las virtudes de ambos. A grandes rasgos, su funcionamiento consistió en realizar varias iteraciones, como ocurre en el modelo de cascada, pero sin terminar ninguna. Sin embargo, mediante la realización de las iteraciones es como el sistema evolucionó constantemente, ya que después de cada iteración se fueron agregando nuevas características, especificaciones, funciones y opciones que se iban requiriendo. Dicho de otro modo, el modelo incremental repitió la forma de trabajar del modelo de cascada, pero con la característica de agregar pequeñas modificaciones o actualizaciones, las cuales se fueron integrando al sistema. Es de esta forma en la que los usuarios finales se vieron sumamente sumergidos en el desarrollo y, por consiguiente, en la obtención de un resultado óptimo [68, 69].

Capítulo 4

Algoritmos para la Reconstrucción

El sistema desarrollado en la presente tesis, tiene la finalidad primordial de llevar a cabo reconstrucciones de microestructuras de materiales mediante un conjunto de procesos lógicamente relacionados. La importancia de entender el funcionamiento de cada uno de dichos procesos le permitirá al lector tener una idea más amplia de cómo procesan la información durante su interacción.

En este capítulo, se realiza un exposición detallada de los algoritmos programados que intervienen directamente durante la reconstrucción de materiales realizada por el sistema. Cabe recalcar que se presentan los métodos y las variables internas más relevantes para cada función, explicando el funcionamiento de las distintas mecánicas y el porqué de su declaración. El capítulo inicia con el primer proceso presente cuando el sistema entra en funcionamiento, que es la binarización digital, seguidamente, se continúa con las funciones de correlación de CL y DP, y cómo éstas logran coexistir bajo un mismo conjunto de ciclos. Después, se prosigue con la función de correlación de tamaño de poro y finalmente, se presenta la función encargada de realizar la reconstrucción mediante el recocido simulado.

4.1. Binarizado Digital

La binarización digital consiste en la aplicación de un algoritmo que evalúe individualmente cada píxel de una imagen, en función de la comparación del valor de la intensidad de gris con un umbral. Si el valor de la intensidad de gris del píxel es mayor que el valor del umbral entonces se le asignará el color blanco en la nueva imagen binarizada, de lo contrario, se le asignará el color negro. A continuación, se presenta el funcionamiento algorítmico de la binarización digital dentro del sistema. En el Código 4.1, el proceso comienza con la adquisición de la información de la imagen, desde el equipo hacia la memoria del sistema:

```
1   System.IO.FileStream file = System.IO.File.OpenRead(filechooser.
        Filename);
2   image1.Pixbuf = new Gdk.Pixbuf(file.Name);
```

Código 4.1: Binarizado Parte 1

Una vez cargada la imagen, en el Código 4.2, se declara el umbral y se crea un nuevo vector que guardará la información resultante a lo largo de la evaluación individual de los píxeles. El umbral es establecido a un valor de 127, que es la mitad del máximo valor a 8 bits que se puede obtener por el color de un píxel. El tamaño del nuevo vector está en función del ancho por el largo de la imagen original para así contener todos los píxeles que existen en ésta:

```
3   int umbral = 127;
4   byte[] pixels = new byte[original_img.Height * original_img.Width];
```

Código 4.2: Binarizado Parte 2

En el Código 4.3, la evaluación se lleva a cabo comenzando con la adquisición individual del píxel y, entonces, se calcula el valor unificado de la tonalidad de su color:

```
7   pixelColor = original_img.GetPixel(x, y);
8   gray = (byte)(pixelColor.R * 0.3f + pixelColor.G * 0.59f + pixelColor.
        B * 0.11f);
```

Código 4.3: Binarizado Parte 3

Por último, el valor calculado se compara con el umbral y, dependiendo si fue mayor o menor a éste, se asigna uno de los dos posibles colores en la posición del píxel pero en el nuevo vector. El proceso de evaluación y asignación se repite hasta que las condiciones de los ciclos terminen, es decir, hasta que la imagen sea recorrida por completo.

4.2. Funciones de Correlación: Camino Lineal y Dos Puntos

En el sistema de reconstrucción, las funciones de correlación de camino lineal y de dos puntos, se encuentran bajo un mismo conjunto de ciclos encargados de recorrer por completo la imagen. Su función primordial es la de cuantificar segmentos verticales y horizontales, que cumplan con las condiciones previamente presentadas en las Figuras 2.4 y 2.5. Posteriormente, se presentan los algoritmos de las funciones de correlación para 2D y 3D, señalando las diferencias que la adaptación de la dimensión extra conllevó. Es importante mencionar que una única matriz contendrá la información recabada por las funciones. Dicha matriz tiene un tamaño de 5 filas, y cada fila tiene una longitud igual al largo de la imagen entre dos. Las filas 0 y 1 guardarán la cuantificación de las funciones de correlación DP y CL respectivamente, para píxeles blancos, mientras que las filas 2 y 3 guardarán dichas funciones en el mismo orden, pero para píxeles negros. Por último, la fila 5 guardará todos los segmentos posibles. Para 3D, se lleva a cabo dicha declaración vista en el Código 4.4, pero agregando una variable definida como *layer*, que controlará el ciclo extra necesario para trabajar con un volumen:

```
3   int row, column, relative_length, incremental_f;
4   int real_length = img_width > img_height ? img_width / 2 : img_height
        / 2;
5   real_length = 1 > real_length ? 1 : real_length;
6   UInt32[,] corr_func_2p_lp = new UInt32[5, real_length];
7   UInt32 first_pixel, pixel_denied;
```

Código 4.4: FC 2D: Camino Lineal y Dos Puntos Parte 1

También, se realiza una validación para conocer el valor individualizado del píxel y del vóxel. El valor servirá para saber si el segmento, que se comenzará a evaluar, inicia con un 1 (blanco) o un 0 (negro), y así, definir qué conjunto de ciclos, para cada color, se utilizará. El conjunto de ciclos, mencionados anteriormente, se constituye por un par de conteos para 2D, uno se realiza de forma horizontal mientras que el otro se hace de manera vertical. En el Código 4.5, se aprecia el primer conteo de este par. Dicho conteo horizontal se lleva a cabo en las líneas 15 y 16, mediante la inclusión de un factor incremental, denominado $incremental\_f$, que se encarga de solamente aumentar el valor de la posición de la columna mientras que la variable *row*, que controla la

posición de la fila, se mantiene estática:

```
12  for (incremental_f = 0; incremental_f < relative_length; incremental_f
        ++)
13  {
14    corr_func_2p_lp[4, incremental_f]++;
15    corr_func_2p_lp[0, incremental_f] += (first_pixel & pixels[(row *
          img_width) + column + incremental_f]);
16    pixel_denied = pixel_denied & pixels[(row * img_width) + column +
          incremental_f];
17    corr_func_2p_lp[1, incremental_f] += pixel_denied;
18  }
```

Código 4.5: FC 2D: Camino Lineal y Dos Puntos Parte 2

Así mismo, en el Código 4.5, la ecuación de la función de correlación de DP (Ecu. I) se utiliza de forma directa en la línea 15, mientras que la ecuación de la función de CL (Ecu. II) se aprecia aplicada en las líneas 16 y 17, demostrando la coexistencia de ambas funciones bajo los mismos ciclos. La mecánica explicada anteriormente le permite a las líneas 15 y 16 realizar una suma, donde en algunos casos suman 0 y en otros suman 1. El valor sumado depende, para cada función, si se sigue cumpliendo la condición de mantenerse en la misma fase. Por otra parte, en el Código 4.6, el conteo vertical se presenta en las líneas 22 y 23, mediante el mismo factor incremental mencionado anteriormente, pero ahora se encuentra aumentando solamente el valor de la posición de las filas y ya no el de las columnas:

```
19  for (incremental_f = 0; incremental_f < relative_length; incremental_f
        ++)
20  {
21    corr_func_2p_lp[4, incremental_f]++;
22    corr_func_2p_lp[0, incremental_f] += (first_pixel & pixels[((row +
          incremental_f) * img_width) + column]);
23    pixel_denied = pixel_denied & pixels[((row + incremental_f) *
          img_width) + column];
24    corr_func_2p_lp[1, incremental_f] += pixel_denied;
25  }
```

Código 4.6: FC 2D: Camino Lineal y Dos Puntos Parte 3

En la versión 3D, se agrega un conteo más a los dos explicados anteriormente, encargado de la misma cuantificación presentada pero afectando solamente a la profundidad.

4.3. Función de Correlación: Tamaño de Poro

La función de correlación de tamaño de poro, inmersa en el sistema de reconstrucción, tiene la tarea de cuantificar círculos y esferas, mediante el algoritmo diseñado para 2D y 3D respectivamente, que cumplan con la característica de ser completamente de una sola fase. El desglose de los algoritmos de la función de tamaño de poro, para 2D y 3D, se presenta a continuación, denotando las diferencias entre ambas versiones que básicamente consisten en la adición dimensional necesaria para convertir la versión 2D en 3D. El algoritmo comienza mediante la declaración de la función con sus respectivos parámetros de entrada, que son el ancho y alto de la imagen, así como también el vector resultante al binarizado. En el caso de la versión 3D, la declaración incluye la variable *img_deep*, representativa a la profundidad del volumen.

Después, se declaran las variables de control interno de la función, entre las cuales, se contemplan dos bajo el nombre de *minimum_radius* y *maximum_radius*, la primera con un valor de 0 y la segunda con un valor de 20. Dichas variables definen el radio mínimo y máximo de los círculos a evaluar por el algoritmo. De igual forma, se declara la matriz encargada de resguardar la información resultante al proceso de cuantificación de la función. Esta matriz tiene un tamaño de 3 filas, y cada fila tiene una longitud igual al máximo radio permisible más 1. La primera fila de dicha matriz contendrá el resultado de los círculos en la fase sólida, mientras que la segunda fila contendrá el mismo resultado pero para la fase vacía, y por último, la tercera fila contendrá todos los círculos posibles con un radio máximo de 20 píxeles. En la versión 3D, se integran las variables *layer*, *edge_layer* y *radius_layer* para el control extra necesario.

Continuando con el algoritmo en 2D, se lleva a cabo la inicialización de dos ciclos encargados de recorrer toda la imagen, pero con la peculiaridad de que, mientras el radio mínimo va aumentando automáticamente, la zona a evaluar se va reduciendo. De esta manera, se evitan situaciones donde la evaluación intente acceder a posiciones inexistentes, debido a la innecesaria búsqueda de círculos con un radio mayor a la distancia de algún borde. Para la versión en 3D, se realiza la declaración de un ciclo más a los dos anteriores, debido a que la profundidad del volumen interviene. También, se lleva a cabo la validación que permite ubicar al píxel o vóxel en turno, en función de la fase en la que se encuentre, con el conjunto de ciclos encargados de realizar la evaluación de la función.

Seguidamente, en el Código 4.7, de la línea 12 hasta la 15, se declara una serie de condiciones que, en conjunto, se encargarán de hallar y asignar a la variable nombrada como *main_edge*, el valor de la distancia más corta que exista entre la posición actual y alguno de los bordes de las filas o columnas. El resultado de la asignación mencionada anteriormente, servirá para condicionar al ciclo que se declara en la línea 16, mismo que se encarga de mantener el valor del radio principal hasta que se termine toda la evaluación a partir de la posición actual. El radio principal, nombrado en el código como *main_radius*, es el valor en turno que sirve como referencia de control que delimita el rango circular de búsqueda de píxeles, sobre la posición actual:

```
12   edge_column = column < (img_width - column - 1) ? column : img_width -
          column - 1;
13   edge_row = row < (img_height - row - 1) ? row : img_height - row - 1;
14   edge_column = edge_column < edge_row ? edge_column : edge_row;
15   main_edge = maximum_radius < edge_column ? maximum_radius :
          edge_column;
16   for (main_radius = minimum_radius; main_radius <= main_edge;
          main_radius++)
```

Código 4.7: FC 2D: Tamaño de Poro Parte 1

La versión 3D, realiza un proceso parecido al explicado anteriormente, pero éste incluye el borde de la profundidad, que se presenta debido a la inclusión de la dimensión extra, y en este caso, ya no se realiza una búsqueda por círculos, en cambio, dicha búsqueda es mediante esferas.

Prosiguiendo, se declara la variable llamada *control_radius*, encargada de contener el resultado de la multiplicación del radio principal por si mismo una vez más. Dicha variable tendrá un uso específico que se explicará más adelante. Luego, se declaran los ciclos encargados del recorrido alrededor de la posición en turno. Estos ciclos tienen la característica de contener una doble condición. La primera consiste en verificar si se ha encontrado un píxel de fase diferente con respecto a la fase de la posición actual, y la segunda condición verifica si la longitud del radio de la fila o columna sigue siendo menor que la del radio principal. En la versión 3D, se lleva a cabo la adición de un ciclo más, ya que el recorrido alrededor de la posición actual ya no es de forma circular, sino, de modo esférico.

Una vez inicializados los ciclos que controlan la posición global, y los ciclos que controlan el recorrido alrededor de la misma posición, en el Código 4.8, se declara la evaluación condicional que consiste en la representación algorítmica de la ecuación de la función de correlación

de tamaño de poro (Ecu. III). Es importante aclarar que el espacio de trabajo por el cual se lleva a cabo el recorrido forma un cuadrado en la versión 2D, y un cubo en la versión 3D, y se realiza de esta manera para evitar calcular la raíz cuadrada de la distancia, logrando así, reducir el esfuerzo computacional. A grandes rasgos, la evaluación consiste en saber si el píxel evaluado se encuentra dentro del radio del círculo, ya que las posiciones contenidas en las esquinas del espacio de trabajo no entran en el radio. El valor obtenido corresponde a la variable nombrada *evaluator*, en la línea 20. Este valor se obtiene comenzando con la multiplicación del radio de la columna y el radio de la fila por sí mismos, y luego se suman dichos resultados, después se compara el resultado de la suma con el valor de la variable *control_radius*, y se toma el valor del píxel si está dentro del círculo, o cero, si éste se encuentra fuera. Mientras el valor de la variable *evaluator* sea igual a la fase del píxel origen, el proceso de evaluación continuará su recorrido sin ninguna clase de cambio, sin embargo, en el momento que se deje de cumplir dicha condición de igualdad de fase, se registrará el cambio en la variable *pixel_denied*. Esta variable se encarga de verificar si existe un píxel con una fase distinta a la del píxel origen durante el recorrido:

```
20    evaluator = Convert.ToUInt32((radius_row * radius_row + radius_column
          * radius_column) <= control_radius ? pixels[(row + radius_row) *
          img_width + (column + radius_column)] : 0);
21    pixel_denied = pixel_denied & evaluator;
```

Código 4.8: FC 2D: Tamaño de Poro Parte 2

Las diferencias primordiales con la versión en 3D, consisten en la aparición de la variable *radius_layer*, integrada en la multiplicación de los radios, y la variable *layer*, que está en el acceso al vóxel.

Al finalizar cada recorrido, independientemente del valor del radio principal, se lleva a cabo un registro, como se puede apreciar en la línea 22 del Código 4.9, cuando la variable *pixel_denied* haya registrado algún cambio de fase se sumará un 0, por otra parte, se sumará un 1 si el recorrido bajo el radio principal se mantuvo siempre en la misma fase que la del píxel origen. En la línea 23 del mismo código, también se registra el recorrido pero sin ninguna clase de condición:

```
22    corr_func_ps[0, main_radius] += pixel_denied;
23    corr_func_ps[2, main_radius]++;
```

Código 4.9: FC 2D: Tamaño de Poro Parte 3

4.4. Reconstrucción con Recocido Simulado

El sistema expuesto en la presente tesis, utiliza el método de recocido simulado para llevar a cabo las reconstrucciones de las microestructuras, a partir de imágenes bidimensionales de las mismas. A grandes rasgos, este método realiza la búsqueda de una reconstrucción óptima mediante la comparación iterativa de una nueva solución con una solución anterior y, dependiendo de cual tenga una mejor calidad, decide si quedarse con la nueva o mantenerse en la anterior. La particularidad de este método consiste en la utilización de una variable que representa la temperatura aplicada al material, y que con el paso del tiempo va disminuyendo su valor mientras se realiza la búsqueda de soluciones. También, en el caso de que la nueva solución no tenga una mejor calidad, el método permite que, mediante una probabilidad de aceptación en función del valor de la temperatura, se pueda aceptar y así lograr salir de mínimos locales. Se lleva a cabo la declaración de las variables internas de la función, entre las cuales se presentan a continuación las más relevantes.

La variable llamada $minimum\_temperature\_factor$, se trata de un factor de precisión para definir la temperatura mínima en función de la longitud del volumen, es decir, mientras éste sea de mayor longitud el factor también lo será. Siguiendo con la declaración de variables, se presentan 3 variables de control. La primera, llamada *temperature*, se encarga de definir la temperatura inicial del sistema, la segunda variable, llamada *epsilon*, es el valor del error mínimo permisible por el ciclo principal del recocido simulado y, por último, la variable llamada *alpha*, que es el factor que controla que tan rápido el sistema bajará de temperatura. Es importante mencionar que el valor asignado a cada una de ellas fue el resultado de un conjunto de pruebas para determinar qué cantidades, para cada variable, serían las que le otorgaran al sistema el equilibrio ideal entre precisión y tiempo de procesamiento.

Una vez establecidas las variables de control, el algoritmo prosigue con la obtención del error inicial mediante el cálculo del error cuadrático medio. Dicho error, es el resultado de cuantificar la diferencia existente entre las funciones de correlación de DP y CL en 3D con respecto a las 2D, mismas que tienen la información de la imagen original, mientras que las 3D contienen la información del volumen inicial. El algoritmo obtiene este error comenzando con un ciclo encargado de recorrer las matrices de las funciones en 2D y 3D.

El error cuadrático medio, se encarga de comparar dos grupos de

datos para medir la cantidad de error que existe entre ellos [70]. Como se puede apreciar en la línea 10 del siguiente código, matemáticamente se inicia con la división del valor en turno de la columna actual de la función de correlación, con el total contenido en la quinta fila de las matrices, esta división se realiza para normalizar el valor en turno, tanto de la función en 2D como en la 3D. Seguidamente, los resultados de ambas divisiones se restan para hallar la diferencia entre el valor predicho en la función 3D, y el valor conocido en la función 2D. Después, el resultado de la resta se eleva al cuadrado para evitar valores negativos y obtener el valor absoluto del error. Esto se realiza para cada fase contenida en las 2 funciones de correlación, es decir, se realiza dicho proceso un total de 4 veces por iteración para abarcar las dos fases en cada función de correlación, y después, sumar los resultados para luego ser acumulados en la variable *solution_quality* en la línea 13:

```
10   solution_quality_2p_w = ((double)corr_func_2p_lp_3d[0,
         position_control] / (double)corr_func_2p_lp_3d[4, position_control
         ]) - ((double)corr_func_2p_lp[0, position_control] / (double)
         corr_func_2p_lp[4, position_control]);
11   solution_quality_2p_w = Math.Pow(solution_quality_2p_w, 2);
12   ...
13   solution_quality = solution_quality + solution_quality_2p_w +
         solution_quality_lp_w + solution_quality_2p_b +
         solution_quality_lp_b;
```

Código 4.10: Recocido Simulado Parte 1

Una vez que se recorren todas las posiciones de las funciones de correlación, el error del volumen de trabajo declarado en el Código 4.11, se obtiene mediante una probabilidad de aceptación que toma en cuenta la longitud del volumen, y es igual a la división de 1 entre la longitud de la matriz de las funciones de correlación multiplicada por dos, y todo esto, es multiplicado por el valor contenido en la variable *solution_quality*, misma que es representativa de la calidad de la solución inicial [62]:

```
14   error = (1.0 / (2.0 * (double)corr_func_2p_lp_size)) *
         solution_quality;
```

Código 4.11: Recocido Simulado Parte 2

A partir del final del ciclo expuesto anteriormente, el recocido simulado se encargará de reducir el error obtenido mediante la búsqueda de soluciones que cuenten con una mejor calidad. Esta búsqueda comienza con la declaración del ciclo principal de la función. El ciclo cuenta con

dos condiciones que, hasta que alguna de las dos deje de cumplirse, controlarán la realización de iteraciones. La primera condición consisten en que la temperatura mínima sea alcanzada, y la segunda, en la reducción del error hasta que tenga un valor aceptable.

El ciclo principal de la función, comienza con un mecanismo encargado de intercambiar dos vóxeles de fase distinta. Esto se logra mediante la búsqueda aleatoria de los mismos. En el Código 4.12, es mediante el método $random.Next()$, que aparece en las líneas 16, 17 y 18, el que devuelve valores aleatorios entre 0 y la longitud del volumen. Una vez que se obtienen estos valores, en la línea 19, se realiza el acceso individual al vóxel para conocer la fase en la que se encuentra:

```
16   random_layer_white_b = random.Next(0, length);
17   random_row_white_b = random.Next(0, length);
18   random_column_white_b = random.Next(0, length);
19   control_voxel_value = volume_solution[(random_layer_white_b * length *
          length) + (random_row_white_b * length) + random_column_white_b]
         == 1 ? control_voxel_value + 1 : control_voxel_value;
```

Código 4.12: Recocido Simulado Parte 3

Cuando los 2 vóxeles de fase distinta son encontrados, el mecanismo continúa con el intercambio de sus posiciones y el recálculo que dicho intercambio conlleva sobre la matriz de las funciones de correlación en 3D. En otras palabras, el vóxel de fase sólida pasa a la posición del segundo vóxel, que se encuentra en la fase porosa, y viceversa. En la línea 20 del Código 4.13, el intercambio y el recálculo de las funciones se realiza mediante la función llamada $main\_corr\_func\_2p\_lp\_3d\_1p$, encargada de hacer la resta y suma tridimensional exclusivamente para un sólo elemento, es así como el recocido simulado no pierde esfuerzo computacional que le costaría si tuviera que recalcular ambas funciones para el nuevo volumen. Después, en la línea 21, se realiza el intercambio de fase sobre el volumen de trabajo. El intercambio, tanto en la función como en el volumen, se realiza un par de veces, una por cada vóxel:

```
20   corr_func_2p_lp_3d = main_corr_func_2p_lp_3d_1p(random_column_white_b,
          random_row_white_b, random_layer_white_b, length, volume_solution
         , corr_func_2p_lp_3d);
21   volume_solution[(random_layer_white_b * length * length) + (
         random_row_white_b * length) + random_column_white_b] = 0;
```

Código 4.13: Recocido Simulado Parte 4

Los cambios realizados sobre la matriz de las funciones de correlación, se verán reflejados durante el cálculo de un nuevo error, mismo que puede verse parcialmente en el Código 4.14. En principio, este nuevo cálculo es muy similar al que se realizó en los Códigos 4.10 y 4.11, con la única diferencia de que la variable que contendrá finalmente el resultado es nombrada como *new_error*, y se encuentra presente en la línea 30, justo después del ciclo encargado de calcular la calidad de la nueva solución:

```
22   solution_quality = 0;
23   for (position_control = 0; position_control < corr_func_2p_lp_size;
         position_control++)
24   {
25     solution_quality_2p_w = ((double)corr_func_2p_lp_3d[0,
           position_control] / (double)corr_func_2p_lp_3d[4,
           position_control]) - ((double)corr_func_2p_lp[0,
           position_control] / (double)corr_func_2p_lp[4, position_control
           ]);
26     solution_quality_2p_w = Math.Pow(solution_quality_2p_w, 2);
27     ...
28     solution_quality = solution_quality + solution_quality_2p_w +
           solution_quality_lp_w + solution_quality_2p_b +
           solution_quality_lp_b;
29   }
30   new_error = (1.0 / (2.0 * (double)corr_func_2p_lp_size)) *
         solution_quality;
```

Código 4.14: Recocido Simulado Parte 5

En este punto, el ciclo principal cuenta con dos errores que pasarán a restarse. Si el resultado de la resta, representado por la variable *delta*, entre el nuevo error y el error anterior, es menor a 0, automáticamente se aceptan los cambios realizados sobre la matriz de las funciones de correlación, y también sobre el volumen de trabajo. Sin embargo, como se puede apreciar en el Código 4.15, si el resultado no es favorable para la nueva solución, ésta no se descarta automáticamente, en cambio, entra en una condición extra, que consiste en una probabilidad de aceptación basada en la distribución Boltzmann–Gibbs. Esta probabilidad decidirá si se mantienen o no los cambios de la nueva solución [62]:

```
35   probability = random.NextDouble();
36   acceptance_probability = Math.Exp(-delta / temperature);
37   if (probability < acceptance_probability)
38     error = delta + error;
```

Código 4.15: Recocido Simulado Parte 6

Si después de la segunda condición el resultado sigue siendo desfavorable para la nueva solución, todos los cambios realizados sobre las funciones de correlación, así como también los realizados sobre el volumen de trabajo, se deshacen mediante la realización del proceso inverso al visto en el Código 4.13, dicho proceso inverso se realiza en el Código 4.16. Esta reversibilidad es posible gracias a que las variables utilizadas para obtener las coordenadas de los dos vóxeles intercambiados se mantienen en la memoria durante la misma iteración:

```
39   else
40     corr_func_2p_lp_3d = main_corr_func_2p_lp_3d_1p(
           random_column_white_b, random_row_white_b, random_layer_white_b,
            length, volume_solution, corr_func_2p_lp_3d);
41     volume_solution[(random_layer_white_b * length * length) + (
           random_row_white_b * length) + random_column_white_b] = 1;
```

Código 4.16: Recocido Simulado Parte 7

Al finalizar cada iteración, la temperatura disminuye debido a que modifica su valor multiplicándolo por un factor decremental denominado *alpha*. Y por último, una vez que se alcanza la temperatura mínima o el error disminuye lo suficiente, el ciclo principal termina y la función regresa el volumen reconstruido.

Capítulo 5

Algoritmos para Acelerar la Reconstrucción

La eficiencia computacional, en términos de tiempo de procesamiento, es un recurso informático relevante en la actualidad. Puntualmente, esta característica se refiere a la capacidad de un sistema informático de otorgar un desempeño adecuado, en relación a los recursos computacionales bajo condiciones correctamente establecidas.

En este capítulo, se presentan las funciones que le permiten al sistema acelerar la reconstrucción de los materiales. Se comienza con el proceso de diezmado, que consiste en la reducción espacial de la imagen binarizada, permitiendo perder un 2 % de los detalles contenidos en la misma, pero obteniendo un beneficio con respecto al tiempo de procesamiento de hasta un 99 %. Después, se continúa con la exposición de la función encargada de realizar el volumen inicial, a partir de la información contenida en la matriz de la función de correlación de tamaño de poro, mismo volumen es utilizado por el recocido simulado como volumen de trabajo.

5.1. Diezmado

El proceso completo de diezmado presente en el sistema, se realiza mediante dos funciones que se basan en el trabajo realizado por Ledesma, Barbosa y Ortegón [71]. La primera función se encarga de normalizar la matriz de la función de correlación de CL y DP, y obtener un factor de reducción que permita mantener un 98 % de los detalles presentes en la imagen binarizada. La segunda, se encarga de utilizar dicho factor para definir hasta cuanto se reducirá espacialmente la imagen. A continuación, se inicia con la presentación detallada de la primera función, así como también, su fundamentación matemática.

La primera función, llamada $main\_normalization\_corr\_func\_2p$, se inicializa en conjunto de sus parámetros de entrada, que son: el ancho y alto de la imagen binarizada, así como también la función de correlación de DP y CL. Una vez declarada la función, es importante recalcar que el proceso de normalización que realiza ésta, se encarga de parametrizar, entre 0 y 1, la fracción cuantitativa de las longitudes presentes en la función de correlación de DP [71]. Esta parametrización se realiza tanto para la fase sólida como para la fase porosa. Matemáticamente se define de la siguiente manera:

$$S_j^{(2)}(pos) \longrightarrow \frac{S_j^{(2)}(pos) - \Phi_j^2}{\Phi_j - \Phi_j^2} \qquad \text{(Ecu. IV)}$$

Donde Φ es el total de píxeles en la fase j entre el total de todos los píxeles en la imagen, y de la misma manera, se ve declarado en la línea 4 del Código 5.1:

```
3   double[] normalization_corr_func_2p_1 = new double[real_length];
4   double fi_1 = (double)corr_func_2p_lp[0, 0] / (double)corr_func_2p_lp
        [4, 0];
5   double fi_1_squared = Math.Pow(fi_1, 2);
6   double denominator_fi_1 = fi_1 - fi_1_squared;
```

Código 5.1: Normalización Parte 1

La normalización, como se aprecia en la ecuación anterior (Ecu. IV) mediante la variable *pos*, se debe realizar para todas las posiciones en las dos fases de la matriz de la función de DP. Es en la línea 7 del Código 5.2, donde se declara dicho recorrido por toda la función, y en el siguiente par de líneas, se usa de manera directa la ecuación para ambas fases:

```
7   for (position = 0; position < real_length; position++)
8     normalization_corr_func_2p_1[position] = (((double)corr_func_2p_lp
          [0, position] / (double)corr_func_2p_lp[4, position]) -
          fi_1_squared) / denominator_fi_1;
9     normalization_corr_func_2p_0[position] = (((double)corr_func_2p_lp
          [2, position] / (double)corr_func_2p_lp[4, position]) -
          fi_0_squared) / denominator_fi_0;
```

Código 5.2: Normalización Parte 2

Conforme se realiza el avance del ciclo, y después de calcular los valores normalizados para la posición en turno, se lleva a cabo una condición simultánea. Como se puede apreciar en la línea 10 del Código 5.3, esta condición consiste en revisar si alguno de los resultados en las dos fases fue menor al 2 %, mismo porcentaje que representa el detalle máximo a perder de la imagen. En el momento en que se encuentre el primer valor menor al 2 %, independientemente de la fase en la que esté, se registra la posición en turno y el ciclo termina automáticamente:

```
10  if (normalization_corr_func_2p_1[position] < .02 ||
        normalization_corr_func_2p_0[position] < .02)
11  {
12    first_position = position;
13    position = real_length;
14  }
```

Código 5.3: Normalización Parte 3

Cuando la función encuentra la posición, significativa al 2 % de los valores normalizados, la regresa. Cabe recalcar que esta posición obtenida es utilizada como parámetro de entrada para la función encargada de reducir espacialmente la imagen mediante un cambio de resolución.

La segunda función comienza definiendo los parámetros de entrada, entre ellos, la variable *position*, resultante al proceso de normalización visto anteriormente. Después, el proceso continúa en el Código 5.4, con la definición de las variables encargadas de controlar el proceso de diezmado. Seguidamente, se realiza la explicación de cada una:

```
3   double m_z = Math.Ceiling((3 * (double)img_height) / position);
4   double z = Math.Floor((Math.Log((img_height / m_z), 2)));
5   int reduction_f = (int)Math.Pow(2, z);
6   byte[] decimation = new byte[(int)Math.Pow(reduced_length, 2)];
```

Código 5.4: Diezmado Parte 1

El diezmado se realiza usando una función bilineal, es decir, de cada ventana a reducir se calcula el promedio, y se lleva a cabo en función de un factor de reducción, representado por la variable *reduction_f* en la línea 5, mismo que servirá para definir la nueva resolución de la imagen [71]. Para obtener dicho factor, se realiza un cálculo con base en la longitud de correlación de la función de DP, donde dicha longitud representa el 98 % de los detalles, y en la función se refiere a ésta mediante la variable *position*. Puntualmente, el valor del factor de reducción se obtiene a partir de la siguiente ecuación elevada al cuadrado:

$$Z = \left\lfloor \log_2 \left(\frac{largo\_img}{M_Z} \right) \right\rfloor \qquad \text{(Ecu. V)}$$

Donde el valor de M_Z se obtiene a partir de la siguiente ecuación:

$$M_Z = \left\lceil \frac{3 * largo\_img}{position} \right\rceil \qquad \text{(Ecu. VI)}$$

Las ecuaciones (Ecu. V) y (Ecu. VI), se utilizan en las líneas 4 y 3, donde $\lfloor\ \rfloor$ es el piso y $\lceil\ \rceil$ es el techo respectivamente. Para concluir, la variable llamada *decimation* en la línea 6, es el vector que guardará la nueva imagen generada. Al finalizar el conteo del área en turno, y como se puede apreciar en la línea 12 del Código 5.5, se realiza una división para conocer si el promedio de píxeles fue mayormente blanco o negro, y entonces, asignar el resultado al vector *decimation*:

```
12    decimation[control_pixel] = (byte)(acomulator / (Math.Pow(reduction_f,
          2)) < .5 ? 0 : 1);
```

Código 5.5: Diezmado Parte 2

5.2. Generación del Volumen Inicial

La generación del volumen inicial a partir de una función de correlación, y no de manera aleatoria, le permite a la función encargada de la reconstrucción acelerar dicho proceso. A grandes rasgos, este volumen se crea mediante la inserción de esferas blancas sobre un volumen de trabajo completamente en la fase porosa, siendo la función de correlación de tamaño de poro quien se encarga de controlar tal inserción. A continuación, se realiza una explicación detallada de la función encargada de la generación del volumen inicial.

Se declara la función en conjunto de sus parámetros de entrada, los cuales son: la longitud de la imagen y la función de correlación de tamaño de poro. La variable *volume_control*, es el vector encargado de contener el volumen que servirá de referencia para definir la densidad de vóxeles contenidos a ciertos radios, su longitud es parcialmente definida por el radio máximo. Después, la variable *volume*, es el vector que contendrá el volumen de trabajo. Seguidamente, la variable $control\_corr\_func\_ps\_3d$, contendrá la función de correlación de TP, pero bajo una conversión que se explicará más adelante. La última variable que se presenta, llamada *volumetric_control*, se encarga de almacenar el resultado de la multiplicación entre la fracción de píxeles blancos de radio 0 y la longitud de la imagen.

Una vez se declaran las variables, se prosigue en el Código 5.6 con la transformación de la función de correlación de tamaño de poro a una versión 3D, que tiene como objetivo mantener la proporción de píxeles en ambas fases. Dicha transformación se realiza mediante la multiplicación entre el valor de la posición en turno de la función de correlación de DP en 2D, por un factor igual a la longitud de la imagen. Este factor se va reduciendo de dos en dos en cada iteración:

```
7   for (position_control = 0; position_control <= maximum_radius;
        position_control++)
8   {
9     control_corr_func_ps_3d[0, position_control] = (int)corr_func_ps[0,
          position_control] * variable_length > 0 ? corr_func_ps[0,
          position_control] * (UInt32)variable_length : 0;
10    ...
11    variable_length = variable_length - 2 > 0 ? variable_length - 2 : 0;
12  }
```

Código 5.6: Generación del Volumen Inicial Parte 1

Después, se declara el ciclo encargado de realizar el control de los radios de las esferas que se insertarán, comenzando con la inserción de esferas de radio máximo, y conforme el ciclo avance, se reduce en uno dicho radio hasta llegar a 0. En los otros 3 ciclos, anidados después del ciclo anterior, se procede a trabajar con el volumen de control, mediante la inserción de una sola esfera, comenzando con una de radio máximo.

Mediante el Código 5.7, mientras se recorre el volumen de control, se cambia de fase a los vóxeles que estén dentro del radio en turno:

```
17   volume_control[((maximum_radius + radius_layer) *
         length_volume_control * length_volume_control) + ((maximum_radius
         + radius_row) * length_volume_control + (maximum_radius +
         radius_column))] = (radius_layer * radius_layer + radius_row *
         radius_row + radius_column * radius_column) <= control_radius ? (
         byte)1 : (byte)0;
```

Código 5.7: Generación del Volumen Inicial Parte 2

Se prosigue con una condición, presente en la línea 18 del Código 5.8, encargada de conocer si existe o no alguna esfera con el radio en turno en la función de correlación de control. Si existe alguna, en la línea 19, se evalúa el volumen de control con la función de correlación de TP en 3D, esto se hace para saber la cantidad exacta de esferas bajo el radio de la más grande. Y en la siguiente línea, se inicializa el ciclo encargado de definir cuantas esferas se insertarán en el volumen de trabajo:

```
18   if (control_corr_func_ps_3d[0, position_control] > 0)
19     subtraction = main_corr_func_ps_3d(length_volume_control,
           length_volume_control, length_volume_control, volume_control);
20     for (value_control = 0; value_control < control_corr_func_ps_3d[0,
           position_control]; value_control++)
```

Código 5.8: Generación del Volumen Inicial Parte 3

Para llevar a cabo la inserción aleatoria de esferas, se inicia con la obtención de 3 valores que servirán de coordenadas para el posicionamiento central de la nueva esfera. La obtención de dichas coordenadas se limita a buscar sólo las que estén dentro de la dimensión del diámetro en turno. La finalidad de esto es para no acceder a vóxeles inexistentes.

Cuando las 3 posiciones están correctamente seleccionadas, se inicializan 3 ciclos que se encargarán de recorrer el volumen de trabajo en las 3 dimensiones.

Mientras los 3 ciclos anteriores recorren la extensión alrededor de la esfera, es mediante el Código 5.9 que se evalúa el vóxel en turno, y si

éste está dentro del radio su fase se cambia a un estado sólido, de lo contrario, se mantiene en la fase en la que se encuentre:

```
27   volume[((random_layer + radius_layer) * length * length) + ((
         random_row + radius_row) * length + (random_column + radius_column
         ))] = (radius_layer * radius_layer + radius_row * radius_row +
         radius_column * radius_column) <= control_radius ? (byte)1 :
         volume[((random_layer + radius_layer) * length * length) + ((
         random_row + radius_row) * length + (random_column + radius_column
         ))];
```

Código 5.9: Generación del Volumen Inicial Parte 4

Una vez se termina de insertar la esfera, se realiza la sustracción del número de esferas de radio menor contenidas en la que se insertó, con respecto a la función de correlación de control, y tal sustracción se aprecia en la línea 30 del Código 5.10. La finalidad de esta mecánica consiste en no repetir la inserción de esferas de radio menor contenidas en la insertada. Un claro ejemplo se presenta con una esfera de radio 1, que a su vez, contiene 7 esferas de radio 0:

```
28   for (subtraction_control = position_control - 1; subtraction_control
         >= 0; subtraction_control--)
29   {
30     control_corr_func_ps_3d[0, subtraction_control] = (int)
           control_corr_func_ps_3d[0, subtraction_control] - (int)
           subtraction[0, subtraction_control] > 0 ?
           control_corr_func_ps_3d[0, subtraction_control] - subtraction[0,
            subtraction_control] : 0;
31   }
```

Código 5.10: Generación del Volumen Inicial Parte 5

Cada vez que la inserción de esferas bajo cierto radio concluye, y antes de volver a iniciar el ciclo con un radio menor, es mediante el Código 5.11 que el volumen de control se restablece:

```
32   Array.Clear(volume_control, 0, volume_control.Length);
```

Código 5.11: Generación del Volumen Inicial Parte 6

Al terminar la inserción de todas las esferas en el volumen de trabajo, en el Código 5.12, se continúa con la nivelación entre la fracción volumétrica ideal, definida por la variable *volumetric_control*, y la primera fila de la función de correlación de TP en 3D del volumen de trabajo. La finalidad principal de dicha nivelación consiste en saber cuantos vóxeles

del volumen de trabajo sobran o faltan para coincidir con la fracción volumétrica:

```
33   volume_corr_func_ps_3d = main_corr_func_ps_3d(length, length, length,
         volume);
34   volumetric_overload = (int)volumetric_control - (int)
         volume_corr_func_ps_3d[0, 0];
```

Código 5.12: Generación del Volumen Inicial Parte 7

En función del resultado de la resta, en el Código 5.13, se accede a una condición que, mediante un ciclo, cambiará de fase vóxeles aleatorios hasta equilibrar el volumen de trabajo:

```
35   if (volumetric_overload < 0)
36     for (overload_control = 0; overload_control < volumetric_overload *
           -1;)
```

Código 5.13: Generación del Volumen Inicial Parte 8

El cambio de fase aleatorio se realiza en el Código 5.14. Se comienza en las líneas 37, 38 y 39, con la obtención de 3 coordenadas aleatorias dentro del volumen de trabajo. En la siguiente línea, se accede al vóxel para conocer la fase en la que se encuentra, y dependiendo de esto, en la línea 41, se decide si se cambia de fase o no:

```
37   random_layer = random.Next(0, length);
38   random_row = random.Next(0, length);
39   random_column = random.Next(0, length);
40   overload_control = volume[(random_layer * length * length) + (
         random_row * length) + random_column] == 1 ? overload_control + 1
         : overload_control;
41   volume[(random_layer * length * length) + (random_row * length) +
         random_column] = volume[(random_layer * length * length) + (
         random_row * length) + random_column] == 1 ? (byte)0 : volume[(
         random_layer * length * length) + (random_row * length) +
         random_column];
```

Código 5.14: Generación del Volumen Inicial Parte 9

Finalmente, después de la nivelación explicada anteriormente, la función regresa el volumen de trabajo sobre la que el recocido simulado reconstruirá la microestructura del material.

Capítulo 6

Pruebas y Resultados del Sistema

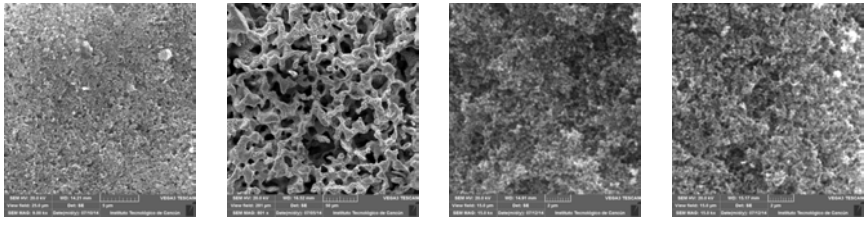
Cualquier sistema informático, independientemente de su finalidad, es susceptible a fallos o inconsistencias incluso en etapas muy avanzadas de desarrollo, sin embargo, la aplicación constante de pruebas a lo largo del proceso de creación, garantizan una etiqueta de calidad. Debido a esto, para la herramienta de reconstrucción de materiales que se presenta en esta tesis, la aplicación de dichas pruebas no fue la excepción, pues durante la creación de cada módulo se aseguró el flujo de información tanto en los parámetros de entrada como en la información que se devuelve. La importancia de realizar pruebas consistentes, fue pieza clave para asegurar la solución a la problemática vista en el primer capítulo de la tesis.

Dichas pruebas y resultados se expondrán a lo largo de este capítulo, comenzando con las pruebas de funcionamiento y desempeño, describiendo la metodología que se utilizó para cuantificar y medir la certeza de las reconstrucciones. Después, se presentan los resultados obtenidos a dichas pruebas, y cómo éstos se validan y utilizan para temas relacionados a la energía, mediante la discusión de los mismos.

6.1. Pruebas de Funcionamiento y Desempeño

Las pruebas de funcionamiento y desempeño realizadas sobre el sistema de reconstrucción, constituyeron la etapa final en el desarrollo de la herramienta. Debido a ello, la importancia de llevar a cabo una correcta planificación de dichas pruebas, garantizó la culminación exitosa de los objetivos previamente establecidos en la presente tesis.

Las pruebas consistieron en llevar a cabo una serie de reconstrucciones a partir de imágenes SEM de electrodos de celdas de combustible de diversos materiales. La selección de estas imágenes se basó en la variación de la morfología natural de los materiales, así como también, en distintos grados de heterogeneidad en cada una. Realizar la selección de muestras de esta manera, le permitió al sistema probar su consistencia resolutiva a diferentes distribuciones morfológicas. A continuación, en la Figura 6.1, se presentan las imágenes SEM que se utilizaron durante el proceso de pruebas, y seguidamente, se expone la constitución microestructural del electrodo que representa cada una:



a b c d

Figura 6.1: Imágenes SEM de Electrodos de Distintos Materiales

La imagen 6.1.a, nombrada 9kX_CCM3, es un electrodo constituido por platino sobre carbón mezclado con un polímero denominado nafión a un 20 %, depositado sobre una membrana. La imagen 6.1.b, nombrada 800X_C2H2O4, es un electrodo conformado por titanio puro con un ataque químico de ácido oxálico ($C_2H_2O_4$).

Las imágenes 6.1.c y 6.1.d, llamadas 15kX_HP1 y 15kX_HP3 respectivamente, son dos zonas distintas de un mismo electrodo también constituido por platino sobre carbón con la misma proporción de nafión, pero depositado sobre un difusor de gas.

6.1.1. Metodología de Aplicación de Pruebas

La metodología que se utilizó para la realización de las reconstrucciones de prueba, comenzó con la selección aleatoria de una zona cuadrada en cada imagen SEM para ser la muestra representativa de su respectivo electrodo. Cabe recalcar que cada zona cuadrada tuvo una dimensión de 100 píxeles por lado. Después de reconstruir las cuatro muestras originales, se hicieron dos reconstrucciones extra para cada una, pero con sus resoluciones al 50 % y 25 %, es decir, se reconstruyó cada muestra sin diezmar (100 píxeles por lado), seguidamente se llevó a cabo una segunda reconstrucción con el diezmado al 50 %, y por último, una tercera reconstrucción con el diezmado al 25 %. El número de reconstrucciones por muestra fue de 3, haciendo un total de 12 reconstrucciones obtenidas satisfactoriamente por el sistema durante la fase de pruebas, mismas que serán explicadas a detalle en la sección de resultados.

El sistema de reconstrucción de materiales estocásticos, genera una serie de parámetros cuantitativos de control mientras realiza la búsqueda de una solución óptima. Estos aspectos cuantitativos se analizaron durante la fase de pruebas, y sirvieron como indicadores de éxito, ya que se utilizaron para definir características como la eficiencia computacional y la consistencia existente en los resultados obtenidos. Estos indicadores fueron propuestos a partir de los requisitos y objetivos establecidos en la presente tesis. A continuación, se presentan los aspectos cuantitativos que fueron registrados en cada reconstrucción:

- Error final.
- Tiempo de procesamiento.
- Número de iteraciones.
- Coeficiente efectivo de transporte, calculado mediante un código externo desarrollado en el lenguaje de programación C.
- Comparación gráfica entre las funciones de correlación 2D y 3D.

Por último, el equipo de cómputo utilizado para las pruebas fue una laptop de la marca Alienware con las siguientes especificaciones técnicas:

- Modelo: Alienware 14.
- Sistema operativo: Windows 10 Home 64 bits.
- Procesador: Intel(R) Core(TM) i7-4700MQ a 2.40 GHz de 8 núcleos.
- Memoria RAM: 16 GB.
- Tarjeta gráfica: NVIDIA GeForce GTX 765M de 2 GB.
- Disco duro: 1000 GB.

6.2. Resultados

A continuación, se exponen las 12 reconstrucciones obtenidas como resultados ante las pruebas de funcionamiento y desempeño. Los resultados se expresan mediante un marco de presentación que comienza con la muestra original utilizada con dimensiones de 100 píxeles por lado, y después, se continúa con la misma muestra diezmada al 50 % y 25 %.

También, en cada reconstrucción, se presentan las vistas frontal y posterior del volumen reconstruido, y seguidamente, se muestra una tabla con los resultados numéricos correspondientes a la solución expuesta en turno, y finalmente, se presentan las 4 comparaciones gráficas correspondientes a las funciones de correlación 2D y 3D respectivamente de la solución, con sus dos fases posibles.

6.2.1. Electrodo de Platino sobre Carbón (9kX_CCM3)

La primera muestra se obtuvo a partir de la Figura 6.1.a, y la zona seleccionada aleatoriamente se presenta en la Figura 6.2:

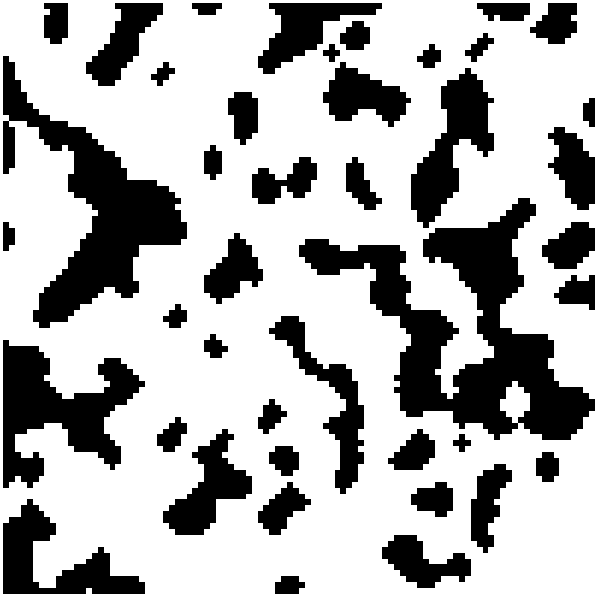
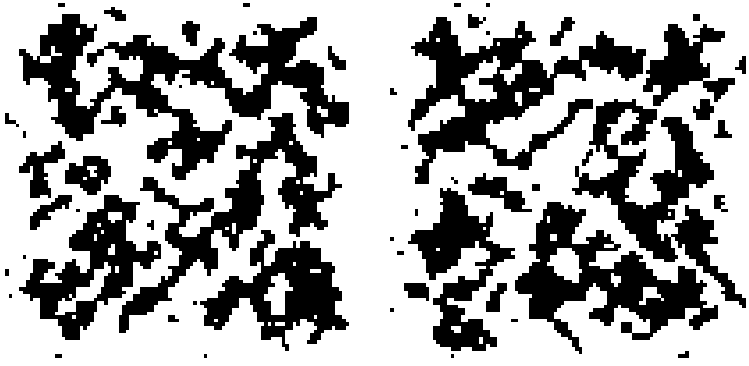


Figura 6.2: Muestra 9kX_CCM3 Sin Diezmar

A continuación, en la Figura 6.3, se presenta la vista frontal (a) y

posterior (b), a partir del volumen reconstruido:



a b

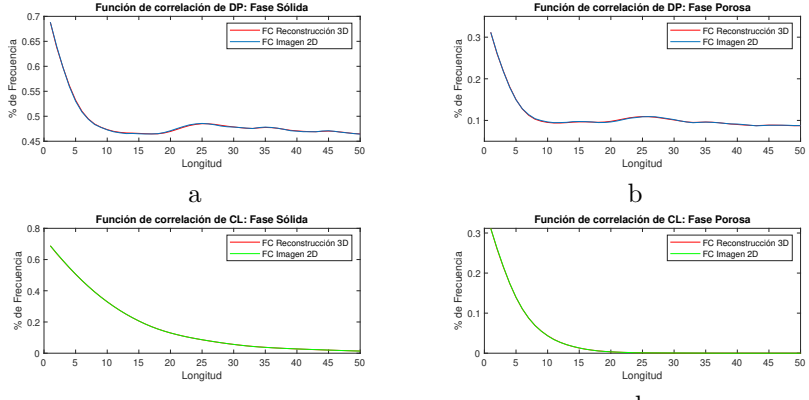
Figura 6.3: Vistas del Volumen Reconstruido

En la Tabla 6.1, se presentan los resultados numéricos obtenidos a partir de la reconstrucción de la muestra 9kX_CCM3 sin diezmar:

9kX_CCM3	Error	Tiempo	Iteraciones	CET
Sin diezmar	<1.00E-06	7 h 26 min 7 s	15,336,000	46.83

Tabla 6.1: Resultados: 9kX_CCM3 Sin Diezmar

En la Figura 6.4, se presentan las comparaciones gráficas entre las funciones de correlación 2D y 3D, en ambas fases:



a b c d

Figura 6.4: Comparación entre FC 2D y 3D

Muestra 9kX_CCM3 Diezmada al 50 %

En la Figura 6.5, se presenta la muestra 9kX_CCM3 diezmada al 50 %, obtenida a partir de la muestra original:

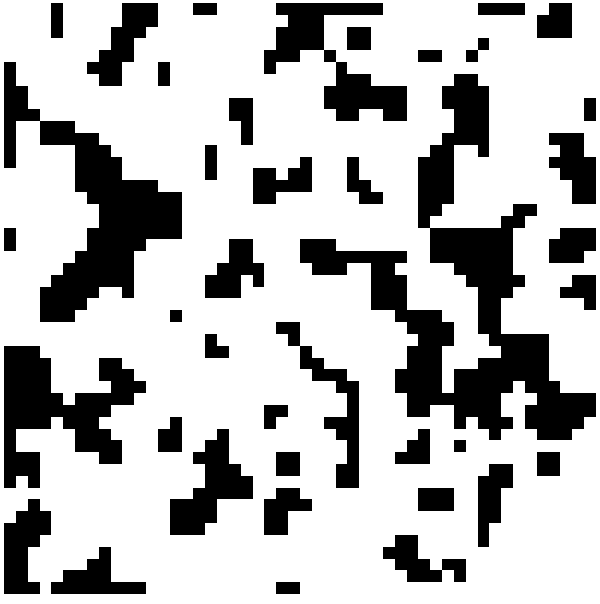


Figura 6.5: Muestra 9kX_CCM3 Diezmada al 50 %

A continuación, en la Figura 6.6, se presenta la vista frontal (a) y posterior (b), a partir del volumen reconstruido:

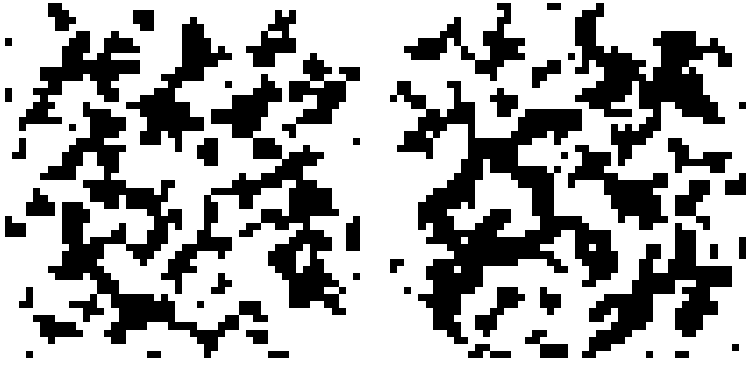


Figura 6.6: Vistas del Volumen Reconstruido

Comparando las Tablas 6.1 y 6.2, se aprecia que el diezmado al 50 %, le permite al sistema reducir 7 horas con 20 minutos la obtención de una solución óptima, con un CET similar bajo el mismo error.

9kX_CCM3	Error	Tiempo	Iteraciones	CET
Diezmado al 50 %	<1.00E-06	0 h 4 min 11 s	537,000	47.86

Tabla 6.2: Resultados: 9kX_CCM3 Diezmada al 50 %

En la Figura 6.7, se presentan las comparaciones gráficas entre las funciones de correlación 2D y 3D, en ambas fases:

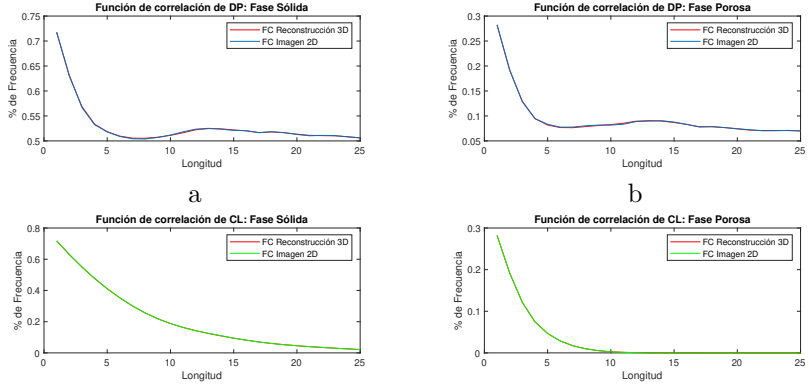


Figura 6.7: Comparación entre FC 2D y 3D

Muestra 9kX_CCM3 Diezmada al 25 %

En la Figura 6.8, se presenta la muestra 9kX_CCM3 diezmada al 25 %, obtenida a partir de la muestra original:

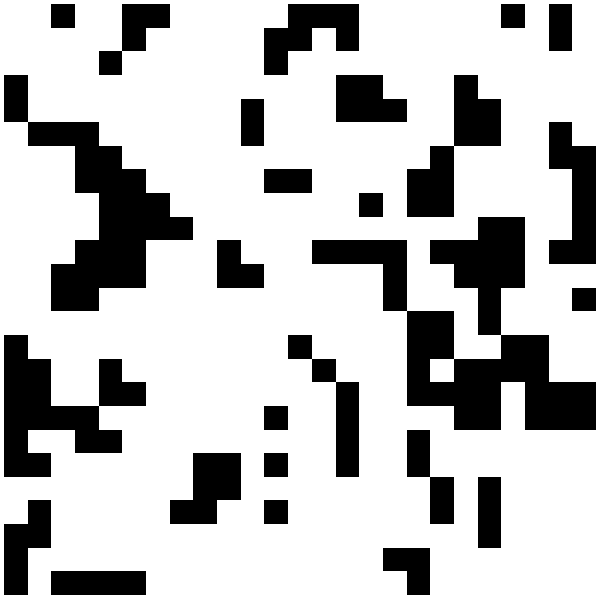
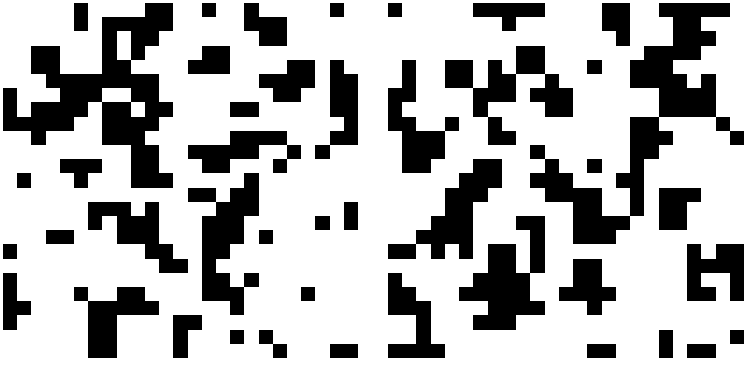


Figura 6.8: Muestra 9kX_CCM3 Diezmada al 25 %

A continuación, en la Figura 6.9, se presenta la vista frontal (a) y posterior (b), a partir del volumen reconstruido:



a b

Figura 6.9: Vistas del Volumen Reconstruido

Comparando las Tablas 6.1 y 6.3, se aprecia que el diezmado al 25 %, le permite al sistema reducir 7 horas con 26 minutos la obtención de una solución óptima, con un CET muy similar bajo el mismo error.

9kX_CCM3	Error	Tiempo	Iteraciones	CET
Diezmado al 25 %	<1.00E-06	0 h 0 min 13 s	57,000	46.59

Tabla 6.3: Resultados: 9kX_CCM3 Diezmada al 25 %

En la Figura 6.10, se presentan las comparaciones gráficas entre las funciones de correlación 2D y 3D, en ambas fases:

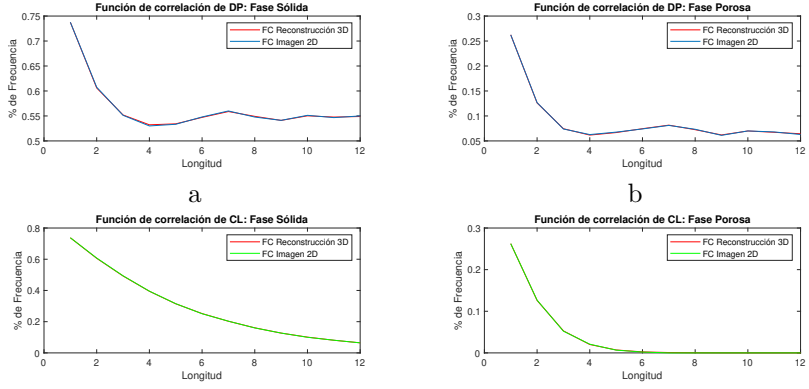


Figura 6.10: Comparación entre FC 2D y 3D

6.2.2. Electrodo de Titanio Puro (800X_C2H2O4)

La segunda muestra se obtuvo a partir de la Figura 6.1.b, y la zona seleccionada aleatoriamente se presenta en la Figura 6.11:

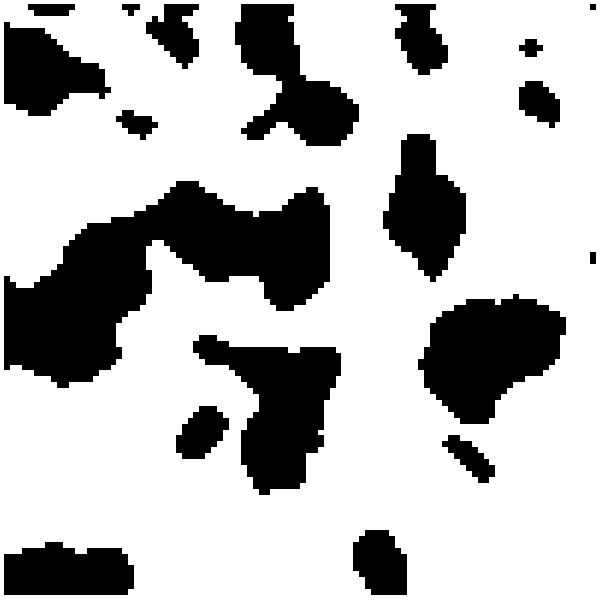


Figura 6.11: Muestra 800X_C2H2O4 Sin Diezmar

A continuación, en la Figura 6.12, se presenta la vista frontal (a) y posterior (b), a partir del volumen reconstruido:

a b

Figura 6.12: Vistas del Volumen Reconstruido

En la Tabla 6.4, se presentan los resultados numéricos obtenidos a partir de la reconstrucción de la muestra 800X_C2H2O4 sin diezmar:

800X_C2H2O4	Error	Tiempo	Iteraciones	CET
Sin diezmar	<1.00E-06	22 h 6 min 7 s	43,626,000	51.04

Tabla 6.4: Resultados: 800X_C2H2O4 Sin Diezmar

En la Figura 6.13, se presentan las comparaciones gráficas entre las funciones de correlación 2D y 3D, en ambas fases:

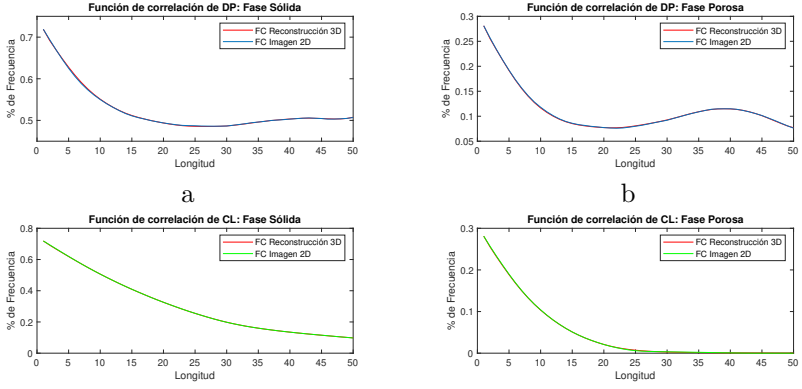


Figura 6.13: Comparación entre FC 2D y 3D

Muestra 800X_C2H2O4 Diezmada al 50 %

En la Figura 6.14, se presenta la muestra 800X_C2H2O4 diezmada al 50 %, obtenida a partir de la muestra original:

Figura 6.14: Muestra 800X_C2H2O4 Diezmada al 50 %

A continuación, en la Figura 6.15, se presenta la vista frontal (a) y posterior (b), a partir del volumen reconstruido:

a b

Figura 6.15: Vistas del Volumen Reconstruido

Comparando las Tablas 6.4 y 6.5, se aprecia que el diezmado al 50 %, le permite al sistema reducir 21 horas con 52 minutos la obtención de una solución óptima, con un CET muy similar bajo el mismo error.

800X_C2H2O4	Error	Tiempo	Iteraciones	CET
Diezmado al 50 %	<1.00E-06	0 h 14 min 45 s	1,809,000	51.18

Tabla 6.5: Resultados: 800X_C2H2O4 Diezmada al 50 %

En la Figura 6.16, se presentan las comparaciones gráficas entre las funciones de correlación 2D y 3D, en ambas fases:

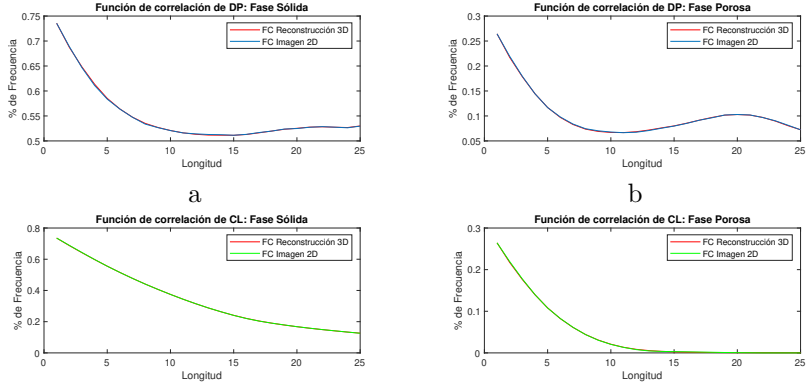


Figura 6.16: Comparación entre FC 2D y 3D

Muestra 800X_C2H2O4 Diezmada al 25 %

En la Figura 6.17, se presenta la muestra 800X_C2H2O4 diezmada al 25 %, obtenida a partir de la muestra original:

Figura 6.17: Muestra 800X_C2H2O4 Diezmada al 25 %

A continuación, en la Figura 6.18, se presenta la vista frontal (a) y posterior (b), a partir del volumen reconstruido:

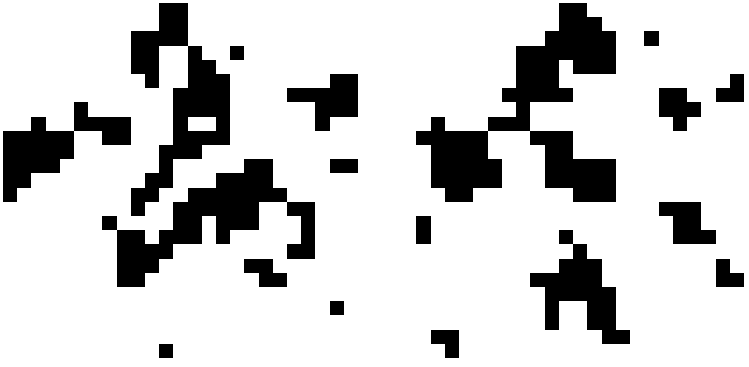


Figura 6.18: Vistas del Volumen Reconstruido

Comparando las Tablas 6.4 y 6.6, se aprecia que el diezmado al 25 %, le permite al sistema reducir 22 horas con 5 minutos la obtención de una solución óptima, con un CET distinto menor por 4 unidades bajo el mismo error.

800X_C2H2O4	Error	Tiempo	Iteraciones	CET
Diezmado al 25 %	<1.00E-06	0 h 0 min 18 s	96,000	47.71

Tabla 6.6: Resultados: 800X_C2H2O4 Diezmada al 25 %

En la Figura 6.19, se presentan las comparaciones gráficas entre las funciones de correlación 2D y 3D, en ambas fases:

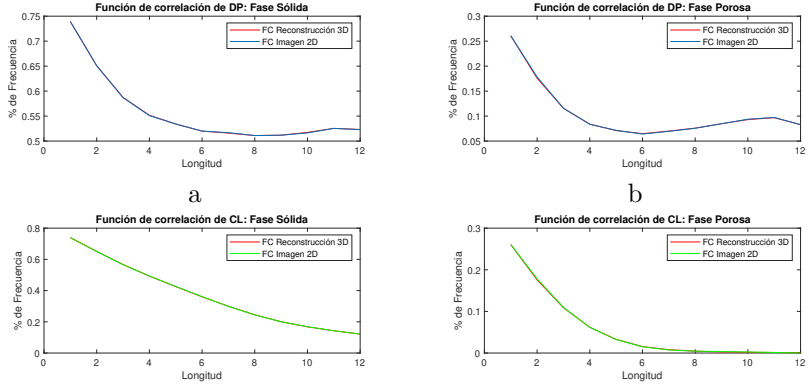


Figura 6.19: Comparación entre FC 2D y 3D

6.2.3. Electrodo de Platino sobre Carbón (15kX_HP1)

La tercera muestra se obtuvo a partir de la Figura 6.1.c, y la zona seleccionada aleatoriamente se presenta en la Figura 6.20:

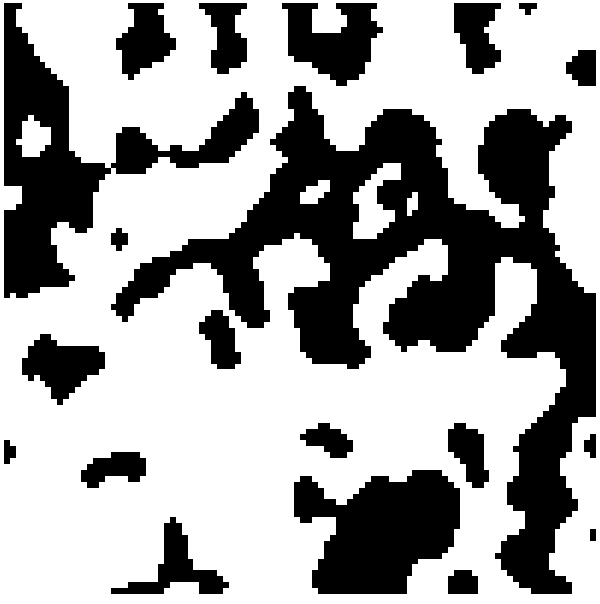


Figura 6.20: Muestra 15kX_HP1 Sin Diezmar

A continuación, en la Figura 6.21, se presenta la vista frontal (a) y posterior (b), a partir del volumen reconstruido:

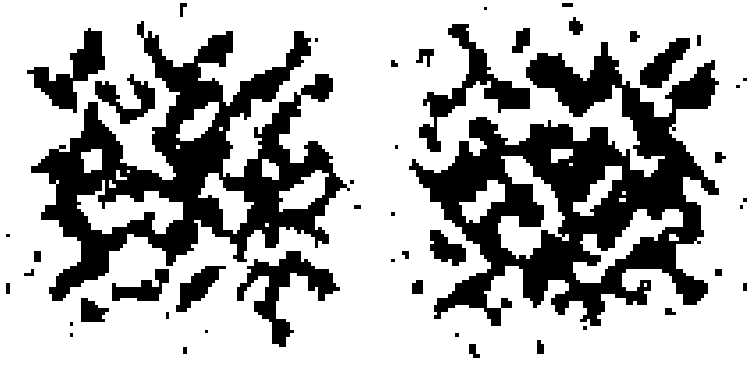


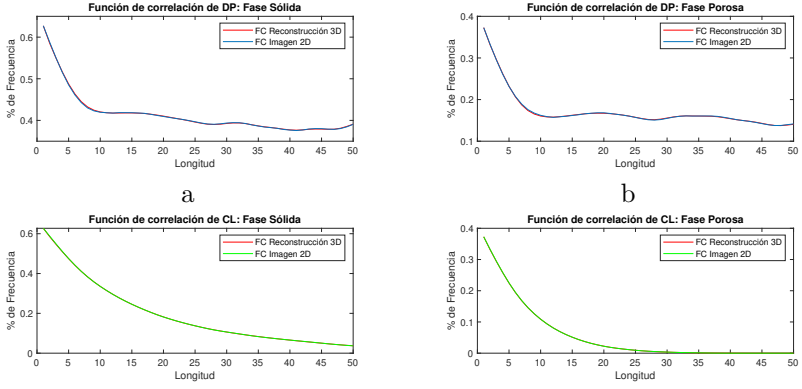
Figura 6.21: Vistas del Volumen Reconstruido

En la Tabla 6.7, se presentan los resultados numéricos obtenidos a partir de la reconstrucción de la muestra 15kX_HP1 sin diezmar:

15kX_HP1	Error	Tiempo	Iteraciones	CET
Sin diezmar	<1.00E-06	12 h 48 min 0 s	26,130,000	37.77

Tabla 6.7: Resultados: 15kX_HP1 Sin Diezmar

En la Figura 6.22, se presentan las comparaciones gráficas entre las funciones de correlación 2D y 3D, en ambas fases:



a b c d

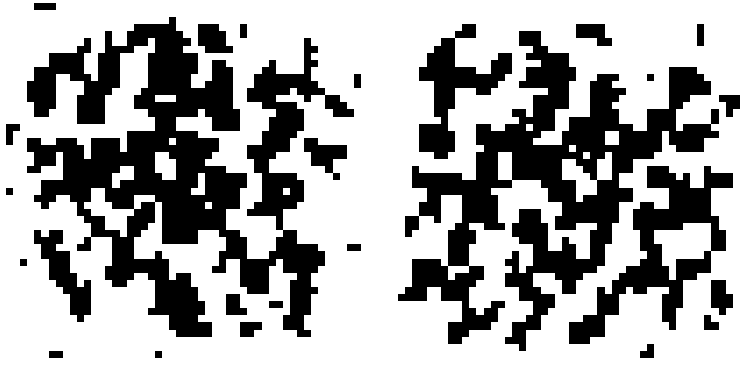
Figura 6.22: Comparación entre FC 2D y 3D

Muestra 15kX_HP1 Diezmada al 50 %

En la Figura 6.23, se presenta la muestra 15kX_HP1 diezmada al 50 %, obtenida a partir de la muestra original:

Figura 6.23: Muestra 15kX_HP1 Diezmada al 50 %

A continuación, en la Figura 6.24, se presenta la vista frontal (a) y posterior (b), a partir del volumen reconstruido:



a b

Figura 6.24: Vistas del Volumen Reconstruido

Comparando las Tablas 6.7 y 6.8, se aprecia que el diezmado al 50 %, le permite al sistema reducir 12 horas con 40 minutos la obtención de una solución óptima, con un CET mayor por 2 unidades bajo el mismo error.

15kX_HP1	Error	Tiempo	Iteraciones	CET
Diezmado al 50 %	<1.00E-06	0 h 8 min 3 s	984,000	39.79

Tabla 6.8: Resultados: 15kX_HP1 Diezmada al 50 %

En la Figura 6.25, se presentan las comparaciones gráficas entre las funciones de correlación 2D y 3D, en ambas fases:

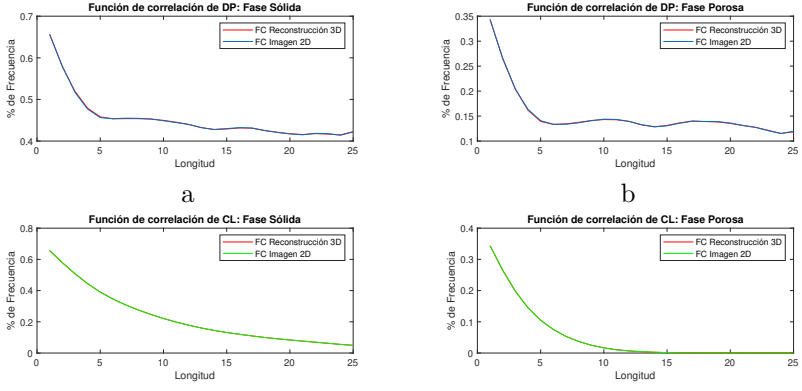


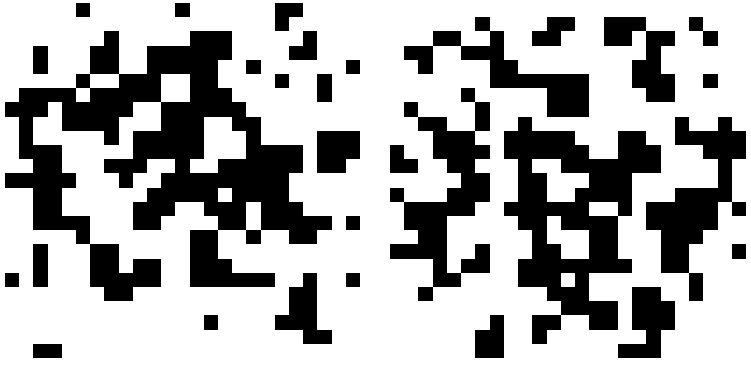
Figura 6.25: Comparación entre FC 2D y 3D

Muestra 15kX_HP1 Diezmada al 25 %

En la Figura 6.26, se presenta la muestra 15kX_HP1 diezmada al 25 %, obtenida a partir de la muestra original:

Figura 6.26: Muestra 15kX_HP1 Diezmada al 25 %

A continuación, en la Figura 6.27, se presenta la vista frontal (a) y posterior (b), a partir del volumen reconstruido:



a b

Figura 6.27: Vistas del Volumen Reconstruido

Comparando las Tablas 6.7 y 6.9, se aprecia que el diezmado al 25 %, le permite al sistema reducir 12 horas con 47 minutos la obtención de una solución óptima, con un CET completamente diferente menor por 5 unidades bajo el mismo error.

15kX_HP1	Error	Tiempo	Iteraciones	CET
Diezmado al 25 %	<1.00E-06	0 h 0 min 15 s	66,000	32.84

Tabla 6.9: Resultados: 15kX_HP1 Diezmada al 25 %

En la Figura 6.28, se presentan las comparaciones gráficas entre las funciones de correlación 2D y 3D, en ambas fases:

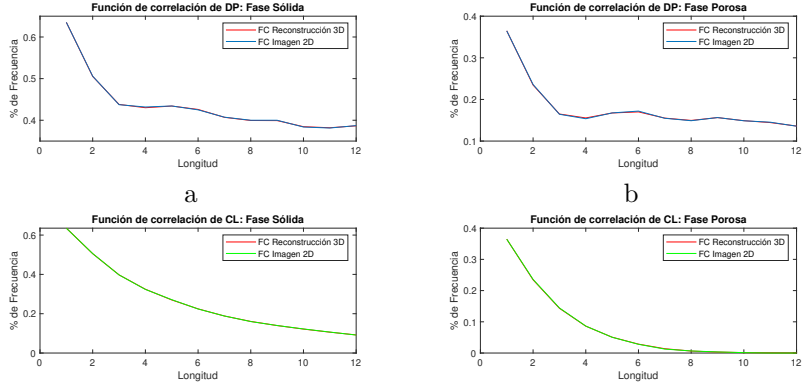


Figura 6.28: Comparación entre FC 2D y 3D

6.2.4. Electrodo de Platino sobre Carbón (15kX_HP3)

La cuarta muestra se obtuvo a partir de la Figura 6.1.d, y la zona seleccionada aleatoriamente se presenta en la Figura 6.29:

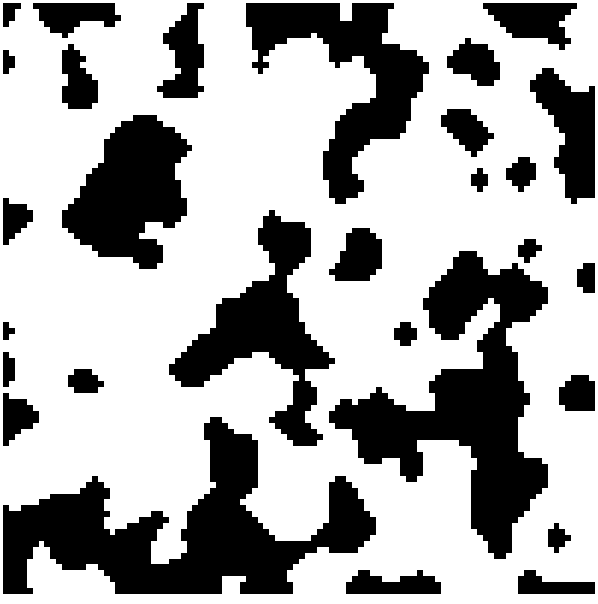
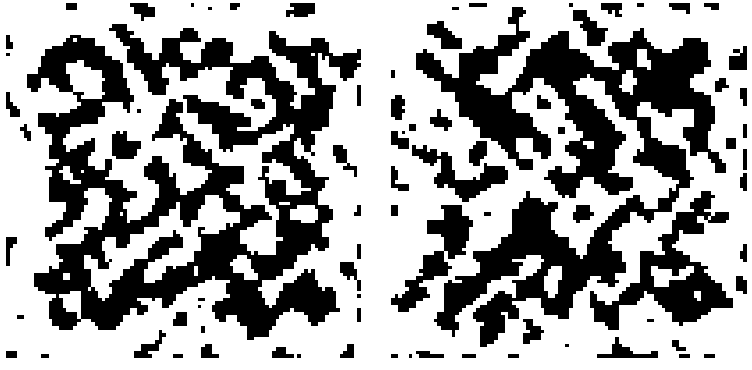


Figura 6.29: Muestra 15kX_HP3 Sin Diezmar

A continuación, en la Figura 6.30, se presenta la vista frontal (a) y posterior (b), a partir del volumen reconstruido:



a b

Figura 6.30: Vistas del Volumen Reconstruido

En la Tabla 6.10, se presentan los resultados numéricos obtenidos a partir de la reconstrucción de la muestra 15kX_HP3 sin diezmar:

15kX_HP3	Error	Tiempo	Iteraciones	CET
Sin diezmar	<1.00E-06	10 h 54 min 35 s	21,927,000	41.53

Tabla 6.10: Resultados: 15kX_HP3 Sin Diezmar

En la Figura 6.31, se presentan las comparaciones gráficas entre las funciones de correlación 2D y 3D, en ambas fases:

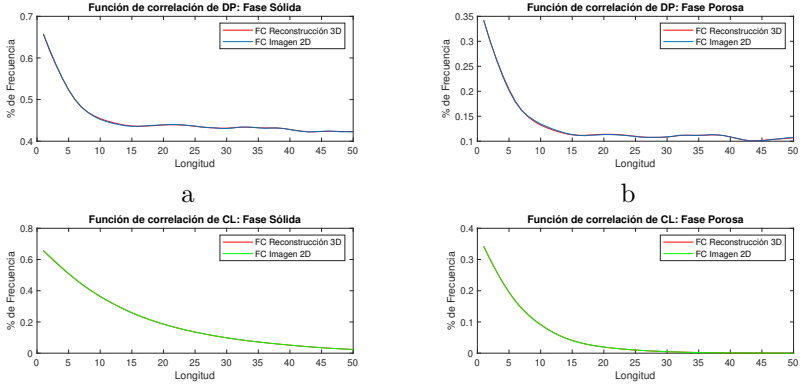


Figura 6.31: Comparación entre FC 2D y 3D

Muestra 15kX_HP3 Diezmada al 50 %

En la Figura 6.32, se presenta la muestra 15kX_HP3 diezmada al 50 %, obtenida a partir de la muestra original:

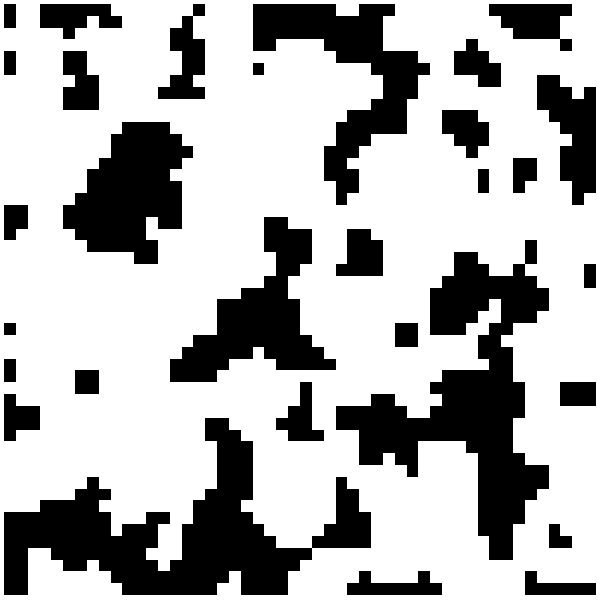
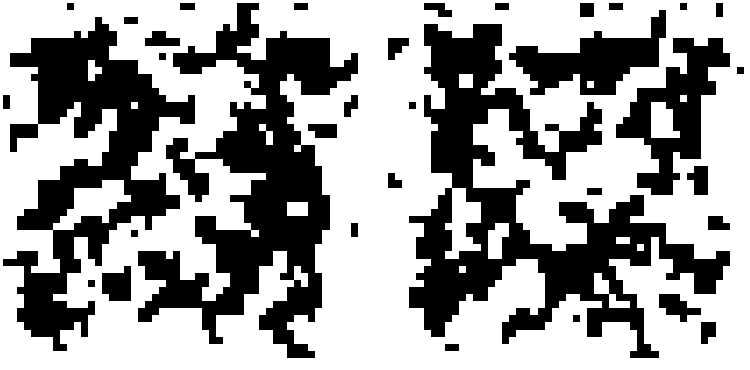


Figura 6.32: Muestra 15kX_HP3 Diezmada al 50 %

A continuación, en la Figura 6.33, se presenta la vista frontal (a) y posterior (b), a partir del volumen reconstruido:



a b

Figura 6.33: Vistas del Volumen Reconstruido

Comparando las Tablas 6.10 y 6.11, se aprecia que el diezmado al 50 %, le permite al sistema reducir 10 horas con 48 minutos la obtención de una solución óptima, con un CET mayor por 2 unidades bajo el mismo error.

15kX_HP3	Error	Tiempo	Iteraciones	CET
Diezmado al 50 %	<1.00E-06	0 h 6 min 43 s	864,000	43.85

Tabla 6.11: Resultados: 15kX_HP3 Diezmada al 50 %

En la Figura 6.34, se presentan las comparaciones gráficas entre las funciones de correlación 2D y 3D, en ambas fases:

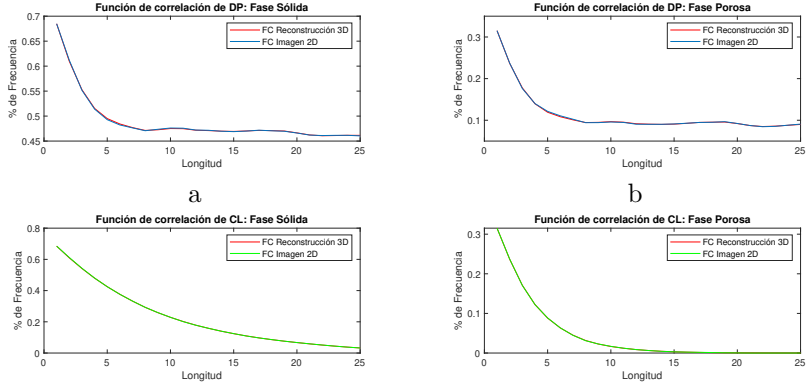


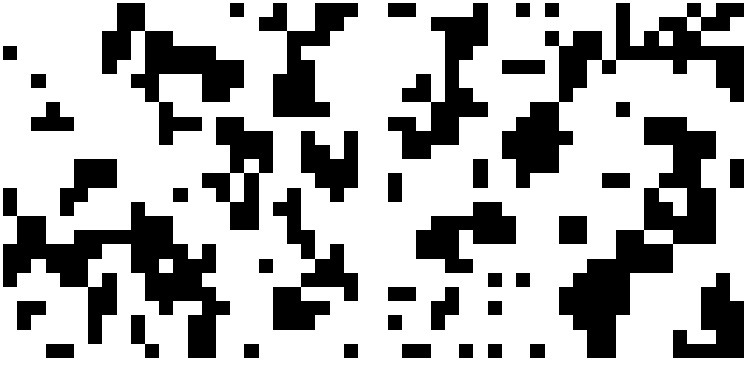
Figura 6.34: Comparación entre FC 2D y 3D

Muestra 15kX_HP3 Diezmada al 25 %

En la Figura 6.35, se presenta la muestra 15kX_HP3 diezmada al 25 %, obtenida a partir de la muestra original:

Figura 6.35: Muestra 15kX_HP3 Diezmada al 25 %

A continuación, en la Figura 6.36, se presenta la vista frontal (a) y posterior (b), a partir del volumen reconstruido:



a b

Figura 6.36: Vistas del Volumen Reconstruido

Comparando las Tablas 6.10 y 6.12, se aprecia que el diezmado al 25 %, le permite al sistema reducir 10 horas con 54 minutos la obtención de una solución óptima, con un CET similar bajo el mismo error.

15kX_HP3	Error	Tiempo	Iteraciones	CET
Diezmado al 25 %	<1.00E-06	0 h 0 min 9 s	60,000	42.22

Tabla 6.12: Resultados: 15kX_HP3 Diezmada al 25 %

En la Figura 6.37, se presentan las comparaciones gráficas entre las funciones de correlación 2D y 3D, en ambas fases:

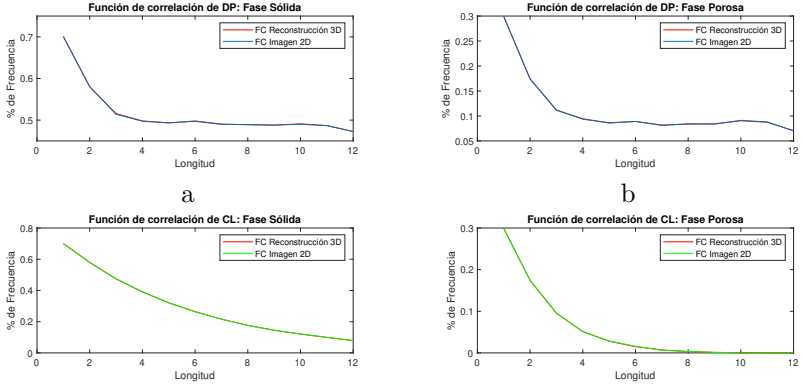


Figura 6.37: Comparación entre FC 2D y 3D

6.3. Discusión de Resultados

A continuación, se realiza la discusión de los resultados, mediante la comparación de cada conjunto de muestras, utilizando las gráficas obtenidas en cada reconstrucción. Cada análisis, incluye la reducción total del tiempo de procesamiento en cada diezmado, y señala la similitud o distinción con el CET de la muestra original. Siendo la similitud con éste, el indicador de haber obtenido una solución aceptable.

En la muestra 9kX_CCM3 (Figura 6.2), mediante la comparación de las funciones de correlación 2D y 3D, tanto para la muestra no diezmada (Figura 6.4), como para la muestra diezmada al 50 % (Figura 6.7) y la diezmada al 25 % (Figura 6.10), se aprecia que en los 3 casos, la similitud gráfica es alta, alcanzando el error mínimo. También, el CET obtenido a distintos niveles de diezmado, es muy similar al de la muestra original, demostrando así, que el proceso de diezmado para este caso, reduce el tiempo de procesamiento en un 99 %.

En la muestra 800X_C2H2O4 (Figura 6.11), mediante la comparación de las funciones de correlación 2D y 3D, tanto para la muestra no diezmada (Figura 6.13), como para la muestra diezmada al 50 % (Figura 6.16) y la diezmada al 25 % (Figura 6.19), se aprecia que en los 3 casos, la similitud gráfica es alta, alcanzando el error mínimo. El CET, solamente es muy similar en el diezmado al 50 %, mientras que para el diezmado al 25 %, se aleja por 4 unidades respecto a la muestra original. Debido a ello, se toma en cuenta la reconstrucción al 50 % como la solución óptima, reduciendo el tiempo de procesamiento en un 99 %.

En la muestra 15kX_HP1 (Figura 6.20), mediante la comparación de las funciones de correlación 2D y 3D, tanto para la muestra no diezmada (Figura 6.22), como para la muestra diezmada al 50 % (Figura 6.25) y la diezmada al 25 % (Figura 6.28), se aprecia que en los 3 casos, la similitud gráfica es alta, alcanzando el error mínimo. Pero en este caso, el CET es completamente distinto en los diferentes niveles de diezmado, considerando así, las soluciones obtenidas como fallidas.

En la muestra 15kX_HP3 (Figura 6.29), mediante la comparación de las funciones de correlación 2D y 3D, tanto para la muestra no diezmada (Figura 6.31), como para la muestra diezmada al 50 % (Figura 6.34) y la diezmada al 25 % (Figura 6.37), se aprecia que en los 3 casos, la similitud gráfica es alta, alcanzando el error mínimo. El CET, muestra una leve variabilidad de 2 unidades en promedio con respecto a la muestra original, con una reducción en el tiempo de procesamiento de un 99 %.

Capítulo 7

Conclusiones y Trabajo Futuro

En este trabajo de tesis, se propuso un modelo y una metodología de desarrollo para la reconstrucción de materiales estocásticos, logrando de esta manera el objetivo propuesto: la creación de una herramienta computacional, capaz de reconstruir materiales a partir de imágenes bidimensionales con una relación positiva entre el tiempo de procesamiento y la obtención de soluciones fiables, habiéndose prestado una especial atención al método de recocido simulado para realizar dichas reconstrucciones, y así, facilitar la adquisición de información microestructural de forma virtual para aplicaciones de energía.

Tanto el modelo como la metodología de desarrollo y las herramientas de creación de software, son instrumentos que se lograron integrar con éxito en el sistema de reconstrucción, y esto se comprobó mediante la evaluación meticulosa de las soluciones generadas.

A continuación, se describirán las conclusiones generales de la tesis y las líneas de trabajo a futuro que podrán continuar desarrollándose a partir de los resultados obtenidos en el presente trabajo.

7.1. Conclusiones

A lo largo del desarrollo de la presente tesis, el conocimiento que se adquirió en los diferentes capítulos, se sumarizan finalmente en la materialización de la herramienta computacional, debido a ello, se presenta una serie de conclusiones generales que recaban la retroalimentación completa de la tesis.

La revisión minuciosa del estado del arte, permitió visualizar el panorama teórico a lo largo del tiempo sobre el estudio de la microestructura de los materiales. La importancia de haber recabado y comprendido los estudios previos referentes a la reconstrucción, otorgó una visión de desarrollo clara y precisa, y de la misma manera, brindó la oportunidad de identificar ideas susceptibles a la mejora, es decir, la adquisición de nuevos puntos de vista fue importante al momento de abordar las soluciones a los retos que se presentaron. También, se revisó el trasfondo científico que rodea al proceso de reconstrucción antes de entrar a temas específicos de software, permitiendo conocer el camino existente entre las características de las distintas disciplinas que intervienen en dicho proceso. Asimismo, la tarea de relacionar los diversos ambientes científicos para unificar el resultado en una herramienta, sirvió como contexto teórico y práctico multidisciplinario para las bases mecatrónicas.

Después, gracias a un marco de trabajo de software correctamente definido, se logró la formulación de una estrategia de desarrollo eficiente, acorde a las necesidades y requerimientos de un software científico. Esta característica adquirida durante el desarrollo de la tesis, se encargó de permitir el involucramiento constante de los usuarios finales, siendo éstos los investigadores, quienes se encargaron de revisar el desarrollo completo de la herramienta. Seguidamente, la programación, implementación y explicación de los algoritmos, permitió comprobar la incorporación entre el recocido simulado y las funciones de correlación para realizar reconstrucciones. También, se comprobó la mejora en la implementación del recocido simulado mediante mecanismos programados, como son: el diezmado, el volumen inicial de trabajo no aleatorio y la actualización de las funciones de correlación para un sólo píxel.

Por último, con la obtención de resultados, la integración de los módulos dentro de la herramienta, demostró su capacidad de resolución ante las pruebas de funcionamiento y desempeño. A continuación, se realiza un recuento de los resultados y beneficios obtenidos:

- Los resultados numéricos obtenidos, disminuyen el panorama de incertidumbre ante las técnicas utilizadas para la realización de reconstrucciones cada vez más fiables mediante la variación de parámetros, como son: el error mínimo, la velocidad en la disminución de temperatura y la distribución de fases de las muestras.
- El módulo encargado de realizar la reconstrucción de materiales mediante el recocido simulado, permite la posibilidad de ser mejorado, estudiado o reutilizado para alguna otra herramienta informática con finalidad científica.
- La reutilización de los módulos independientes, ya sea para mejorarlos o expandir la cadena de procesos afines a la reconstrucción de materiales, brinda la oportunidad de integrar diferentes técnicas para probar soluciones a partir de otros puntos de vista científicos.

7.2. Trabajo Futuro

Dentro del trabajo de investigación realizado y documentado en la presente tesis, es importante identificar las líneas de trabajo a futuro para brindar una continuidad al esfuerzo invertido, y así, contribuir a la expansión del conocimiento científico sobre la reconstrucción de materiales estocásticos. Esta tesis permite la existencia de ideas que fungen como posibles propuestas de desarrollo, algunas de ellas están directamente relacionadas a componentes internos de la herramienta, mientras que otras son el resultado del análisis a posibles extensiones tanto gráficas como de control. A continuación, se presenta una serie de ideas, que por exceder el alcance de la tesis no fueron temas tratados con la suficiente profundidad:

- Binarizado mediante SVM.
- El desarrollo de un módulo, encargado de utilizar al volumen reconstruido para realizar la aplicación de métodos matemáticos y obtener valores que aporten, directamente desde la herramienta, información relevante en términos de energía, como puede ser el cálculo del coeficiente efectivo de transporte.
- Mejoras en la presentación gráfica y de control, para permitir la amplitud de funciones directamente desde la interfaz de la herramienta, como por ejemplo, decidir la precisión de la solución.

Bibliografía

[1] A. C. Lloyd, "The California fuel cell partnership: an avenue to clean air," *Journal of Power Sources*, vol. 86, págs. 57–60, mar. 2000.

[2] G. W. Crabtree, M. S. Dresselhaus, y M. V. Buchanan, "The Hydrogen Economy," *Physics Today*, vol. 57, núm. 12, págs. 39–44, 2004.

[3] R. Barbosa, J. Andaverde, B. Escobar, y U. Cano, "Stochastic reconstruction and a scaling method to determine effective transport coefficients of a proton exchange membrane fuel cell catalyst layer," *Journal of Power Sources*, vol. 196, págs. 1248–1257, feb. 2011.

[4] D. A. Coker y S. Torquato, "Extraction of morphological quantities from a digitized medium," *Journal of Applied Physics*, vol. 77, págs. 6087–6099, jun. 1995.

[5] S. Prager, "Viscous Flow through Porous Media," *Physics of Fluids*, vol. 4, pág. 1477, dic. 1961.

[6] S. Prager, "Improved Variational Bounds on Some Bulk Properties of a Two-Phase Random Medium," *The Journal of Chemical Physics*, vol. 50, págs. 4305–4312, may. 1969.

[7] M. Doi, "A New Variational Approach to the Diffusion and the Flow Problem in Porous Media," *Journal of the Physical Society of Japan*, vol. 40, págs. 567–572, feb. 1976.

[8] G. W. Milton, "Bounds on the Electromagnetic, Elastic, and Other Properties of Two-Component Composites," *Physical Review Letters*, vol. 46, págs. 542–545, feb. 1981.

[9] J. G. Berryman, "Measurement of spatial correlation functions using image processing techniques," *Journal of Applied Physics*, vol. 57, págs. 2374–2384, abr. 1985.

[10] J. G. Berryman y S. C. Blair, "Use of digital image analysis to estimate fluid permeability of porous materials: Application of two-point correlation functions," *Journal of Applied Physics*, vol. 60, págs. 1930–1938, sep. 1986.

[11] M. J. Beran, *Statistical continuum theories*. New York: Interscience Publishers, 1968.

[12] G. W. Milton, "Multicomponent composites, electrical networks and new types of continued fraction I," *Communications in Mathematical Physics*, vol. 111, págs. 281–327, jun. 1987.

[13] G. W. Milton, "Multicomponent composites, electrical networks and new types of continued fraction II," *Communications in Mathematical Physics*, vol. 111, págs. 329–372, sep. 1987.

[14] S. Torquato, "Random Heterogeneous Media: Microstructure and Improved Bounds on Effective Properties," *Applied Mechanics Reviews*, vol. 44, pág. 37, feb. 1991.

[15] E. Patelli y G. Schuëller, "On optimization techniques to reconstruct microstructures of random heterogeneous media," *Computational Materials Science*, vol. 45, págs. 536–549, abr. 2009.

[16] S. Torquato, *Random Heterogeneous Materials*, vol. 16 of *Interdisciplinary Applied Mathematics*. New York, NY: Springer New York, 2002.

[17] S. Sankaran y N. Zabaras, "A maximum entropy approach for property prediction of random microstructures," *Acta Materialia*, vol. 54, págs. 2265–2276, may. 2006.

[18] L. M. Pant, S. K. Mitra, y M. Secanell, "Multigrid hierarchical simulated annealing method for reconstructing heterogeneous media," *Physical Review E*, vol. 92, pág. 063303, dic. 2015.

[19] X. Zhao, J. Yao, e Y. Yi, "A new stochastic method of reconstructing porous media," *Transport in Porous Media*, vol. 69, págs. 1–11, jun. 2007.

[20] T. Tang, Q. Teng, X. He, y D. Luo, "A pixel selection rule based on the number of different-phase neighbours for the simulated annealing reconstruction of sandstone microstructure," *Journal of Microscopy*, vol. 234, págs. 262–268, jun. 2009.

[21] L. M. Pant, S. K. Mitra, y M. Secanell, "Stochastic reconstruction using multiple correlation functions with different-phase-neighbor-based pixel selection," *Physical Review E*, vol. 90, pág. 023306, ago. 2014.

[22] C. Xu, S. Gao, y M. Li, "A novel PCA-based microstructure descriptor for heterogeneous material design," *Computational Materials Science*, vol. 130, págs. 39–49, abr. 2017.

[23] B. L. Adams, S. Kalidindi, y D. T. Fullwood, *Microstructure sensitive design for performance optimization*. Butterworth-Heinemann, 2013.

[24] H. Xu, Y. Li, C. Brinson, y W. Chen, "Descriptor-Based Methodology for Designing Heterogeneous Microstructural Materials System," en *Volume 3A: 39th Design Automation Conference*, pág. V03AT03A049, ASME, ago. 2013.

[25] V. Sundararaghavan y N. Zabaras, "Representation and Classification of Microstructures using Statistical Learning Techniques," en *AIP Conference Proceedings*, vol. 712, págs. 98–102, AIP, jul. 2004.

[26] B. Jenkins, R. Lovel, y J. Thurlby, "Quantification of Iron Ore Sinter Structure by Optical Image Analysis," en *MiCon 90: Advances in Video Technology for Microstructural Control*, págs. 292–299, 100 Barr Harbor Drive, PO Box C700, West Conshohocken, PA 19428-2959: ASTM International, 1991.

[27] M. Tojima, T. Suzuki, y F. Kobayashi, "Classification of graphite shapes in cast irons using neural networks," en *International Conference on Computer-Assisted Materials Design and Process Simulation(COMMP'93)*, (Iron and Steel Institute, Tokyo, Japan), págs. 463–467, 1993.

[28] C. L. Y. Yeong y S. Torquato, "Reconstructing random media," *Physical Review E*, vol. 57, págs. 495–506, ene. 1998.

[29] V. Sundararaghavan y N. Zabaras, "Classification and reconstruction of three-dimensional microstructures using support vector machines," *Computational Materials Science*, vol. 32, págs. 223–239, feb. 2005.

[30] J. Aldazabal, A. Martín-Meizoso, y J. Martínez-Esnaola, "Simulation of liquid phase sintering using the Monte Carlo method," *Materials Science and Engineering: A*, vol. 365, págs. 151–155, ene. 2004.

[31] N. Sadati, M. Zamani, y H. R. F. Mahdavian, "Hybrid Particle Swarm-Based-Simulated Annealing Optimization Techniques," en *IECON 2006 - 32nd Annual Conference on IEEE Industrial Electronics*, págs. 644–648, IEEE, nov. 2006.

[32] J. Kennedy y R. Eberhart, "Particle swarm optimization," en *Proceedings of ICNN'95 - International Conference on Neural Networks*, vol. 4, págs. 1942–1948, IEEE, 1995.

[33] R. C. Eberhart, P. K. Simpson, y R. W. Dobbins, *Computational intelligence PC tools*. AP Professional, 1996.

[34] P. J. Angeline, "Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences," en *Proceedings of the 7th International Conference on Evolutionary Programming VII*, págs. 601–610, Springer, Berlin, Heidelberg, 1998.

[35] R. C. Eberhart e Y. Shi, "Comparison between genetic algorithms and particle swarm optimization," en *Proceedings of the 7th International Conference on Evolutionary Programming VII*, págs. 611–616, Springer, Berlin, Heidelberg, 1998.

[36] J. Kennedy, "The behavior of particles," en *Evolutionary Programming VII*, págs. 579–589, Springer, Berlin, Heidelberg, 1998.

[37] J. F. Kennedy, R. C. Eberhart, e Y. Shi, *Swarm intelligence*. Morgan Kaufmann Publishers, 2001.

[38] Z. Jiang, W. Chen, y C. Burkhart, "Efficient 3D porous microstructure reconstruction via Gaussian random field and hybrid optimization," *Journal of Microscopy*, vol. 252, págs. 135–148, nov. 2013.

[39] A. P. Roberts y M. Teubner, "Transport properties of heterogeneous materials derived from Gaussian random fields: Bounds and simulation," *Physical Review E*, vol. 51, págs. 4141–4154, may. 1995.

[40] A. Roberts, "Statistical reconstruction of three-dimensional porous media from two-dimensional images," *Physical Review E*, vol. 56, págs. 3203–3212, sep. 1997.

[41] M. G. Rozman y M. Utz, "Efficient reconstruction of multiphase morphologies from correlation functions," *Physical Review E*, vol. 63, págs. 1–8, may. 2001.

[42] S. Torquato, "Necessary Conditions on Realizable Two-Point Correlation Functions of Random Media," *Industrial & Engineering Chemistry Research*, vol. 45, núm. 21, págs. 6923–6928, 2006.

[43] Y. Jiao, F. H. Stillinger, y S. Torquato, "Modeling heterogeneous materials via two-point correlation functions: Basic principles," *Physical Review E*, vol. 76, págs. 1–15, sep. 2007.

[44] Y. Jiao, F. H. Stillinger, y S. Torquato, "Modeling heterogeneous materials via two-point correlation functions. II. Algorithmic details and applications," *Physical Review E*, vol. 77, págs. 1–15, mar. 2008.

[45] M. Talukdar, O. Torsaeter, M. Ioannidis, y J. Howard, "Stochastic reconstruction, 3D characterization and network modeling of chalk," *Journal of Petroleum Science and Engineering*, vol. 35, págs. 1–21, jul. 2002.

[46] W. Olchawa, R. Piasecki, R. Wiśniowski, y D. Frączek, "Low-cost approximate reconstructing of heterogeneous microstructures," *Computational Materials Science*, vol. 123, págs. 26–30, oct. 2016.

[47] J. M. Gago, "La microscopía para el estudio de materiales y láminas delgadas," en *Láminas delgadas y recubrimientos: preparación, propiedades y aplicaciones*, cap. 19, págs. 495–513, CSIC, 2003.

[48] S. Günther, B. Kaulich, L. Gregoratti, y M. Kiskinova, "Photoelectron microscopy and applications in surface and materials science," *Progress in Surface Science*, vol. 70, págs. 187–260, jul. 2002.

[49] R. Bostanabad, Y. Zhang, X. Li, T. Kearney, L. C. Brinson, D. W. Apley, W. K. Liu, y W. Chen, "Computational microstructure characterization and reconstruction: Review of the state-of-the-art techniques," *Progress in Materials Science*, vol. 95, págs. 1–41, jun. 2018.

[50] R. Magro, "Binarización de imágenes digitales y su algoritmia como herramienta aplicada a la ilustración entomológica," *Boletín de la Sociedad Entomológica Aragonesa*, vol. 53, págs. 443–464, 2013.

[51] S. Torquato, J. D. Beasley, e Y. C. Chiew, "Two-point cluster function for continuum percolation," *The Journal of Chemical Physics*, vol. 88, págs. 6540–6547, may. 1988.

[52] B. Lu y S. Torquato, "Lineal-path function for random heterogeneous materials," *Physical Review A*, vol. 45, págs. 922–929, ene. 1992.

[53] S. Torquato y B. Lu, "Chord-length distribution function for two-phase random media," *Physical Review E*, vol. 47, págs. 2950–2953, abr. 1993.

[54] Y. Jiao, F. H. Stillinger, y S. Torquato, "A superior descriptor of random textures and its predictive capacity," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 106, págs. 17634–9, oct. 2009.

[55] R. Bostanabad, A. T. Bui, W. Xie, D. W. Apley, y W. Chen, "Stochastic microstructure characterization and reconstruction via supervised learning," *Acta Materialia*, vol. 103, págs. 89–102, ene. 2016.

[56] A. E. Scheidegger, *The physics of flow through porous media*. Toronto: University of Toronto Press, 1974.

[57] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, y E. Teller, "Equation of State Calculations by Fast Computing Machines," *The Journal of Chemical Physics*, vol. 21, págs. 1087–1092, jun. 1953.

[58] S. Kirkpatrick, C. D. Gelatt, y M. P. Vecchi, "Optimization by simulated annealing," *Science (New York, N.Y.)*, vol. 220, págs. 671–680, may. 1983.

[59] S. Kirkpatrick, "Optimization by simulated annealing: Quantitative studies," *Journal of Statistical Physics*, vol. 34, págs. 975–986, mar. 1984.

[60] R. H. J. M. Otten y L. P. P. P. Ginneken, *The Annealing Algorithm*. Springer US, 1989.

[61] J. C. Spall, *Introduction to stochastic search and optimization: estimation, simulation, and control*. Wiley-Interscience, 2003.

[62] A. P. Engelbrecht, *Computational intelligence: an introduction*. John Wiley & Sons, 2007.

[63] F. J. Ceballos Sierra, *El lenguaje de programación C#*. Ra-Ma, 2001.

[64] L. Joyanes Aguilar y M. Fernández Azuela, *C#: Manual de programación*. McGraw-Hill, 2002.

[65] S. Logan, *Gtk+ programming in C*. Prentice Hall, 2002.

[66] A. Krause, *Foundations of GTK+ development*. Apress, 2007.

[67] W. Smith, *Learning Xamarin Studio*. Packt Publishing, 2014.

[68] V. Fernández Alarcón, *Desarrollo de sistemas de información: una metodología basada en el modelado*. UPC, 2010.

[69] R. Noriega Martínez, *El Proceso de Desarrollo de Software*. IT Campus Academy, 2017.

[70] E. L. Lehmann y G. Casella, *Theory of point estimation*. Springer, 1998.

[71] R. Ledesma, R. Barbosa, y J. Ortegón, "Effect of the image resolution on the statistical descriptors of heterogeneous media," *Physical Review E*, vol. 97, pág. 023304, feb. 2018.

Anexo A: Código del Sistema de Reconstrucción

```
1   using System;
2   using System.Diagnostics;
3   using Gtk;
4   using System.Drawing;
5   using System.IO;
6   using System.Collections.Generic;
7
8   public partial class MainWindow : Gtk.Window
9   {
10    public MainWindow() : base(Gtk.WindowType.Toplevel)
11    {
12      Build();
13    }
14
15    protected void OnDeleteEvent(object sender, DeleteEventArgs a)
16    {
17      Application.Quit();
18      a.RetVal = true;
19    }
20
21    protected void OnButton1Clicked(object sender, EventArgs e)
22    {
23      Stopwatch stopwatch = new Stopwatch();
24      FileChooserDialog filechooser = new FileChooserDialog("Seleccionar imagen...",
            this, FileChooserAction.Open, "Cancelar", ResponseType.Cancel, "Abrir",
          ResponseType.Accept);
25      if (filechooser.Run() == (int)ResponseType.Accept)
26      {
27        stopwatch.Start();
28        System.IO.FileStream file = System.IO.File.OpenRead(filechooser.Filename);
29        image1.Pixbuf = new Gdk.Pixbuf(file.Name);
30        //--Binarización--//
31        Bitmap original_img = new Bitmap(file.Name);
32        int x, y;
33        int umbral = 127;
34        byte[] pixels = new byte[original_img.Height * original_img.Width];
35        int pixels_cont = 0;
36        byte gray, value, value_pixel;
37        Color pixelColor;
38        for (y = 0; y < original_img.Height; y++)
39        {
40          for (x = 0; x < original_img.Width; x++)
41          {
42            //Color del píxel
43            pixelColor = original_img.GetPixel(x, y);
44            //Escala de grises
45            gray = (byte)(pixelColor.R * 0.3f + pixelColor.G * 0.59f + pixelColor.B
                  * 0.11f);
46            //Blanco o negro
```

```
47              value = 0;
48              value_pixel = 0;
49              if (gray > umbral)
50              {
51                value = 255;
52                value_pixel = 1;
53              }
54              //Asignar el valor al vector de píxeles
55              pixels[pixels_cont] = value_pixel;
56              pixels_cont++;
57              //Asignar el nuevo color
58              Color newColor = System.Drawing.Color.FromArgb(value, value, value);
59              original_img.SetPixel(x, y, newColor);
60            }
61          }
62          ///--Función de correlación: Camino lineal y dos puntos--///
63          UInt32[,] corr_func_2p_lp = main_corr_func_2p_lp(original_img.Width,
                original_img.Height, pixels);
64          ///--Normalización de la FC: Dos puntos--///
65          int position = main_normalization_corr_func_2p(original_img.Width,
                original_img.Height, corr_func_2p_lp);
66          ///--Diezmado--///
67          pixels = main_decimation(position, original_img.Width, original_img.Height,
                pixels);
68          ///--Función de correlación: Camino lineal y dos puntos--///
69          corr_func_2p_lp = main_corr_func_2p_lp((int)Math.Sqrt(pixels.Length), (int)
                Math.Sqrt(pixels.Length), pixels);
70          ///--Función de correlación: Tamaño de poro--///
71          UInt32[,] corr_func_ps = main_corr_func_ps((int)Math.Sqrt(pixels.Length), (
                int)Math.Sqrt(pixels.Length), pixels);
72          ///--Volumen 3D--///
73          byte[] pre_volume = main_volume_corr_func_ps((int)Math.Sqrt(pixels.Length),
                corr_func_ps);
74          ///--Reconstrucción--///
75          byte[] reconstruction = simulated_annealing((int)Math.Sqrt(pixels.Length),
                pre_volume, corr_func_2p_lp);
76          ///--Genera imagen original--///
77          original_img.Save(System.IO.Path.GetFileNameWithoutExtension(filechooser.
                Filename) + "_ORIGINAL.png");
78          var buffer = System.IO.File.ReadAllBytes(System.IO.Path.
                GetFileNameWithoutExtension(filechooser.Filename) + "_ORIGINAL.png");
79          var pixy = new Gdk.Pixbuf(buffer);
80          image1.Pixbuf = pixy;
81          ///--Genera imagen diezmada--///
82          Bitmap decimated_img = new Bitmap((int)Math.Sqrt(pixels.Length), (int)Math.
                Sqrt(pixels.Length));
83          for (y = 0; y < (int)Math.Sqrt(pixels.Length); y++)
84          {
85            for (x = 0; x < (int)Math.Sqrt(pixels.Length); x++)
86            {
87              value = pixels[((int)Math.Sqrt(pixels.Length) * y) + x] == 1 ? (byte)255
                      : (byte)0;
88              Color newColor = System.Drawing.Color.FromArgb(value, value, value);
89              decimated_img.SetPixel(x, y, newColor);
90            }
91          }
92          decimated_img.Save(System.IO.Path.GetFileNameWithoutExtension(filechooser.
                Filename) + "_DIEZMADO.png");
93          var buffer2 = System.IO.File.ReadAllBytes(System.IO.Path.
                GetFileNameWithoutExtension(filechooser.Filename) + "_DIEZMADO.png");
94          var pixy2 = new Gdk.Pixbuf(buffer2);
95          image2.Pixbuf = pixy2;
96          ///--Genera corte frontal del volumen reconstruido--///
97          Bitmap front_view_volume = new Bitmap((int)Math.Sqrt(pixels.Length), (int)
                Math.Sqrt(pixels.Length));
98          for (y = 0; y < (int)Math.Sqrt(pixels.Length); y++)
99          {
100           for (x = 0; x < (int)Math.Sqrt(pixels.Length); x++)
101           {
102             value = reconstruction[((int)Math.Sqrt(pixels.Length) * y) + x] == 1 ? (
                    byte)255 : (byte)0;
103             Color newColor = System.Drawing.Color.FromArgb(value, value, value);
104             front_view_volume.SetPixel(x, y, newColor);
105           }
```

```
106         }
107         front_view_volume.Save(System.IO.Path.GetFileNameWithoutExtension(
                filechooser.Filename) + "_VISTA_FRONTAL_VOLUMEN.png");
108         var buffer3 = System.IO.File.ReadAllBytes(System.IO.Path.
                GetFileNameWithoutExtension(filechooser.Filename) + "
                _VISTA_FRONTAL_VOLUMEN.png");
109         var pixy3 = new Gdk.Pixbuf(buffer3);
110         image3.Pixbuf = pixy3;
111         ///--Genera corte posterior del volumen reconstruido--///
112         Bitmap back_view_volume = new Bitmap((int)Math.Sqrt(pixels.Length), (int)
                Math.Sqrt(pixels.Length));
113         for (y = 0; y < (int)Math.Sqrt(pixels.Length); y++)
114         {
115           for (x = 0; x < (int)Math.Sqrt(pixels.Length); x++)
116           {
117             value = reconstruction[((int)Math.Sqrt(pixels.Length) * (int)Math.Sqrt(
                    pixels.Length) * ((int)Math.Sqrt(pixels.Length) - 1)) + ((int)Math.
                    Sqrt(pixels.Length) * y) + x] == 1 ? (byte)255 : (byte)0;
118             Color newColor = System.Drawing.Color.FromArgb(value, value, value);
119             back_view_volume.SetPixel(x, y, newColor);
120           }
121         }
122         back_view_volume.Save(System.IO.Path.GetFileNameWithoutExtension(filechooser
                .Filename) + "_VISTA_POSTERIOR_VOLUMEN.png");
123         var buffer4 = System.IO.File.ReadAllBytes(System.IO.Path.
                GetFileNameWithoutExtension(filechooser.Filename) + "
                _VISTA_POSTERIOR_VOLUMEN.png");
124         var pixy4 = new Gdk.Pixbuf(buffer4);
125         image4.Pixbuf = pixy4;
126         ///--Salida del sistema--///
127         stopwatch.Stop();
128         label8.Text = (stopwatch.Elapsed).ToString();
129         filechooser.Destroy();
130         MessageDialog md = new MessageDialog(null, DialogFlags.Modal, MessageType.
                Info, ButtonsType.Ok, "Reconstrucción Terminada");
131         md.Run();
132         md.Destroy();
133       }
134     }
135
136     UInt32[,] main_corr_func_2p_lp(int img_width, int img_height, byte[] pixels)
137     {
138       int row, column, relative_length, incremental_f;
139       int real_length = img_width > img_height ? img_width / 2 : img_height / 2;
140       real_length = 1 > real_length ? 1 : real_length;
141       UInt32[,] corr_func_2p_lp = new UInt32[5, real_length];
142       UInt32 first_pixel, pixel_denied;
143       for (row = 0; row < img_height; row++)
144       {
145         for (column = 0; column < img_width; column++)
146         {
147           first_pixel = pixels[(row * img_width) + column];
148           if (first_pixel == 1)
149           {
150             pixel_denied = first_pixel;
151             relative_length = real_length < img_width - column ? real_length :
                    img_width - column;
152             for (incremental_f = 0; incremental_f < relative_length; incremental_f
                    ++)
153             {
154               corr_func_2p_lp[4, incremental_f]++;
155               corr_func_2p_lp[0, incremental_f] += (first_pixel & pixels[(row *
                      img_width) + column + incremental_f]);
156               pixel_denied = pixel_denied & pixels[(row * img_width) + column +
                      incremental_f];
157               corr_func_2p_lp[1, incremental_f] += pixel_denied;
158             }
159             pixel_denied = first_pixel;
160             relative_length = real_length < img_height - row ? real_length :
                    img_height - row;
161             for (incremental_f = 0; incremental_f < relative_length; incremental_f
                    ++)
162             {
163               corr_func_2p_lp[4, incremental_f]++;
```

```
164                 corr_func_2p_lp[0, incremental_f] += (first_pixel & pixels[((row +
                        incremental_f) * img_width) + column]);
165                 pixel_denied = pixel_denied & pixels[((row + incremental_f) *
                        img_width) + column];
166                 corr_func_2p_lp[1, incremental_f] += pixel_denied;
167               }
168             }
169             else
170             {
171               first_pixel = 1;
172               pixel_denied = 1;
173               relative_length = real_length < img_width - column ? real_length :
                      img_width - column;
174               for (incremental_f = 0; incremental_f < relative_length; incremental_f
                      ++)
175               {
176                 corr_func_2p_lp[4, incremental_f]++;
177                 corr_func_2p_lp[2, incremental_f] += (first_pixel ^ pixels[(row *
                        img_width) + column + incremental_f]);
178                 pixel_denied = pixel_denied & (first_pixel ^ pixels[(row * img_width)
                        + column + incremental_f]);
179                 corr_func_2p_lp[3, incremental_f] += pixel_denied;
180               }
181               pixel_denied = 1;
182               relative_length = real_length < img_height - row ? real_length :
                      img_height - row;
183               for (incremental_f = 0; incremental_f < relative_length; incremental_f
                      ++)
184               {
185                 corr_func_2p_lp[4, incremental_f]++;
186                 corr_func_2p_lp[2, incremental_f] += (first_pixel ^ pixels[((row +
                        incremental_f) * img_width) + column]);
187                 pixel_denied = pixel_denied & (first_pixel ^ pixels[((row +
                        incremental_f) * img_width) + column]);
188                 corr_func_2p_lp[3, incremental_f] += pixel_denied;
189               }
190             }
191           }
192         }
193         return corr_func_2p_lp;
194       }
195
196       int main_normalization_corr_func_2p(int img_width, int img_height, UInt32[,]
            corr_func_2p_lp)
197       {
198         int position;
199         int real_length = img_width > img_height ? img_width / 2 : img_height / 2;
200         int first_position = real_length;
201         double[] normalization_corr_func_2p_1 = new double[real_length];
202         double fi_1 = (double)corr_func_2p_lp[0, 0] / (double)corr_func_2p_lp[4, 0];
203         double fi_1_squared = Math.Pow(fi_1, 2);
204         double denominator_fi_1 = fi_1 - fi_1_squared;
205         double[] normalization_corr_func_2p_0 = new double[real_length];
206         double fi_0 = (double)corr_func_2p_lp[2, 0] / (double)corr_func_2p_lp[4, 0];
207         double fi_0_squared = Math.Pow(fi_0, 2);
208         double denominator_fi_0 = fi_0 - fi_0_squared;
209         for (position = 0; position < real_length; position++)
210         {
211           normalization_corr_func_2p_1[position] = (((double)corr_func_2p_lp[0,
                  position] / (double)corr_func_2p_lp[4, position]) - fi_1_squared) /
                  denominator_fi_1;
212           normalization_corr_func_2p_0[position] = (((double)corr_func_2p_lp[2,
                  position] / (double)corr_func_2p_lp[4, position]) - fi_0_squared) /
                  denominator_fi_0;
213           if (normalization_corr_func_2p_1[position] < .02 ||
                  normalization_corr_func_2p_0[position] < .02)
214           {
215             first_position = position;
216             position = real_length;
217           }
218         }
219         return first_position;
220       }
221
```

```
222     byte[] main_decimation(int position, int img_width, int img_height, byte[]
            pixels)
223     {
224       double m_z = Math.Ceiling((3 * (double)img_height) / position);
225       double z = Math.Floor((Math.Log((img_height / m_z), 2)));
226       int reduction_f = (int)Math.Pow(2, z);
227       reduction_f = reduction_f < 1 ? 1 : reduction_f;
228       int reduced_length = img_height / reduction_f;
229       byte[] decimation = new byte[(int)Math.Pow(reduced_length, 2)];
230       int outside_pixel, inside_pixel, acomulator = 0, control_pixel = 0;
231       for (int vertical_increment = 0; vertical_increment < reduced_length;
              vertical_increment++)
232       {
233         outside_pixel = reduction_f * img_width * vertical_increment;
234         for (int horizontal_increment = 0; horizontal_increment < reduced_length;
                horizontal_increment++)
235         {
236           inside_pixel = outside_pixel;
237           for (int vertical_count = 0; vertical_count < reduction_f; vertical_count
                  ++)
238           {
239             for (int horizontal_count = 0; horizontal_count < reduction_f;
                    horizontal_count++)
240             {
241               acomulator = acomulator + pixels[inside_pixel];
242               inside_pixel++;
243             }
244             inside_pixel = (inside_pixel - reduction_f) + img_width;
245           }
246           decimation[control_pixel] = (byte)(acomulator / (Math.Pow(reduction_f, 2))
                  < .5 ? 0 : 1);
247           control_pixel++;
248           acomulator = 0;
249           outside_pixel = outside_pixel + reduction_f;
250         }
251       }
252       return decimation;
253     }
254
255     UInt32[,] main_corr_func_ps(int img_width, int img_height, byte[] pixels)
256     {
257       int row, column, edge_column, edge_row, main_edge, main_radius, control_radius
              , radius_row, radius_column;
258       int minimum_radius = 0;
259       int maximum_radius = 20;
260       UInt32[,] corr_func_ps = new UInt32[3, maximum_radius + 1];
261       UInt32 first_pixel, pixel_denied, evaluator;
262       for (row = minimum_radius; row < img_height - minimum_radius; row++)
263       {
264         for (column = minimum_radius; column < img_width - minimum_radius; column++)
265         {
266           first_pixel = pixels[(row * img_width) + column];
267           if (first_pixel == 1)
268           {
269             pixel_denied = 1;
270             edge_column = column < (img_width - column - 1) ? column : img_width -
                    column - 1;
271             edge_row = row < (img_height - row - 1) ? row : img_height - row - 1;
272             edge_column = edge_column < edge_row ? edge_column : edge_row;
273             main_edge = maximum_radius < edge_column ? maximum_radius : edge_column;
274             for (main_radius = minimum_radius; main_radius <= main_edge; main_radius
                    ++)
275             {
276               control_radius = main_radius * main_radius;
277               for (radius_row = -main_radius; pixel_denied != 0 && radius_row <=
                      main_radius; radius_row++)
278               {
279                 for (radius_column = -main_radius; pixel_denied != 0 &&
                        radius_column <= main_radius; radius_column++)
280                 {
281                   evaluator = Convert.ToUInt32((radius_row * radius_row +
                          radius_column * radius_column) <= control_radius ? pixels[(
                          row + radius_row) * img_width + (column + radius_column)] :
                          1);
```

```
282                   pixel_denied = pixel_denied & evaluator;
283                 }
284               }
285               corr_func_ps[0, main_radius] += pixel_denied;
286               corr_func_ps[2, main_radius]++;
287             }
288           }
289           else
290           {
291             first_pixel = 1;
292             pixel_denied = 1;
293             edge_column = column < (img_width - column - 1) ? column : img_width -
                    column - 1;
294             edge_row = row < (img_height - row - 1) ? row : img_height - row - 1;
295             edge_column = edge_column < edge_row ? edge_column : edge_row;
296             main_edge = maximum_radius < edge_column ? maximum_radius : edge_column;
297             for (main_radius = minimum_radius; main_radius <= main_edge; main_radius
                    ++)
298             {
299               control_radius = main_radius * main_radius;
300               for (radius_row = -main_radius; pixel_denied != 0 && radius_row <=
                      main_radius; radius_row++)
301               {
302                 for (radius_column = -main_radius; pixel_denied != 0 &&
                        radius_column <= main_radius; radius_column++)
303                 {
304                   evaluator = Convert.ToUInt32((radius_row * radius_row +
                          radius_column * radius_column) <= control_radius ? pixels[(
                          row + radius_row) * img_width + (column + radius_column)] :
                          0);
305                   pixel_denied = pixel_denied & (first_pixel ^ evaluator);
306                 }
307               }
308               corr_func_ps[1, main_radius] += pixel_denied;
309               corr_func_ps[2, main_radius]++;
310             }
311           }
312         }
313       }
314       return corr_func_ps;
315     }
316
317     byte[] main_volume_corr_func_ps(int length, UInt32[,] corr_func_ps)
318     {
319       int position_control, control_radius, radius_layer, radius_row, radius_column,
              value_control, random_layer, random_row, random_column,
            subtraction_control, volumetric_overload, overload_control;
320       int maximum_radius = 20;
321       int length_volume_control = (maximum_radius * 2) + 1;
322       int variable_length = length;
323       byte[] volume_control = new byte[(int)Math.Pow(length_volume_control, 3)];
324       byte[] volume = new byte[(int)Math.Pow(length, 3)];
325       UInt32[,] subtraction, volume_corr_func_ps_3d;
326       UInt32[,] control_corr_func_ps_3d = new UInt32[3, maximum_radius + 1];
327       UInt32 volumetric_control = corr_func_ps[0, 0] * (UInt32)length;
328       Random random = new Random();
329       for (position_control = 0; position_control <= maximum_radius;
              position_control++)
330       {
331         control_corr_func_ps_3d[0, position_control] = (int)corr_func_ps[0,
                position_control] * variable_length > 0 ? corr_func_ps[0,
                position_control] * (UInt32)variable_length : 0;
332         control_corr_func_ps_3d[1, position_control] = (int)corr_func_ps[1,
                position_control] * variable_length > 0 ? corr_func_ps[1,
                position_control] * (UInt32)variable_length : 0;
333         control_corr_func_ps_3d[2, position_control] = (int)corr_func_ps[2,
                position_control] * variable_length > 0 ? corr_func_ps[2,
                position_control] * (UInt32)variable_length : 0;
334         variable_length = variable_length - 2 > 0 ? variable_length - 2 : 0;
335       }
336       for (position_control = maximum_radius; position_control >= 0;
              position_control--)
337       {
338         control_radius = position_control * position_control;
```

```
339         for (radius_layer = -position_control; radius_layer <= position_control;
                 radius_layer++)
340         {
341           for (radius_row = -position_control; radius_row <= position_control;
                  radius_row++)
342           {
343             for (radius_column = -position_control; radius_column <=
                    position_control; radius_column++)
344             {
345               volume_control[((maximum_radius + radius_layer) *
                      length_volume_control * length_volume_control) + ((maximum_radius
                       + radius_row) * length_volume_control + (maximum_radius +
                      radius_column))] = (radius_layer * radius_layer + radius_row *
                      radius_row + radius_column * radius_column) <= control_radius ? (
                      byte)1 : (byte)0;
346             }
347           }
348         }
349         if (control_corr_func_ps_3d[0, position_control] > 0)
350         {
351           subtraction = main_corr_func_ps_3d(length_volume_control,
                length_volume_control, length_volume_control, volume_control);
352           for (value_control = 0; value_control < control_corr_func_ps_3d[0,
                position_control]; value_control++)
353           {
354             random_layer = random.Next((0 + position_control), (length -
                  position_control));
355             random_row = random.Next((0 + position_control), (length -
                  position_control));
356             random_column = random.Next((0 + position_control), (length -
                  position_control));
357             control_radius = position_control * position_control;
358             for (radius_layer = -position_control; radius_layer <= position_control;
                   radius_layer++)
359             {
360               for (radius_row = -position_control; radius_row <= position_control;
                     radius_row++)
361               {
362                 for (radius_column = -position_control; radius_column <=
                       position_control; radius_column++)
363                 {
364                   volume[((random_layer + radius_layer) * length * length) + ((
                         random_row + radius_row) * length + (random_column +
                         radius_column))] = (radius_layer * radius_layer + radius_row
                         * radius_row + radius_column * radius_column) <=
                         control_radius ? (byte)1 : volume[((random_layer +
                         radius_layer) * length * length) + ((random_row + radius_row)
                          * length + (random_column + radius_column))];
365                 }
366               }
367             }
368             for (subtraction_control = position_control - 1; subtraction_control >=
                    0; subtraction_control--)
369             {
370               control_corr_func_ps_3d[0, subtraction_control] = (int)
                      control_corr_func_ps_3d[0, subtraction_control] - (int)
                      subtraction[0, subtraction_control] > 0 ? control_corr_func_ps_3d
                      [0, subtraction_control] - subtraction[0, subtraction_control] :
                      0;
371             }
372           }
373         }
374         Array.Clear(volume_control, 0, volume_control.Length);
375       }
376       volume_corr_func_ps_3d = main_corr_func_ps_3d(length, length, length, volume);
377       volumetric_overload = (int)volumetric_control - (int)volume_corr_func_ps_3d[0,
              0];
378       if (volumetric_overload < 0)
379       {
380         for (overload_control = 0; overload_control < volumetric_overload * -1;)
381         {
382           random_layer = random.Next(0, length);
383           random_row = random.Next(0, length);
384           random_column = random.Next(0, length);
```

```
385             overload_control = volume[(random_layer * length * length) + (random_row *
                      length) + random_column] == 1 ? overload_control + 1 :
                    overload_control;
386             volume[(random_layer * length * length) + (random_row * length) +
                    random_column] = volume[(random_layer * length * length) + (
                    random_row * length) + random_column] == 1 ? (byte)0 : volume[(
                    random_layer * length * length) + (random_row * length) +
                    random_column];
387           }
388         }
389         else if (volumetric_overload > 0)
390         {
391           for (overload_control = 0; overload_control < volumetric_overload;)
392           {
393             random_layer = random.Next(0, length);
394             random_row = random.Next(0, length);
395             random_column = random.Next(0, length);
396             overload_control = volume[(random_layer * length * length) + (random_row *
                      length) + random_column] == 0 ? overload_control + 1 :
                    overload_control;
397             volume[(random_layer * length * length) + (random_row * length) +
                    random_column] = volume[(random_layer * length * length) + (
                    random_row * length) + random_column] == 0 ? (byte)1 : volume[(
                    random_layer * length * length) + (random_row * length) +
                    random_column];
398           }
399         }
400         return volume;
401       }
402
403       UInt32[,] main_corr_func_ps_3d(int img_width, int img_height, int img_deep, byte
              [] voxels)
404       {
405         int layer, row, column, edge_column, edge_row, edge_layer, main_edge,
                main_radius, control_radius, radius_layer, radius_row, radius_column;
406         int minimum_radius = 0;
407         int maximum_radius = 20;
408         UInt32[,] corr_func_ps_3d = new UInt32[3, maximum_radius + 1];
409         UInt32 first_voxel, voxel_denied, evaluator;
410         for (layer = minimum_radius; layer < img_deep - minimum_radius; layer++)
411         {
412           for (row = minimum_radius; row < img_height - minimum_radius; row++)
413           {
414             for (column = minimum_radius; column < img_width - minimum_radius; column
                      ++)
415             {
416               first_voxel = voxels[(layer * img_height * img_width) + (row * img_width
                        ) + column];
417               if (first_voxel == 1)
418               {
419                 voxel_denied = 1;
420                 edge_column = column < (img_width - column - 1) ? column : img_width -
                          column - 1;
421                 edge_row = row < (img_height - row - 1) ? row : img_height - row - 1;
422                 edge_layer = layer < (img_deep - layer - 1) ? layer : img_deep - layer
                          - 1;
423                 edge_column = edge_column < edge_row ? edge_column : edge_row;
424                 edge_column = edge_column < edge_layer ? edge_column : edge_layer;
425                 main_edge = maximum_radius < edge_column ? maximum_radius :
                          edge_column;
426                 for (main_radius = minimum_radius; main_radius <= main_edge;
                          main_radius++)
427                 {
428                   control_radius = main_radius * main_radius;
429                   for (radius_layer = -main_radius; voxel_denied != 0 && radius_layer
                            <= main_radius; radius_layer++)
430                   {
431                     for (radius_row = -main_radius; voxel_denied != 0 && radius_row <=
                              main_radius; radius_row++)
432                     {
433                       for (radius_column = -main_radius; voxel_denied != 0 &&
                                radius_column <= main_radius; radius_column++)
434                       {
435                         evaluator = Convert.ToUInt32((radius_layer * radius_layer +
```

```
                                radius_row * radius_row + radius_column * radius_column)
                                <= control_radius ? voxels[((layer + radius_layer) *
                                img_height * img_width) + ((row + radius_row) * img_width
                                 + (column + radius_column))] : 1);
436                           voxel_denied = voxel_denied & evaluator;
437                         }
438                       }
439                     }
440                     corr_func_ps_3d[0, main_radius] += voxel_denied;
441                     corr_func_ps_3d[2, main_radius]++;
442                   }
443                 }
444                 else
445                 {
446                   first_voxel = 1;
447                   voxel_denied = 1;
448                   edge_column = column < (img_width - column - 1) ? column : img_width -
                          column - 1;
449                   edge_row = row < (img_height - row - 1) ? row : img_height - row - 1;
450                   edge_layer = layer < (img_deep - layer - 1) ? layer : img_deep - layer
                          - 1;
451                   edge_column = edge_column < edge_row ? edge_column : edge_row;
452                   edge_column = edge_column < edge_layer ? edge_column : edge_layer;
453                   main_edge = maximum_radius < edge_column ? maximum_radius :
                          edge_column;
454                   for (main_radius = minimum_radius; main_radius <= main_edge;
                          main_radius++)
455                   {
456                     control_radius = main_radius * main_radius;
457                     for (radius_layer = -main_radius; voxel_denied != 0 && radius_layer
                            <= main_radius; radius_layer++)
458                     {
459                       for (radius_row = -main_radius; voxel_denied != 0 && radius_row <=
                              main_radius; radius_row++)
460                       {
461                         for (radius_column = -main_radius; voxel_denied != 0 &&
                                radius_column <= main_radius; radius_column++)
462                         {
463                           evaluator = Convert.ToUInt32((radius_layer * radius_layer +
                                radius_row * radius_row + radius_column * radius_column)
                                <= control_radius ? voxels[((layer + radius_layer) *
                                img_height * img_width) + ((row + radius_row) * img_width
                                 + (column + radius_column))] : 0);
464                           voxel_denied = voxel_denied & (first_voxel ^ evaluator);
465                         }
466                       }
467                     }
468                     corr_func_ps_3d[1, main_radius] += voxel_denied;
469                     corr_func_ps_3d[2, main_radius]++;
470                   }
471                 }
472               }
473             }
474           }
475           return corr_func_ps_3d;
476         }
477
478         byte[] simulated_annealing(int length, byte[] volume_solution, UInt32[,]
                corr_func_2p_lp)
479         {
480           int position_control, control_voxel_value, random_layer_white_b,
                random_row_white_b, random_column_white_b, random_layer_black_w,
                random_row_black_w, random_column_black_w;
481           int corr_func_2p_lp_size = length / 2;
482           corr_func_2p_lp_size = 1 > corr_func_2p_lp_size ? 1 : corr_func_2p_lp_size;
483           int minimum_temperature_factor = (int)Math.Ceiling(Math.Log(length, 2)) * 20;
484           double solution_quality_2p_w, solution_quality_lp_w, solution_quality_2p_b,
                solution_quality_lp_b, error, new_error, delta, probability,
                acceptance_probability;
485           double solution_quality = 0;
486           double temperature = 1E-6;
487           double minimum_temperature = temperature * Math.Pow(.1,
                minimum_temperature_factor);
488           double epsilon = 1E-6;
```

```
489         double alpha = 0.9999;
490         UInt32[,] corr_func_2p_lp_3d = main_corr_func_2p_lp_3d(length, length, length,
                volume_solution);
491         Random random = new Random();
492         for (position_control = 0; position_control < corr_func_2p_lp_size;
                position_control++)
493         {
494           solution_quality_2p_w = ((double)corr_func_2p_lp_3d[0, position_control] / (
                  double)corr_func_2p_lp_3d[4, position_control]) - ((double)
                  corr_func_2p_lp[0, position_control] / (double)corr_func_2p_lp[4,
                  position_control]);
495           solution_quality_2p_w = Math.Pow(solution_quality_2p_w, 2);
496           solution_quality_lp_w = ((double)corr_func_2p_lp_3d[1, position_control] / (
                  double)corr_func_2p_lp_3d[4, position_control]) - ((double)
                  corr_func_2p_lp[1, position_control] / (double)corr_func_2p_lp[4,
                  position_control]);
497           solution_quality_lp_w = Math.Pow(solution_quality_lp_w, 2);
498           solution_quality_2p_b = ((double)corr_func_2p_lp_3d[2, position_control] / (
                  double)corr_func_2p_lp_3d[4, position_control]) - ((double)
                  corr_func_2p_lp[2, position_control] / (double)corr_func_2p_lp[4,
                  position_control]);
499           solution_quality_2p_b = Math.Pow(solution_quality_2p_b, 2);
500           solution_quality_lp_b = ((double)corr_func_2p_lp_3d[3, position_control] / (
                  double)corr_func_2p_lp_3d[4, position_control]) - ((double)
                  corr_func_2p_lp[3, position_control] / (double)corr_func_2p_lp[4,
                  position_control]);
501           solution_quality_lp_b = Math.Pow(solution_quality_lp_b, 2);
502           solution_quality = solution_quality + solution_quality_2p_w +
                  solution_quality_lp_w + solution_quality_2p_b + solution_quality_lp_b;
503         }
504         error = (1.0 / (2.0 * (double)corr_func_2p_lp_size)) * solution_quality;
505         while (temperature > minimum_temperature && epsilon < error)
506         {
507           control_voxel_value = 0;
508           random_layer_white_b = 0;
509           random_row_white_b = 0;
510           random_column_white_b = 0;
511           random_layer_black_w = 0;
512           random_row_black_w = 0;
513           random_column_black_w = 0;
514           while (control_voxel_value == 0)
515           {
516             random_layer_white_b = random.Next(0, length);
517             random_row_white_b = random.Next(0, length);
518             random_column_white_b = random.Next(0, length);
519             control_voxel_value = volume_solution[(random_layer_white_b * length *
                    length) + (random_row_white_b * length) + random_column_white_b] == 1
                     ? control_voxel_value + 1 : control_voxel_value;
520             while (control_voxel_value == 1)
521             {
522               random_layer_black_w = random.Next(0, length);
523               random_row_black_w = random.Next(0, length);
524               random_column_black_w = random.Next(0, length);
525               control_voxel_value = volume_solution[(random_layer_black_w * length *
                      length) + (random_row_black_w * length) + random_column_black_w] ==
                       0 ? control_voxel_value + 1 : control_voxel_value;
526               if (control_voxel_value == 2)
527               {
528                 corr_func_2p_lp_3d = main_corr_func_2p_lp_3d_1p(random_column_white_b,
                        random_row_white_b, random_layer_white_b, length,
                        volume_solution, corr_func_2p_lp_3d);
529                 volume_solution[(random_layer_white_b * length * length) + (
                        random_row_white_b * length) + random_column_white_b] = 0;
530                 corr_func_2p_lp_3d = main_corr_func_2p_lp_3d_1p(random_column_black_w,
                        random_row_black_w, random_layer_black_w, length,
                        volume_solution, corr_func_2p_lp_3d);
531                 volume_solution[(random_layer_black_w * length * length) + (
                        random_row_black_w * length) + random_column_black_w] = 1;
532               }
533             }
534           }
535           solution_quality = 0;
536           for (position_control = 0; position_control < corr_func_2p_lp_size;
                  position_control++)
```

```
537         {
538           solution_quality_2p_w = ((double)corr_func_2p_lp_3d[0, position_control] /
                  (double)corr_func_2p_lp_3d[4, position_control]) - ((double)
                corr_func_2p_lp[0, position_control] / (double)corr_func_2p_lp[4,
                position_control]);
539           solution_quality_2p_w = Math.Pow(solution_quality_2p_w, 2);
540           solution_quality_lp_w = ((double)corr_func_2p_lp_3d[1, position_control] /
                  (double)corr_func_2p_lp_3d[4, position_control]) - ((double)
                corr_func_2p_lp[1, position_control] / (double)corr_func_2p_lp[4,
                position_control]);
541           solution_quality_lp_w = Math.Pow(solution_quality_lp_w, 2);
542           solution_quality_2p_b = ((double)corr_func_2p_lp_3d[2, position_control] /
                  (double)corr_func_2p_lp_3d[4, position_control]) - ((double)
                corr_func_2p_lp[2, position_control] / (double)corr_func_2p_lp[4,
                position_control]);
543           solution_quality_2p_b = Math.Pow(solution_quality_2p_b, 2);
544           solution_quality_lp_b = ((double)corr_func_2p_lp_3d[3, position_control] /
                  (double)corr_func_2p_lp_3d[4, position_control]) - ((double)
                corr_func_2p_lp[3, position_control] / (double)corr_func_2p_lp[4,
                position_control]);
545           solution_quality_lp_b = Math.Pow(solution_quality_lp_b, 2);
546           solution_quality = solution_quality + solution_quality_2p_w +
                solution_quality_lp_w + solution_quality_2p_b + solution_quality_lp_b
                ;
547         }
548         new_error = (1.0 / (2.0 * (double)corr_func_2p_lp_size)) * solution_quality;
549         delta = new_error - error;
550         if (delta < 0)
551         {
552           error = delta + error;
553         }
554         else
555         {
556           probability = random.NextDouble();
557           acceptance_probability = Math.Exp(-delta / temperature);
558           if (probability < acceptance_probability)
559           {
560             error = delta + error;
561           }
562           else
563           {
564             corr_func_2p_lp_3d = main_corr_func_2p_lp_3d_1p(random_column_white_b,
                    random_row_white_b, random_layer_white_b, length, volume_solution,
                    corr_func_2p_lp_3d);
565             volume_solution[(random_layer_white_b * length * length) + (
                    random_row_white_b * length) + random_column_white_b] = 1;
566             corr_func_2p_lp_3d = main_corr_func_2p_lp_3d_1p(random_column_black_w,
                    random_row_black_w, random_layer_black_w, length, volume_solution,
                    corr_func_2p_lp_3d);
567             volume_solution[(random_layer_black_w * length * length) + (
                    random_row_black_w * length) + random_column_black_w] = 0;
568           }
569         }
570         temperature *= alpha;
571       }
572       return volume_solution;
573     }
574
575     UInt32[,] main_corr_func_2p_lp_3d(int img_width, int img_height, int img_deep,
            byte[] voxels)
576     {
577       int layer, row, column, relative_length, incremental_f;
578       int real_length = img_width > img_height ? img_width / 2 : img_height / 2;
579       real_length = 1 > real_length ? 1 : real_length;
580       UInt32[,] corr_func_2p_lp_3d = new UInt32[5, real_length];
581       UInt32 first_voxel, voxel_denied;
582       for (layer = 0; layer < img_deep; layer++)
583       {
584         for (row = 0; row < img_height; row++)
585         {
586           for (column = 0; column < img_width; column++)
587           {
588             first_voxel = voxels[(layer * img_height * img_width) + (row * img_width
                    ) + column];
```

```
589             if (first_voxel == 1)
590             {
591               voxel_denied = first_voxel;
592               relative_length = real_length < img_width - column ? real_length :
                      img_width - column;
593               for (incremental_f = 0; incremental_f < relative_length; incremental_f
                      ++)
594               {
595                 corr_func_2p_lp_3d[4, incremental_f]++;
596                 corr_func_2p_lp_3d[0, incremental_f] += (first_voxel & voxels[(layer
                        * img_height * img_width) + ((row * img_width) + column +
                        incremental_f)]);
597                 voxel_denied = voxel_denied & voxels[(layer * img_height * img_width
                      ) + ((row * img_width) + column + incremental_f)];
598                 corr_func_2p_lp_3d[1, incremental_f] += voxel_denied;
599               }
600               voxel_denied = first_voxel;
601               relative_length = real_length < img_height - row ? real_length :
                      img_height - row;
602               for (incremental_f = 0; incremental_f < relative_length; incremental_f
                      ++)
603               {
604                 corr_func_2p_lp_3d[4, incremental_f]++;
605                 corr_func_2p_lp_3d[0, incremental_f] += (first_voxel & voxels[(layer
                        * img_height * img_width) + (((row + incremental_f) *
                        img_width) + column)]);
606                 voxel_denied = voxel_denied & voxels[(layer * img_height * img_width
                      ) + (((row + incremental_f) * img_width) + column)];
607                 corr_func_2p_lp_3d[1, incremental_f] += voxel_denied;
608               }
609               voxel_denied = first_voxel;
610               relative_length = real_length < img_deep - layer ? real_length :
                      img_deep - layer;
611               for (incremental_f = 0; incremental_f < relative_length; incremental_f
                      ++)
612               {
613                 corr_func_2p_lp_3d[4, incremental_f]++;
614                 corr_func_2p_lp_3d[0, incremental_f] += (first_voxel & voxels[((
                        layer + incremental_f) * img_height * img_width) + ((row *
                        img_width) + column)]);
615                 voxel_denied = voxel_denied & voxels[((layer + incremental_f) *
                        img_height * img_width) + ((row * img_width) + column)];
616                 corr_func_2p_lp_3d[1, incremental_f] += voxel_denied;
617               }
618             }
619             else
620             {
621               first_voxel = 1;
622               voxel_denied = 1;
623               relative_length = real_length < img_width - column ? real_length :
                      img_width - column;
624               for (incremental_f = 0; incremental_f < relative_length; incremental_f
                      ++)
625               {
626                 corr_func_2p_lp_3d[4, incremental_f]++;
627                 corr_func_2p_lp_3d[2, incremental_f] += (first_voxel ^ voxels[(layer
                        * img_height * img_width) + ((row * img_width) + column +
                        incremental_f)]);
628                 voxel_denied = voxel_denied & (first_voxel ^ voxels[(layer *
                        img_height * img_width) + ((row * img_width) + column +
                        incremental_f)]);
629                 corr_func_2p_lp_3d[3, incremental_f] += voxel_denied;
630               }
631               voxel_denied = 1;
632               relative_length = real_length < img_height - row ? real_length :
                      img_height - row;
633               for (incremental_f = 0; incremental_f < relative_length; incremental_f
                      ++)
634               {
635                 corr_func_2p_lp_3d[4, incremental_f]++;
636                 corr_func_2p_lp_3d[2, incremental_f] += (first_voxel ^ voxels[(layer
                        * img_height * img_width) + (((row + incremental_f) *
                        img_width) + column)]);
637                 voxel_denied = voxel_denied & (first_voxel ^ voxels[(layer *
```

```
                        img_height * img_width) + (((row + incremental_f) * img_width)
                        + column)]);
638                   corr_func_2p_lp_3d[3, incremental_f] += voxel_denied;
639                 }
640                 voxel_denied = 1;
641                 relative_length = real_length < img_deep - layer ? real_length :
                        img_deep - layer;
642                 for (incremental_f = 0; incremental_f < relative_length; incremental_f
                        ++)
643                 {
644                   corr_func_2p_lp_3d[4, incremental_f]++;
645                   corr_func_2p_lp_3d[2, incremental_f] += (first_voxel ^ voxels[((
                        layer + incremental_f) * img_height * img_width) + ((row *
                        img_width) + column)]);
646                   voxel_denied = voxel_denied & (first_voxel ^ voxels[((layer +
                        incremental_f) * img_height * img_width) + ((row * img_width) +
                         column)]);
647                   corr_func_2p_lp_3d[3, incremental_f] += voxel_denied;
648                 }
649               }
650             }
651           }
652         }
653         return corr_func_2p_lp_3d;
654       }
655
656       UInt32[,] main_corr_func_2p_lp_3d_1p(int static_column, int static_row, int
              static_layer, int length, byte[] voxels, UInt32[,] corr_func_2p_lp_3d)
657       {
658         int variable_column, relative_length, incremental_f, variable_row,
                variable_layer;
659         int real_length = length / 2;
660         real_length = 1 > real_length ? 1 : real_length;
661         UInt32 first_voxel, voxel_denied;
662         for (variable_column = 0; variable_column < length; variable_column++)
663         {
664           first_voxel = voxels[(static_layer * length * length) + (static_row * length
                ) + variable_column];
665           if (first_voxel == 1)
666           {
667             voxel_denied = first_voxel;
668             relative_length = real_length < length - variable_column ? real_length :
                    length - variable_column;
669             for (incremental_f = 0; incremental_f < relative_length; incremental_f++)
670             {
671               corr_func_2p_lp_3d[4, incremental_f]--;
672               corr_func_2p_lp_3d[0, incremental_f] -= (first_voxel & voxels[(
                    static_layer * length * length) + ((static_row * length) +
                    variable_column + incremental_f)]);
673               voxel_denied = voxel_denied & voxels[(static_layer * length * length) +
                    ((static_row * length) + variable_column + incremental_f)];
674               corr_func_2p_lp_3d[1, incremental_f] -= voxel_denied;
675             }
676           }
677           else
678           {
679             first_voxel = 1;
680             voxel_denied = 1;
681             relative_length = real_length < length - variable_column ? real_length :
                    length - variable_column;
682             for (incremental_f = 0; incremental_f < relative_length; incremental_f++)
683             {
684               corr_func_2p_lp_3d[4, incremental_f]--;
685               corr_func_2p_lp_3d[2, incremental_f] -= (first_voxel ^ voxels[(
                    static_layer * length * length) + ((static_row * length) +
                    variable_column + incremental_f)]);
686               voxel_denied = voxel_denied & (first_voxel ^ voxels[(static_layer *
                    length * length) + ((static_row * length) + variable_column +
                    incremental_f)]);
687               corr_func_2p_lp_3d[3, incremental_f] -= voxel_denied;
688             }
689           }
690         }
691         for (variable_row = 0; variable_row < length; variable_row++)
```

```
692       {
693         first_voxel = voxels[(static_layer * length * length) + (variable_row *
                length) + static_column];
694         if (first_voxel == 1)
695         {
696           voxel_denied = first_voxel;
697           relative_length = real_length < length - variable_row ? real_length :
                length - variable_row;
698           for (incremental_f = 0; incremental_f < relative_length; incremental_f++)
699           {
700             corr_func_2p_lp_3d[4, incremental_f]--;
701             corr_func_2p_lp_3d[0, incremental_f] -= (first_voxel & voxels[(
                    static_layer * length * length) + (((variable_row + incremental_f)
                    * length) + static_column)]);
702             voxel_denied = voxel_denied & voxels[(static_layer * length * length) +
                    (((variable_row + incremental_f) * length) + static_column)];
703             corr_func_2p_lp_3d[1, incremental_f] -= voxel_denied;
704           }
705         }
706         else
707         {
708           first_voxel = 1;
709           voxel_denied = 1;
710           relative_length = real_length < length - variable_row ? real_length :
                length - variable_row;
711           for (incremental_f = 0; incremental_f < relative_length; incremental_f++)
712           {
713             corr_func_2p_lp_3d[4, incremental_f]--;
714             corr_func_2p_lp_3d[2, incremental_f] -= (first_voxel ^ voxels[(
                    static_layer * length * length) + (((variable_row + incremental_f)
                    * length) + static_column)]);
715             voxel_denied = voxel_denied & (first_voxel ^ voxels[(static_layer *
                    length * length) + (((variable_row + incremental_f) * length) +
                    static_column)]);
716             corr_func_2p_lp_3d[3, incremental_f] -= voxel_denied;
717           }
718         }
719       }
720       for (variable_layer = 0; variable_layer < length; variable_layer++)
721       {
722         first_voxel = voxels[(variable_layer * length * length) + (static_row *
                length) + static_column];
723         if (first_voxel == 1)
724         {
725           voxel_denied = first_voxel;
726           relative_length = real_length < length - variable_layer ? real_length :
                length - variable_layer;
727           for (incremental_f = 0; incremental_f < relative_length; incremental_f++)
728           {
729             corr_func_2p_lp_3d[4, incremental_f]--;
730             corr_func_2p_lp_3d[0, incremental_f] -= (first_voxel & voxels[((
                    variable_layer + incremental_f) * length * length) + ((static_row *
                     length) + static_column)]);
731             voxel_denied = voxel_denied & voxels[((variable_layer + incremental_f) *
                     length * length) + ((static_row * length) + static_column)];
732             corr_func_2p_lp_3d[1, incremental_f] -= voxel_denied;
733           }
734         }
735         else
736         {
737           first_voxel = 1;
738           voxel_denied = 1;
739           relative_length = real_length < length - variable_layer ? real_length :
                length - variable_layer;
740           for (incremental_f = 0; incremental_f < relative_length; incremental_f++)
741           {
742             corr_func_2p_lp_3d[4, incremental_f]--;
743             corr_func_2p_lp_3d[2, incremental_f] -= (first_voxel ^ voxels[((
                    variable_layer + incremental_f) * length * length) + ((static_row *
                     length) + static_column)]);
744             voxel_denied = voxel_denied & (first_voxel ^ voxels[((variable_layer +
                    incremental_f) * length * length) + ((static_row * length) +
                    static_column)]);
745             corr_func_2p_lp_3d[3, incremental_f] -= voxel_denied;
```

```
746             }
747           }
748         }
749         voxels[(static_layer * length * length) + (static_row * length) +
                static_column] = voxels[(static_layer * length * length) + (static_row *
                length) + static_column] == 1 ? (byte)0 : (byte)1;
750         for (variable_column = 0; variable_column < length; variable_column++)
751         {
752           first_voxel = voxels[(static_layer * length * length) + (static_row * length
                ) + variable_column];
753           if (first_voxel == 1)
754           {
755             voxel_denied = first_voxel;
756             relative_length = real_length < length - variable_column ? real_length :
                    length - variable_column;
757             for (incremental_f = 0; incremental_f < relative_length; incremental_f++)
758             {
759               corr_func_2p_lp_3d[4, incremental_f]++;
760               corr_func_2p_lp_3d[0, incremental_f] += (first_voxel & voxels[(
                      static_layer * length * length) + ((static_row * length) +
                      variable_column + incremental_f)]);
761               voxel_denied = voxel_denied & voxels[(static_layer * length * length) +
                      ((static_row * length) + variable_column + incremental_f)];
762               corr_func_2p_lp_3d[1, incremental_f] += voxel_denied;
763             }
764           }
765           else
766           {
767             first_voxel = 1;
768             voxel_denied = 1;
769             relative_length = real_length < length - variable_column ? real_length :
                    length - variable_column;
770             for (incremental_f = 0; incremental_f < relative_length; incremental_f++)
771             {
772               corr_func_2p_lp_3d[4, incremental_f]++;
773               corr_func_2p_lp_3d[2, incremental_f] += (first_voxel ^ voxels[(
                      static_layer * length * length) + ((static_row * length) +
                      variable_column + incremental_f)]);
774               voxel_denied = voxel_denied & (first_voxel ^ voxels[(static_layer *
                      length * length) + ((static_row * length) + variable_column +
                      incremental_f)]);
775               corr_func_2p_lp_3d[3, incremental_f] += voxel_denied;
776             }
777           }
778         }
779         for (variable_row = 0; variable_row < length; variable_row++)
780         {
781           first_voxel = voxels[(static_layer * length * length) + (variable_row *
                length) + static_column];
782           if (first_voxel == 1)
783           {
784             voxel_denied = first_voxel;
785             relative_length = real_length < length - variable_row ? real_length :
                    length - variable_row;
786             for (incremental_f = 0; incremental_f < relative_length; incremental_f++)
787             {
788               corr_func_2p_lp_3d[4, incremental_f]++;
789               corr_func_2p_lp_3d[0, incremental_f] += (first_voxel & voxels[(
                      static_layer * length * length) + (((variable_row + incremental_f)
                      * length) + static_column)]);
790               voxel_denied = voxel_denied & voxels[(static_layer * length * length) +
                      (((variable_row + incremental_f) * length) + static_column)];
791               corr_func_2p_lp_3d[1, incremental_f] += voxel_denied;
792             }
793           }
794           else
795           {
796             first_voxel = 1;
797             voxel_denied = 1;
798             relative_length = real_length < length - variable_row ? real_length :
                    length - variable_row;
799             for (incremental_f = 0; incremental_f < relative_length; incremental_f++)
800             {
801               corr_func_2p_lp_3d[4, incremental_f]++;
```

```
802             corr_func_2p_lp_3d[2, incremental_f] += (first_voxel ^ voxels[(
                    static_layer * length * length) + (((variable_row + incremental_f)
                    * length) + static_column)]);
803             voxel_denied = voxel_denied & (first_voxel ^ voxels[(static_layer *
                    length * length) + (((variable_row + incremental_f) * length) +
                    static_column)]);
804             corr_func_2p_lp_3d[3, incremental_f] += voxel_denied;
805           }
806         }
807
808       }
809       for (variable_layer = 0; variable_layer < length; variable_layer++)
810       {
811         first_voxel = voxels[(variable_layer * length * length) + (static_row *
                length) + static_column];
812         if (first_voxel == 1)
813         {
814           voxel_denied = first_voxel;
815           relative_length = real_length < length - variable_layer ? real_length :
                length - variable_layer;
816           for (incremental_f = 0; incremental_f < relative_length; incremental_f++)
817           {
818             corr_func_2p_lp_3d[4, incremental_f]++;
819             corr_func_2p_lp_3d[0, incremental_f] += (first_voxel & voxels[((
                    variable_layer + incremental_f) * length * length) + ((static_row *
                     length) + static_column)]);
820             voxel_denied = voxel_denied & voxels[((variable_layer + incremental_f) *
                     length * length) + ((static_row * length) + static_column)];
821             corr_func_2p_lp_3d[1, incremental_f] += voxel_denied;
822           }
823         }
824         else
825         {
826           first_voxel = 1;
827           voxel_denied = 1;
828           relative_length = real_length < length - variable_layer ? real_length :
                length - variable_layer;
829           for (incremental_f = 0; incremental_f < relative_length; incremental_f++)
830           {
831             corr_func_2p_lp_3d[4, incremental_f]++;
832             corr_func_2p_lp_3d[2, incremental_f] += (first_voxel ^ voxels[((
                    variable_layer + incremental_f) * length * length) + ((static_row *
                     length) + static_column)]);
833             voxel_denied = voxel_denied & (first_voxel ^ voxels[((variable_layer +
                    incremental_f) * length * length) + ((static_row * length) +
                    static_column)]);
834             corr_func_2p_lp_3d[3, incremental_f] += voxel_denied;
835           }
836         }
837       }
838       return corr_func_2p_lp_3d;
839     }
840   }
```