Universidad Autónoma Metropolitana Unidad Azcapotzalco

División de Ciencias Básicas e Ingeniería

Licenciatura en Ingeniería en Computación

Proyecto tecnológico

Generación y descomposición de cubo en piezas para rompecabezas

Alan Anaya Boyer

206204719

Dr. Gueorgi Khatchatourov

Profesor titular del departamento de sistemas

Trimestre 2014 Primavera

29 de Agosto del año 2014

Declaratoria.

Yo, Dr. Gueorgi Khatchatourov, declaro que aprobé el contenido del presente Reporte de Proyecto de Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Firma: _____

Dr. Gueorgi Khatchatourov.
Profesor titular del departamento de sistemas.

Yo, Alan Anaya Boyer, doy mi autorización a la Coordinación de Servicios de Información de la Universidad Autónoma Metropolitana, Unidad Azcapotzalco, para publicar el presente documento en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Firma: _____

Alan Anaya Boyer. 206204719

Tabla de contenido

Índice de figuras

Índice de tablas

Generación y descomposición de cubo en piezas para rompecabezas

1. Resumen

En este proyecto se puede construir inicialmente un cubo de longitud N , por medio de unas unidades atómicas conocidas como POLICUBOS, Posteriormente el cubo construido es procesado mediante un algoritmo el cual descompone el cubo inicial en seis piezas para rompecabezas las cuales están asignadas a cada una de las caras del cubo. Mediante una máscara de bits del tamaño de la longitud al cuadrado la cual genera todas las posibles combinaciones en 2 dimensiones para cada cara, en donde cada bit representa un POLICUBO de la cara respectiva, se obtiene un resultado, el cual se prueba su continuidad mediante un algoritmo basado en FLOOD FILL para 2 dimensiones. Al obtener un candidato se pasa a la construcción de las torres con una altura inicial de valor uno con los valores de la máscara obtenidos en el recorrido inicial, posteriormente se hace una prueba de conectividad en 3 dimensiones mediante el algoritmo utilizado en FLOOD FILL para 2 dimensiones modificado para 3 dimensiones. Al obtener un candidato con la prueba de conectividad en 3 dimensiones pasa a la construcción de la segunda pieza y así sucesivamente hasta llegar a la sexta pieza, al tener 6 piezas candidatas, si se alcanza el volumen total de POLICUBOS igual al del cubo inicial se encuentra una combinación candidata para el rompecabezas guardando el resultado en un archivo, continuando el recorrido hasta que se cubra el valor 2^(N^2) que es el número máximo de combinaciones de las caras y posteriormente incrementando la altura inicial en 1 hasta la final de tamaño de la Longitud N. Debido a la complejidad computacional que se calcula posteriormente se eligieron los valores de N = 3 y N = 5 como las longitudes a tratar en el algoritmo aunque esta implementado para calcular una longitud máxima de 13. Las combinaciones de rompecabezas generadas por el algoritmo están acompañadas con su visualización en 3D mediante una técnica de visualización de gráficos llamada OpenGL. Por ejemplo para N = 3, el algoritmo genera 1435 combinaciones posibles para rompecabezas.

2. Introducción

2.1.Antecedentes

En el mercado se conocen rompecabezas donde un objeto se descompone en partes. Consideramos que es importante comprender al fondo las técnicas computacionales de la generación de tales rompecabezas.

El cubo Soma [5] (figuras 1, 2) es un rompecabezas geométrico, con siete piezas formadas con cubos que hay que unir para conseguir un cubo mayor. Fue creado por Piet Hein en el año 1936. Se dice que durante una conferencia de Heisenberg, Hein empezó a pensar en los distintos POLICUBOS que se podían obtener uniendo varios cubos del mismo tamaño, y comprobó que todos los POLICUBOS irregulares formados por cuatro o menos cubos sumaban un total de 27 cubos, y podían unirse en un cubo mayor con tres cubos de arista.

Posteriormente, el matemático John Conway comprobó que había 240 formas distintas de resolver el problema principal. Con las piezas del cubo Soma se pueden crear otras formas, con diseños geométricos más o menos interesantes o incluso diseños figurativos. Hay recopilaciones con miles de estas figuras. Las siete figuras del Soma se pueden identificar con un número o con una letra. Los POLICUBOS de 4 o menos cubos que no figuran en esta lista son todos regulares.

Figura 1.Cubo Soma.

Figura 2.Descomposición Cubo Soma.

El cubo de Bedlam [6] (figuras 3, 4) es un puzzle mecánico inventado por el experto británico en puzzles Bruce Bedlam. El puzzle consiste en 13 piezas policúbicas: 12 pentacubos y un tetracubo. El objetivo es ensamblar las piezas en un cubo de 4 x 4 x 4. Hay 19186 formas distintas de hacerlo, excluyendo rotaciones y reflexiones. Aunque el cubo de Bedlam es esencialmente justo el siguiente paso lógico al cubo Soma de 3 x 3 x 3, es mucho más difícil de resolver.

Figura 3.Cubo Bedlam.

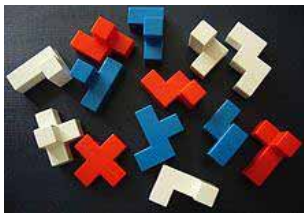

Figura 4.Descomposición Cubo Bedlam.

El rompecabezas Slothouber-Graatsma[7] es un problema de empaquetamiento que utiliza para el empaquetamiento seis bloques de $1 \times 2 \times 2$ (Mostrados en la figura 5) y tres bloques de $1 \times 1 \times 1$ en una caja de $3 \times 3 \times 3$. La solución a este rompecabezas es única (hasta mostrar reflexiones y rotaciones). El rompecabezas es esencialmente el mismo si los tres bloques de $1 \times 1 \times 1$ se quedan fuera, por lo que la tarea es hacer que los seis bloques de $1 \times 2 \times 2$ quepan en una caja cúbica con volumen 27. El rompecabezas Slothouber-Graatsma es considerado como el más pequeño no trivial 3D problema de empaquetamiento.

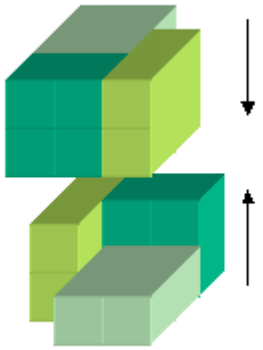

Figura 5.Seis bloques de los nueve totales en el rrompecabezas Slothouber-Graatsma.

2.2.Formulación del problema

Un POLICUBO consiste en un conjunto de módulos cúbicos unitarios unidos por sus lados. La teoría de POLICUBOS [1], es una rama de las matemáticas que se ocupa de estudiar el comportamiento de unidades modulares cúbicas, tal que unidas por sus caras configuran formas en el espacio tridimensional.

Para realizar un avance relacionado con esta idea podemos plantear el problema de dividir un cubo en dos o más partes.

La construcción de nuestras piezas estuvo realizada mediante los siguientes puntos:

1. La construcción de nuestras piezas la realizamos a partir del cubo inical intentando desarmarlo paso a paso.
2. Realizamos la descomposición para todas las caras de manera sucesiva.
3. En cada cara se obtiene una sola pieza, la cual es considerada como un objeto formado por policubos de un volumen conectado.
4. La operación utilizada para sacar las piezas se realiza mediante traslación a lo largo de la dirección ortogonal a la cara correspondiente.
5. En el proceso de generación de volúmenes, a partir de una superficie marcada en cada cara elegida, las torres construidas sobre cada cuadrado de la superficie para generar una pieza tienen la misma longitud.

Cabe mencionar que estas cuatro condiciones delimitan de alguna manera construcción de rompecabezas, por ejemplo, comparando con antecedentes. Por otro lado podemos proponer otras restricciones naturales no incluidos en este conjunto, por ejemplo, podemos especificar que en cada etapa de la descomposición el resto del rompecabezas no sea des integrable al

sacudirlo mecánicamente. Estos factores deben ser considerados para trabajos futuros extendiendo el proyecto actual.

En ese sentido el requerimiento de tener una sola pieza para cada cara (punto 3) y los requerimientos de los puntos 4 y 5, en futuras extensiones de este trabajo pueden ser eliminados o reducidos.

Tomando en cuenta que el universo de las combinaciones de descomposiciones del cubo a través de disecciones arbitrarias es infinito, se propone delimitar el problema para que el conjunto de las descomposiciones posibles sea finito. Para ello en nuestro caso supongamos que el propio cubo sea formado por cubos unitarios y dichas partes se construyen como conjuntos formados por estos detalles atómicos. Entonces, la clase de volúmenes para representar las partes posibles se delimita por los objetos conocidos como POLICUBOS.

Al aumentar el número de cubos unitarios en la dimensión del cubo, obviamente aumenta la complejidad de resolver el problema.

En este proyecto se tiene como finalidad desarrollar un método que utilice un enfoque modular y que ocupe combinatoria sobre estos módulos.

Luego, al encontrar soluciones posibles, queremos aprovechar la técnica de visualización para mostrar tanto las partes de rompecabezas encontradas como el propio proceso de descomposición en dinámica.

3. Justificación

Utilizar un método que utilice módulos formados por cubos unitarios para descomposición del cubo permite reducir el problema original a un problema combinatorio.

Este proyecto propone una metodología que sea capaz de encontrar todas las descomposiciones bajo restricciones mencionadas en los puntos tratados en la sección anterior. A pesar de aquellas limitaciones la complejidad numérica de nuestra solución crece demasiado rápido junto con la dimensión del cubo. Se logró investigar la dependencia de la complejidad tanto analíticamente como mediante unas pruebas computacionales con cubos de diferentes dimensiones (3, 5).

El diseño e implementación de metodologías y algoritmos, tales como la búsqueda de descomposiciones y la visualización, así como investigación de la complejidad numérica y la integración de componentes en un producto final, ha sido el resultado final de la realización de este proyecto. Además los resultados obtenidos en este proyecto pueden ser utilizados en práctica para la producción de rompecabezas reales.

4. Objetivos

4.1.Objetivo General:

Diseñar e implementar un algoritmo que permita realizar la generación y descomposición de un cubo en partes, los cuales representen las piezas de un rompecabezas.

4.2.Objetivos específicos:

- Diseñar un algoritmo que permita generar las piezas del rompecabezas en base a la teoría de POLICUBOS.
- Diseñar las estructuras de datos para guardar las piezas resultantes del rompecabezas.
- Diseñar algoritmo de búsqueda que permita descomponer el cubo en piezas con todas sus partes linealmente conectadas.
- Visualizar el proceso de descomposición de cada combinación encontrada a través de un programa implementado mediante OpenGL.

5. Marco teórico

5.1.Algoritmo

En un sentido amplio, dado un problema y un dispositivo donde resolverlo, es necesario proporcionar un método preciso que lo resuelva, adecuado al dispositivo. A tal método lo denominamos algoritmo [8]. Existen dos aspectos muy importantes de los algoritmos, como son su diseño y el estudio de su eficiencia. El primero se refiere a la búsqueda de métodos o procedimientos, secuencias finitas de instrucciones adecuadas al dispositivo que disponemos, que permitan resolver el problema. Por otra parte, el segundo nos permite pedir de alguna forma el coste (en tiempo y recursos) que consume un algoritmo para encontrar la solución y nos ofrece la posibilidad de comparar distintos algoritmos que resuelven un mismo problema.

5.2. Eficiencia y complejidad

Una vez dispongamos de un algoritmo que funciona correctamente, es necesario definir criterios para medir su rendimiento o comportamiento [8]. Estos criterios se centran principalmente en su simplicidad y en el uso eficiente de los recursos. Respecto al uso eficiente de los recursos, éste suele medirse en función de dos parámetros: el espacio, es decir, memoria que utiliza, y el tiempo, lo que tarda en ejecutarse. Ambos representan los costes que supone encontrar la solución al problema planteado mediante un algoritmo.

Dichos parámetros van a servir además para comparar algoritmos entre sí, permitiendo determinar el más adecuado de entre varios que solucionan un mismo problema.
El tiempo de ejecución de un algoritmo va a depender de diversos factores como son: los datos de entrada que le suministremos, la calidad del código generado por el compilador para crear el programa objeto, la naturaleza y rapidez de las instrucciones máquina del procesador concreto que ejecute el programa, y la complejidad intrínseca del algoritmo.
La unidad de tiempo a la que deben hacer referencia estas medidas de eficiencia no puede ser expresada en segundos o en otra unidad de tiempo concreta, pues no existe una computadora estándar al que puedan hacer referencia todas las medidas. Teóricamente $T(n)$ debe indicar el número de instrucciones ejecutadas por un ordenador idealizado. En nuestro caso se calcula la complejidad computacional del algoritmo de construcción de las piezas en el desarrollo del proyecto.

5.2.1. Principio de Invarianza

Dado un algoritmo y dos implementaciones suyas I1 e I2, que tardan T1(n) y T2(n) segundos respectivamente, el Principio de Invarianza afirma que existe una constante real $c > 0$ y un número natural n0 tales que para todo $n \geq n0$ se verifica que $T1(n) \leq cT2(n)$. Es decir, el tiempo de ejecución de dos implementaciones distintas de un algoritmo dado no va a diferir más que en una constante multiplicativa.

5.2.2. Regla de la suma.

Si T1(n) y T2(n) son las funciones que expresan los tiempos de ejecución de dos fragmentos de un programa, y se acotan de forma que se tiene:
T1(n) = O (f1(n)) y T2(n) = O (f2(n))
Se puede decir que:
T1(n) + T2(n) = O (Max (f1(n), f2(n))).

5.2.3. Regla del producto

Si T1(n) y T2(n) son las funciones que expresan los tiempos de ejecución de dos fragmentos de un programa, y se acotan de forma que se tiene:
T1(n) = O (f1(n)) y T2(n)=O (f2(n))
Se puede decir que:
T1(n) T2(n) = O (f1(n) f2(n)).

5.3. Algoritmo de relleno por difusión(FLOOD FILL)

El algoritmo de relleno por difusión [3], también llamado algoritmo de relleno, o -directamente del inglés- FLOODFILL determina el área formada por elementos contiguos en una matriz multidimensional. Se usa en la herramienta Bote de pintura de programas de

dibujo para determinar qué partes de un mapa de bits se van a rellenar de un color (o una textura), y en juegos como el Buscaminas, Puyo, Lumines y Magical Drop para determinar qué piezas pueden retirarse o seleccionarse.
El algoritmo de relleno en programas de dibujo, creado por S. Fazzini, requiere tres parámetros: un nodo inicial, un color para sustituir, y otro de relleno. El algoritmo rastrea todos los nodos que sean del color seleccionado, y a la vez contiguos entre sí y con el inicial, y los sustituye por el color de relleno. En nuestro caso se aplica a la prueba de conectividad de volúmenes generados. Es decir, si a partir de una semilla el relleno coincide con todo volumen contenedor de la semilla, entonces la prueba es positiva, en el caso contrario no. Esto se aplica en los algoritmos de verificación en dos dimensiones, el algoritmo de verificación en tres dimensiones y para el algoritmo de construcción de las torres sustituyendo, el color a sustituir y el color de relleno del algoritmo original por unas banderas de tipo booleano una con valor de false y otra con valor de true, que les indica a los algoritmos antes mencionados, los policubos que se utilizaran en el proceso de selección de candidatos.

5.4.Combinaciones de bits

Con un bit podemos representar solamente dos valores, que suelen representarse como 0, 1. Para representar o codificar más información en un dispositivo digital, necesitamos una mayor cantidad de bits [9]. Si usamos dos bits, tendremos cuatro combinaciones posibles:
0 0 - Los dos están "apagados"
0 1 - El primero está "apagado" y el segundo "encendido"
1 0 - El primero está "encendido" y el segundo "apagado"
1 1 - Los dos están "encendidos"
Con estas cuatro combinaciones podemos representar hasta cuatro valores diferentes, como por ejemplo, los colores azul, verde, rojo y magenta.
A través de secuencias de bits, se puede codificar cualquier valor discreto como números, palabras, e imágenes. Cuatro bits forman un nibble, y pueden representar hasta 2^4 = 16 valores diferentes; ocho bits forman un octeto, y se pueden representar hasta 2^8 = 256 valores diferentes. En general, con un número n de bits pueden representarse hasta 2n valores diferentes. En nuestro caso al crear la máscara de bits del tamaño del cuadrado de la longitud ,por ejemplo con longitud 3, tendremos una máscara de 9 bits la cual va a representar los 9 policubos de una cara del cubo inicial para cada una de sus seis cara, o sea sus 6 piezas, al ser un conjunto de nueve bits tenemos (2^9) – 1 posibles combinaciones de bits que es igual a 511 combinaciones debido a que se omite la combinación en donde todos los bits son cero, esto significa que tenemos 511 combinaciones por cada pieza del rompecabezas para un cubo de 3x3x3 que se someten a las pruebas de conectividad para la construcción de las torres en cada respectiva pieza del rompecabezas.

6. Desarrollo del proyecto

El funcionamiento principal de nuestro algoritmo consta en su parte inicial el introducir un valor N, el cual tiene que ser impar entre 3 y 13 ya que el programa está diseñado con memoria estática para procesar máximo una longitud de 13 y también, a que la representación visual del cubo para números pares no es tan clara como para números impares debido a la combinación de POLICUBOS negros y de color, pero se puede realizar sin ningún problema para procesar valores pares, con este valor dado almacenado en la variable Longitud se debe generar un cubo con un volumen formado por POLICUBOS(figuras 6, 7, 8) el cual posteriormente va a ser descompuesto en 6 piezas de rompecabezas, una pieza por cada cara del cubo.

Figura 8.Cubo 3x3.

Figura 6.Cubo 5x5.

Figura 7.Cubo 7x7.

6.1.Diagrama de Clases

Figura 9.Diagrama de Clases.

6.2.Clase Principal

En la clase principal se tiene una variable del tipo time_t para llevar un control de la fecha real de ejecución del programa y esta fecha es almacenada en una variable de tipo char para tener el registro de los resultados en un archivo se creado mediante ofstream.

Mediante las variables GLsizei wSize = 700, hSize = 700 se obtiene el tamaño de la ventana para la graficación, con las variables GLboolean bCull = glIsEnabled(GL_CULL_FACE), bDepth = glIsEnabled(GL_DEPTH_TEST), bOutline = (GLboolean)true, bLight = glIsEnabled(GL_LIGHTING), GLenum shademode = GL_FLAT se generan los efectos visuales de los cubos, Mediante las variables GLdouble ex = EX, ey = EY, ez = EZ, cx = CX, cy = CY, cz = CZ,ux = UX, uy = UY,uz = UZ, se asignan valores predeterminados para la posición de la cámara, mediante las variables delta = DELTA, deltaR = DELTA_R, se modifica la posición de la cámara hacer acercamientos, alejamientos y cambios de posición de la misma, con la variable GLfloat Color[3] = { 1.0, 1.0, 1.0 } se asigna un color inicial del volumen total y mediante la variable GLfloat Desp = 0, se controla el desplazamiento de las piezas al ser graficadas.

Al modificar los variables ex = Longitud * 2.5, ey = Longitud * 2.5, ez = Longitud * 2.5, cx = Longitud / 2, cy = Longitud / 2, cz = Longitud / 2 se posiciona la cámara dependiendo del valor de la longitud del cubo.

Con la función TeclasEspeciales(cx, cy, cz, ex, ey, ez, ux, uy, uz, deltaR, delta, bCull, bDepth, bOutline, shademode, bLight,Color,Desp) y sus parámetros se hace la asignación de valores para su utilización para las modificaciones por teclas especiales y de teclado que muestran los efectos del gráfico.

Este cubo es creado mediante una instancia de la clase Cubo llamada Volumen que recibe como parámetro en su constructor la longitud y un apuntador aun arreglo de colores para poder modificar los colores del cubo inicial. Posteriormente con la función miembro de Volumen llamada CrearCubo() se crea un arreglo tridimensional llamado cubo que es una instancia de la clase Policubo que en sus miembros tiene como variables las coordenadas (x, y, z,) declaradas como enteros que indican de la posición del policubo en ese arreglo tridimensional, una variable ocupado de tipo booleano que sirve para indicar si ese policubo se utiliza o en el proceso de construcción y graficación y una variable de tipo int llamada color que indica el color del policubo en una posición especifica.

Posteriormente se invoca al miembro de la clase Volumen ResultCubo() para imprimir los resultados por la salida estándar y en un archivo de texto. Inmediatamente después el objeto Volumen es enviado mediante la función RenderizarPolicubo() a la función Display() para su graficación en pantalla.

Se crea una instancia de la clase Rompecabezas llamada rompecabezas que recibe como parámetros en su constructor la dirección de la clase Volumen, la longitud y la fecha, que posteriormente llama al miembro CrearRompecabezas() que es la clase que se encarga de la creación del rompecabezas,

Posteriormente se llama a la variable tipo de la clase rompecabezas la cual si su valor es n utiliza las funciones RenderizarDesCubo() enviándole como parámetro la clase rompecabezas y la función RenderizarPolicubo() con parámetro la clase Volumen para la graficación de las piezas de rompecabezas mediante la función display.

Posteriormente se utiliza al conjunto de funciones de OpenGL para lograr la graficación del objeto deseado, glutInit(&argc, argv) para inicializar el grafico, glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB) para el modo de despliegue de imagen, glutInitWindowSize(wSize, hSize) para inicializar el tamaño de la ventana, glutInitWindowPosition(1, 1) para inicializar la posición de la ventana, glutCreateWindow(argv[0]), para crear la ventana, SetupRC() para elegir el fondo y configuraciones de luz, glutDisplayFunc(Display) para desplegar el gráfico, glutReshapeFunc(ChangeSize) para manejar la perspectiva del gráfico, glutSpecialFunc(SpecialKeys) para el uso de las teclas especiales como F1 por ejemplo, glutKeyboardFunc(keyboard) para usar el teclado como la tecla W y glutMainLoop() para mantener el gráfico en un ciclo(el código fuente se incluye en el apéndice).

6.3.Clase Policubo

En esta clase se tienen las variables del tipo int posicionx, posiciony, posicionz que representan las coordenadas de los policubos en un arreglo tridimensional, se tiene la variable de tipo entero llamada color que guarda un valor que representa la coloración del policubo y en la variable de tipo booleano llamada ocupado que indica si se utiliza o no el policubo tanto en la graficación o en la construcción de las piezas mediante los algoritmos utilizados. La clase en su constructor Policubo(int x, int y , int z, bool Ocupado) inicial recibe los valores de las coordenadas y el estatus. En el miembro SetPosicion(int &x,int &y,int &z) se asignan las coordenadas, con el miembro GetPosicion(int &x, int &y, int &z) se obtienen las coordenadas, SetStatus(bool &Ocupado) con este miembro se asigna el estatus, con este miembro GetStatus() se obtiene el estatus, con este miembro o función SetColor(int &Color) de asigna el color, con la función GetColor() se obtiene el color, la función GrafPolicubo() contiene los valores para dibujar en la pantalla un policubo de dimensión uno, con la función Cuadrado() se construye un cuadrado que es utilizado por la función GrafPolicubo() y finalmente la función Reset() , manda todos los valores a cero(el código fuente se incluye en el apéndice).

6.4.Clase Cubo

En esta clase se tiene como variables, un apuntador llamado *color del tipo GLfloat que apunta a los colores que puede tener el cubo inicial, un arreglo tridimensional llamado cubo[maxsize][maxsize][maxsize] del tipo Policubo con maxsize de tamaño 13 que sirve para crear el cubo principal formado por policubos y la variable Longitud que determina el tamaño del cubo.

Mediante le constructor de la clase de recibe la longitud y los colores que puede tener el cubo mediante un apuntador Cubo(int longitud, GLfloat *Color), con la función CrearCubo() se construye el cubo principal con un triple ciclo for con variables i , j , k que se inicializan en cero hasta longitud -1, y un estado con valor true que son asignando a los policubos del arreglo mediante su constructor y una variable pivote con valor inicial de uno que les asigna un color a cada elemento del arreglo mediante SetColor incrementándose la variable pivote para obtener valores diferentes en cada policubo, con la función ResultCubo() se imprime en pantalla en un archivo mediante un ciclo for triple los resultados de la creación del cubo, mediante la función GrafCubo() de nuevo mediante un ciclo for triple y con el color asignado al policubo mediante la variable pivote se llama a la función GrafPolicubo() para graficar el cubo creado por policubos mostrado en la figura 6,7,8(el código fuente se incluye en el apéndice).

6.5.Diagrama de flujo del Algoritmo para la creación de las piezas

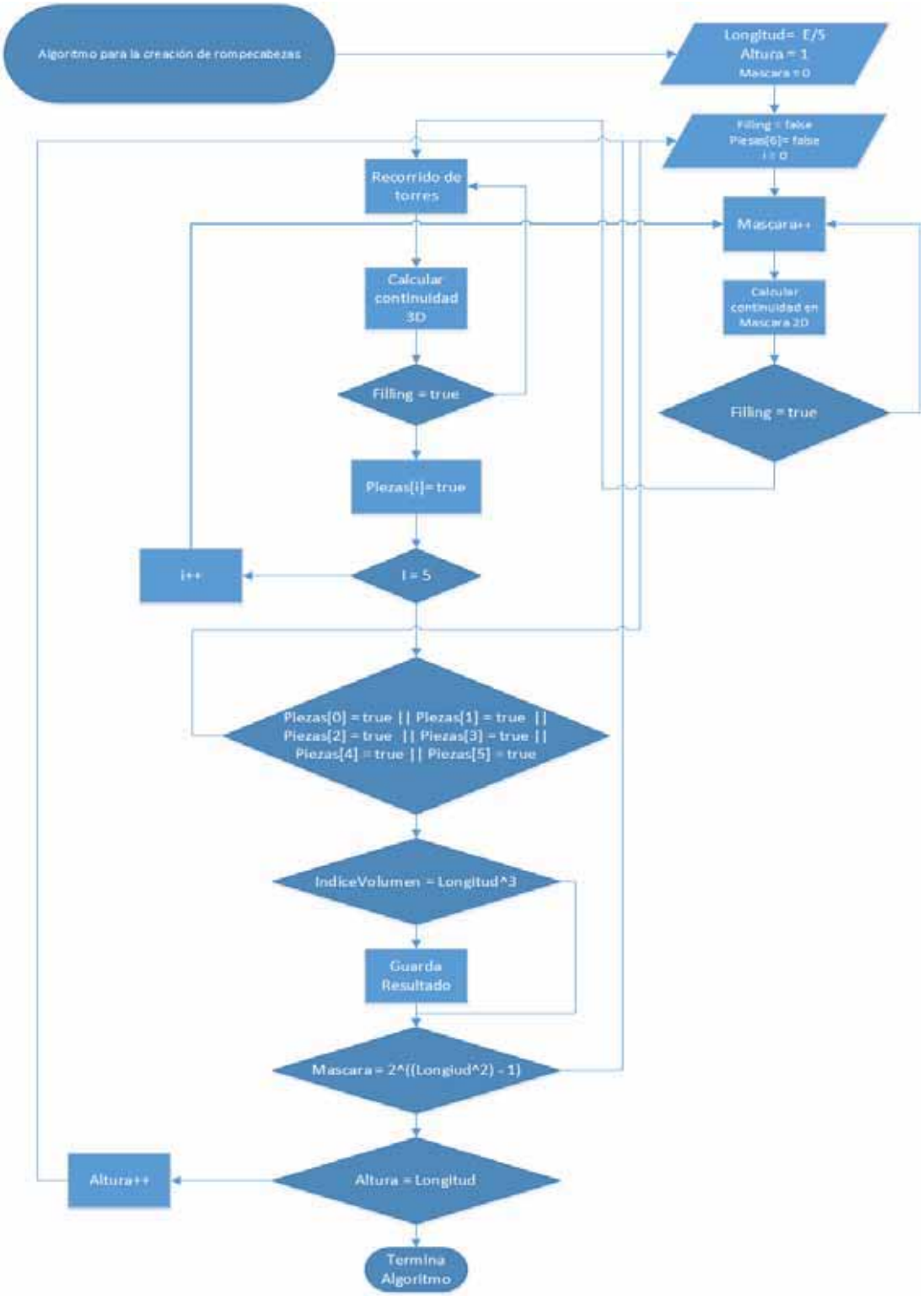

Figura 10.Diagrama de Flujo algoritmo para creación de las piezas.

6.5.1. Clase Rompecabezas

En esta clase se encuentran los algoritmos que dan funcionamiento principal a la creación de las piezas mediante las búsqueda de candidatos en un conjunto amplio de posibilidades delimitados por restricciones de continuidad entre los policubos de las pieza que serán en breve explicados por las funciones que los representan.

En las variables de esta clase se tiene la variable tipo de tipo char que sirve para seleccionar si se grafican o no las piezas del rompecabezas desde la clase principal, se tiene un apuntador a un Cubo *cubo = new Cubo() que nos sirve para utilizar el cubo creado en la clase principal para generar nuestras soluciones, también utilizamos las variables de tipo Cubo CajaDePolicubosX, CajaDePolicubosMenosX, CajaDePolicubosY, CajaDePolicubosMenosY, CajaDePolicubosZ, CajaDePolicubosMenosZ para almacenar las posibles combinaciones de piezas mientras se procesa el algoritmo, utilizamos las variables de tipo entero Indicex, IndiceMenosx, Indicey, IndiceMenosy, Indicez, IndiceMenosz para llevar un contador que nos sirve para marcar los cubos que se encuentran conectados mediante Flood Fill 2D,también se tiene un contador de tipo entero llamados IndiceTorresx, IndiceTorresMenosx, IndiceTorresy, IndiceTorresMenosy, IndiceTorresz, IndiceTorresMenosz que nos sirve para saber cuántos policubos se usaron para construir las torres de cada pieza para posteriormente verificar su continuidad mediante el algoritmo de Flood Fill 3D, la variable Longitud de tipo entero que guarda la longitud inicial del cubo principal para poder procesar las piezas, las variables siguientes de tipo entero alturax, alturaMenosx, alturay, alturaMenosy, alturaz, alturaMenosz que determinan la altura máxima que se utilizara para la construcción de las torres en 3D, el arreglo de tipo booleano Piezas[caras] con caras de tamaño 6 que sirve de indicador para saber las piezas que faltan por utilizar, una variable de tipo entero IndiceVolumen que nos sirve para saber si se cubrió el volumen completo de policubos en la construcción de las piezas, las variables de tipo booleano
mascarax[tamañomaximo],mascaraMenosx[tamañomaximo],mascaray[tamañomaximo],mascaraMenosy[tamañomaximo],mascaraz[tamañomaximo],mascaraMenosz[tamañomaximo] con tamañomaximo igual a 170 que es son los arreglos auxiliares para generar y utilizar las combinaciones posibles de cada cara para crear las piezas con nuestro algoritmo y la variable apuntador *Fecha de tipo char que contiene la fecha para la impresión de resultados.

En las funciones miembro iniciando por el constructor Rompecabezas(Cubo *Cubo, int longitud, char *fecha), que recibe como parámetros la dirección del Cubo, la longitud del Cubo y la fecha se asignan los valores de estos parámetros a sus miembros para su utilización en el algoritmo,

En la función miembro CrearRompecabezas() se tiene la clave del funcionamiento de la creación de las piezas que se muestra en la figura 10, en esta función se tienen dos procesos distintos sirve para el ingreso manual de las coordenadas para graficar un cubo descompuesto

en seis piezas con los valores de las máscaras como coordenadas 2D y seis alturas como coordenadas 3D mediante la entrada estándar del sistema, el otro proceso es para el cálculo automático de los candidatos a ser piezas mediante la combinación de ciclos y funciones que dan la estructura base al funcionamiento de nuestro algoritmo, con la función ResultRompecabezas() obtenemos los resultados de la creación de las piezas resultantes en un archivo y en la salida estándar, las siguientes seis funciones son invocada mediante la función display para así poder graficar las piezas resultantes, GrafCaraX(), GrafCaraMenosX(), GrafCaraY(), GrafCaraMenosY(), GrafCaraZ(), GrafCaraMenosZ() estas funciones llaman a la función ProcesarGrafica(Cubo CajaDePolicubos, int Indice, GLfloat color[]), con los parámetros específicos para cada pieza para procesar correctamente las piezas al ser graficadas, con la función ResetCubo(bool status) reiniciamos los valores tanto de las seis piezas como del Cubo inicial para recalcular el siguiente posible candidato para ser solución, la función ProcesarDatos(Cubo CajaDePolicubos, int Indice, char Cara[]), es llamada por la función antes mencionada ResultRompecabezas() con los parámetros correctos para las impresión de resultados de cada pieza(el código fuente se incluye en el apéndice),

6.5.2. Funciones específicas para el funcionamiento del algoritmo

En esta función miembro que es la inicial en nuestro algoritmo ProcesaFilling2D(Cubo &CajaDePolicubos, int &Indice, int i, int j, int k, int imascara, bool mascara[], bool target_color, bool replacement_color, char cara) y con sus parámetros se procesa el valor de una máscara con valor entero inicial en cero convirtiéndola a su equivalente en binario luego se seleccionan los bits marcados como true los cuales se utilizan para procesar los policubos de nuestra pieza mediante la función Filling2D y determinar si hay continuidad entre la selección de piezas, también se tiene un contador que indica el número de bits con valor uno para posteriormente compararlo con el contador de la función Filling2D(), dando como resultado un true si existe la continuidad o un false si no existe la continuidad provocando que se incremente la variable mascara y se repita el procedimiento dentro de un ciclo Do-While hasta que el conjunto de bits que representa una piezas tengan continuidad para pasar al proceso de construcción de torres.

En la función Filling2D(Cubo &CajaDePolicubos, int &Indice, int i, int j, int k, bool target_color, bool replacement_color, char cara) que es una versión modificada de el algoritmo de Flood Fill se recorre recursivamente las caras marcadas como true incrementado un contador que, si coincide con el contador de la función ProcesaFilling2D(), se tiene un candidato inicial en 2D permitiendo salir del ciclo antes mencionado y pasar a la construcción de las torres.

La siguiente parte de nuestro algoritmo es la de la construcción de las torres mediante la función ProcesaConstruccionDeTorres(Cubo &CajaDePolicubos, int &Indice, int altura, int i, int j, int k, int imascara, bool mascara[], bool target_color, bool replacement_color, char

cara) que recibe los mismos parámetros que las funciones que procesan las cara 2D pero con un parámetro adicional que es la altura, esta función procesa los valores de la máscara obtenidos anteriormente para proporcionárselos a la función ConstruccionTorres() que es la encargada de crear las torres para cada pieza, esta función le asigna el valor verdadero al arreglo indicador de las piezas utilizadas si la función ConstruccionTorres() creo torres con los parámetros utilizados.

La función ConstruccionTorres(Cubo &CajaDePolicubos, int &Indice, int altura, int i, int j, int k, bool target_color, bool replacement_color, char cara) utiliza la recursión y un proceso de construcción basado en Flood Fill, y como limite la altura de las torres para crear las torres hacia la profundidad de cada pieza con el fin de crear piezas tridimensionales llevando un contador de los policubos que esta función va absorbiendo para usar este contador en la prueba de conectividad en 3D con la función ProcesaFilling3D().

La función ProcesaFilling3D(Cubo &CajaDePolicubos, int &Indice, int i, int j, int k, int imascara, bool mascara[], bool target_color, bool replacement_color, char cara) procesa los datos de la misma manera que la función ProcesaFillin2D() solo que en lugar de llamar a la función Filling2D() invoca a la función Filling3D() para la verificación en tridimensional de las piezas.

Con la función Filling3D(Cubo &CajaDePolicubos, int &Indice, int i, int j, int k, bool target_color, bool replacement_color, char cara) se recorre la profundidad de cada pieza de manera recursiva llevando un conteo de las piezas que fueron seleccionadas para crear las torres y comparando ese valor con el valor del índice de torres construidas mediante la función ProcesaFilling3D dando como resultado verdadero si estos coinciden permitiendo salir de un ciclo Do-While permitiendo continuar con la construcción de la siguiente pieza.

Las siguientes dos funciones se utilizaron en todos los procesos para crear transformaciones lineales desde R1 a R2 como de R2 a R1 para poder mapear correctamente las coordenadas de nuestra mascara a las coordenadas de las posición de los policubos para cada cara en 2D MapeaDeR2aR1(int x, int y), MapeaDeR1aR2(int x,int Coordenadas[2]) (el código fuente se incluye en el apéndice).

6.6.Funciones para Graficación

6.6.1. Función AsignacionVariables

Esta función está dedicada exclusivamente a asignar los valores de las variables para las funciones especiales otorgadas al grafico como por ejemplo el movimiento de la cámara(el código fuente se incluye en el apéndice).

6.6.2. Función SetupRC

Esta función se utiliza para poder asignar el color del fondo de nuestro grafico como también asignar el tipo de luz y sus efectos y seleccionar la parte frontal del gráfico(el código fuente se incluye en el apéndice).

6.6.3. Función ChangeSize

Esta función es pasada como argumento a la función especial glutReshapeFunc() de OpenGL, y se encarga de manejar la perspectiva del gráfico, el modo de proyección y el punto visibilidad del grafico por la ventana(el código fuente se incluye en el apéndice).

6.6.4. Función Keyboard

Esta función es pasada como argumento a la función especial glutKeyboardFunc() de OpenGL, se definieron usos específicos de distintas teclas para el funcionamiento del grafico mediante un switch-case los cuales se mencionan y explican en el manual del usuario incluido en el apéndice(el código fuente se incluye en el apéndice).

6.6.5. Función SpecialKeys

Esta función es pasada como argumento a la función especial glutSpecialFunc() de OpenGL, Y está encargada de asignarle comportamientos a las teclas especiales como F1 las cuales van a visualizar distintos efectos a los gráficos mediante condiciones if, los cuales se mencionan y explican en el manual del usuario incluido en el apéndice (el código fuente se incluye en el apéndice).

6.6.6. Función Display

Esta función es pasada como argumento a la función especial glutDisplaylFunc() de OpenGL, esta función es la encargada de graficar nuestro cubo en pantalla , por medio de dos variables globales DesplegarCubo de tipo Cubo y DesplegarDescCubo de tipo Rompecabezas y las funciones RenderizarPolicubo(Cubo Cubo) y RenderizarDescCubo(Rompecabezas piezas) recibe los valores de los objetos Volumen y rompecabezas los cuales con las funciones de traficación creadas ya antes mencionadas se crea el grafico en pantalla.
Esta función utiliza la función glLookAt para posicionar la cámara y la función glTranslatef() para poder desplegar movimiento a las piezas del rompecabezas (el código fuente se incluye en el apéndice).

6.7.Complejidad computacional

6.7.1. Seudocódigo del algoritmo principal

```
For desde ix = 1 hasta N hacer
  For desde iMx = 1 hasta N hacer
    For  desde iy = 1 hasta N hacer
      For desde iMy = 1 hasta N hacer
        For desde iz = 1 hasta N hacer
          For desde iMz = 1 hasta N hacer
            Si (ix > N / 2 || iMx > N / 2 || iy > N / 2 || iMy >     N / 2 || iz > N / 2 || iMz > N / 2)
            entonces imascara = 0, imascarax = imascara, imascaraMenosx = imascara, imascaray
            = imascara, imascaraMenosy = imascara, imascaraz = imascara, imascaraMenosz =
            imascara.
            Hacer
                  Hacer
                        For desde i = 0 hasta 6 hacer
                        Hacer falso Piezas[i]
                        Fin
                  IndiceVolumen = 0
                  ResetCubo(target_color)
                        Hacer
                        Incrementar en uno imascarax
                        Repetir mientras ProcesaFilling2D() sea falso
                  alturax = N - ix
                        Hacer
                        Piezas[0] = ProcesaConstruccionDeTorres()
                        Decrementar en uno alturax
                        Repetir mientras ProcesaFilling3D() se falso
                        Hacer
                        Incrementar en uno imascaraMenosx
                        Repetir mientras ProcesaFilling2D() sea falso
                  alturaMenosx = N - iMx
                        Hacer
                        Piezas[1] = ProcesaConstruccionDeTorres()
                        Decrementar en uno alturaMenosx
                        Repetir mientras ProcesaFilling3D() se falso
                        Hacer
                        Incrementar en uno imascaray
                        Repetir mientras ProcesaFilling2D() sea falso
```

```
alturay = N - iy
        Hacer
        Piezas[2] = ProcesaConstruccionDeTorres()
        Decrementar en uno alturay
        Repetir mientras ProcesaFilling3D() se falso
        Hacer
        Incrementar en uno imascaraMenosy
        Repetir mientras ProcesaFilling2D() sea falso
alturaMenosy = N - iMy
        Hacer
        Piezas[3] = ProcesaConstruccionDeTorres()
        Decrementar en uno alturaMenosy
        Repetir mientras ProcesaFilling3D() se falso
        Hacer
        Incrementar en uno imascaraz
        Repetir mientras ProcesaFilling2D() sea falso
alturaz = N - iz
        Hacer
        Piezas[4] = ProcesaConstruccionDeTorres()
        Decrementar en uno alturaz
        Repetir mientras ProcesaFilling3D() se falso
        Hacer
        Incrementar en uno imascaraMenosz
        Repetir mientras ProcesaFilling2D() sea falso
alturaMenosz = N - iMz
        Hacer
        Piezas[5] = ProcesaConstruccionDeTorres()
        Decrementar en uno alturaMenosz
        Repetir mientras ProcesaFilling3D() se falso
Repetir mientras Piezas[0] o Piezas[1] o Piezas[2] o Piezas[3] o Piezas[4] o
Piezas[5] sean igual a falso
Si (IndiceVolumen = N^3) entonces Guarda resultado en archivo
```

Repetir mientras (imascarax sea menor que ((2^(N^2)) - 1).

6.7.2. Cálculo de la complejidad computacional

En la parte más profunda de nuestro algoritmo encontramos 6 ciclos do-while para la verificación 2D y 6 ciclos do-while para la construcción de las torres y su verificación 3D.

Las funciones ProcesaFilling2D que se encuentra dentro de la verificación 2D tiene en su código como ciclo de mayor profundidad el siguiente:

```
For  I = 1 hasta n has //en donde  es la longitud N al cuadrado
        Si mascara[I] = verdadero entonces
        Filling2D()
```

La función filling2D es del orden O(4^n) en donde n es la longitud al cuadrado, ya que su ejecución realiza cuatro llamadas recursivas de longitud máxima de N^2. Por lo tanto el tiempo de ejecución de la función ProcesaFilling2D es de O(n*4^n) en donde n es N^2 por lo tanto es O(N^2*4^(N^2)).

Con un proceso semejante se determinó que la función ProcesaConstruccionDeTorres() contiene la siguiente función:

```
For  I = 1 hasta n has (en donde  es la longitud N al cuadrado)
        Si mascara[I] = verdadero entonces
        ConstruccionDeTorres()
```

En donde la función ConstruccionDeTorres tiene una complejidad computacional de O(1) ya que las torres no dependen de la longitud por lo tanto la función ProcesaConstruccionDeTorres tiene una complejidad computacional de O(N^2).

Y por último la función ProcesaFilling3D() tiene una ejecución igual a proceso de construcción de torres por lo tanto deducimos que el tiempo de ejecución es de O(N^2).

Los ciclos encargados de procesar la verificación 2D dependen del resultado de la función PocesarFilling2D() que a su vez esta depende del valor de la máscara por lo tanto el tiempo de ejecución de estos ciclos es de O(2^(N^2)), debido que estos ciclos se ejecutan hasta alcanzar el valor máximo y en combinación con la función de ProcesaFilling2D, el tiempo de ejecución total de esta parte del algoritmo es de O(N^2*4^(N^2)*2^(N^2)) que se simplifica en O(N^2*(8^(N^2))) .
Los ciclos encargados de la construcción de las torres dependen del Resultado de la función ProcesaFilling3D la cual depende del resultado de la construcción de las torres y esta depende de la longitud inicial para las alturas, la cual es N por lo tanto el valor del tiempo de ejecución de esta ciclo es igual a O(N*N^2).

Finalmente tenemos que, al tener seis ciclos principales para las alturas y que el mayor tiempo de ejecución dentro de estos ciclos es el del ciclo para procesar la verificación 2D, tenemos que con estos datos el algoritmo tiene una complejidad total aproximada de O(N^6*N^2*(8^(N^2)) que se simplifica en O((N^8)*(8^(N^2))).

7. Resultados

En la carpeta contenedora del archivo ejecutable se encuentra una carpeta con el nombre de Resultados en la cual se incluyen tres archivos de texto, el primero con el nombre de Resultado.txt incluye los resultados de la creación inicial de un cubo así como también los resultados de la creación de las seis piezas del rompecabezas cuando se ingresa manualmente las coordenadas separados por la fecha y hora en que se realizó cada ejecución.

Los archivos ResultadoAutomatico3x3.txt y ResultadoAutomatico5x5.txt guardan los resultados obtenidos del recorrido de todo el algoritmo para candidatos de piezas, es decir, incluye las coordenadas de los candidatos a ser piezas de rompecabezas que cumplen con las especificaciones de continuidad, también imprime el tiempo de ejecución del algoritmo y también separa por la fecha y hora cada ejecución realizada.

7.1.Resultados 3x3

En la ejecución del algoritmo para un cubo de 3x3 se lograron obtener 1435 posibles candidatos a ser piezas del rompecabezas en un tiempo aproximado de 38.079 segundos de los cuales se tomaron 16 para verificar la efectividad del algoritmo los 1435 resultados se proporcionan en el archivo correspondiente (ResultadoAutomatico3x3.txt).

7.2.Desplazamiento ortogonal de las Piezas

En las figuras 11, 12 y 13 se muestran el desplazamiento ortogonal de las piezas para separarlas mediante la tecla Q, este desplazamiento también puede ser revertido mediante la tecla A.

Figura 11.Desplazamiento 1.

Figura 12.Desplazamiento 2.

Figura 13.Desplazamiento 3.

Coordenadas de los Resultados 3x3		
Índice Resultado	Coordenadas(mascara) 2D	Coordenadas(Alturas) 3D
1 (Figura 14)	27	1:3:2:2:3:2
2 (Figura 15)	27	3:2:1:3:1:2
3 (Figura 16)	59	1:1:1:1:3:3
4 (Figura 19)	59	1:1:3:1:1:3
5 (Figura 18)	59	3:1:3:2:1:3
6 (Figura 17)	155	2:3:3:3:2:2
7 (Figura 22)	155	3:2:1:3:3:3
8 (Figura 20)	187	1:3:2:1:3:3
9 (Figura 21)	187	3:3:3:2:1:1
10 (Figura 25)	315	1:1:1:2:3:3
11 (Figura 24)	315	2:1:3:3:3:2
12 (Figura 23)	411	1:3:2:2:3:1
13 (Figura 26)	411	2:3:2:3:3:3
14 (Figura 27)	411	2:3:3:3:3:1
15 (Figura 28)	443	1:1:1:1:3:1
16 (Figura 29)	443	1:1:3:2:3:1

Tabla 1.Resultado 3x3.

Figura 14.Resultado 3x3 1.

Figura 15.Resultado 3x3 2.

Figura 16.Resultado 3x3 3.

Figura 19. Resultado 3x3 4.

Figura 18.Resultado 3x3 5.

Figura 17.Resultado 3x3 6.

Figura 22. Resultado 3x3 7.

Figura 20.Resultado 3x3 8.

Figura 21.Resultado 3x3 9.

Figura 25.Resultado 3x3 10.

Figura 24.Resultado 3x3 11.

Figura 23.Resultado 3x3 12.

Figura 26.Resultado 3x3 13.

Figura 27.Resultado 3x3 14.

Figura 28.Resultado 3x3 15.

Figura 29.Resultado 3x3 16.

7.3.Resultados 5x5

En la ejecución del algoritmo para un cubo de 5x5 se lograron obtener solamente 30 posibles candidatos debido a que el tiempo de ejecución del algoritmo para un cubo de 5x5 se torna muy grande, también se encontró la forma de ingresar coordenadas de manera manual utilizando una altura de longitud máxima fija en combinación de las máscara obtenidas para obtener otros resultados diferentes a los calculados automáticamente pero basados en esos resultados, de los cuales se tomaron 26 para verificar la efectividad del algoritmo los 30 resultados se proporcionan en el archivo correspondiente (ResultadoAutomatico5x5.txt).

Coordenadas de los resultados 5x5		
Índice Resultado	Coordenadas(mascara) 2D	Coordenadas(Alturas) 3D
1 (Figura 32)	507375	5:5:5:5:5:5
2 (Figura 31)	1031663	5:5:5:5:5:5
3 (Figura 30)	1048047	5:5:5:5:5:5
4 (Figura 41)	1048559	1:1:1:1:5:5
5 (Figura 40)	1048559	3:4:5:3:5:5
6 (Figura 39)	1048559	5:5:5:5:5:5
7 (Figura 38)	8895983	5:5:5:5:5:5
8 (Figura 37)	9420271	5:5:5:5:5:5
9 (Figura 36)	9436655	5:5:5:5:5:5
10 (Figura 35)	9437167	5:5:5:5:5:5
11 (Figura 34)	13090287	5:5:5:5:5:5
12 (Figura 33)	13614575	5:5:5:5:5:5
13 (Figura 44)	13630959	5:5:5:5:5:5
14 (Figura 43)	13631471	5:5:5:5:5:5
15 (Figura 42)	15187439	5:5:5:5:5:5
16 (Figura 47)	15711727	5:5:5:5:5:5
17 (Figura 46)	15728111	5:5:5:5:5:5
18 (Figura 45)	15728623	5:5:5:5:5:5
19 (Figura 50)	17825775	1:1:1:1:5:4
20 (Figura 49)	17825775	3:3:3:3:5:5
21 (Figura 48)	26214383	1:1:1:1:5:3
22 (Figura 53)	26214383	4:4:4:4:5:4
23 (Figura 52)	30408687	1:1:1:1:5:2
24 (Figura 51)	30408687	1:2:3:4:5:4
25 (Figura 55)	32505839	1:1:1:1:5:1
26 (Figura 54)	32505839	3:3:5:5:3:3

Tabla 2.Resultado 5 x5.

Figura 32.Resultado 5x5 1.

Figura 31.Resultado 5x5 2.

Figura 30.Resultado 5x5 3.

Figura 41.Resultado 5x5 4.

Figura 40.Resultado 5x5 5.

Figura 39. Resultado 5x5 6.

Figura 38.Resultado 5x5 7.

Figura 37.Resultado 5x5 8.

Figura 36.Resultado 5x5 9.

Figura 35.Resultado 5x5 10.

Figura 34. Resultado 5x5 11.

Figura 33.Resultado 5x5 12.

Figura 44.Resultado 5x5 13.

Figura 43.Resultado 5x5 14.

Figura 42.Resultado 5x5 15.

Figura 47.Resultado 5x5 16.

Figura 46.Resultado 5x5 17.

Figura 45.Resultado 5x5 18.

Figura 50.Resultado 5x5 19.

Figura 49.Resultado 5x5 20.

Figura 48.Resultado 5x5 21.

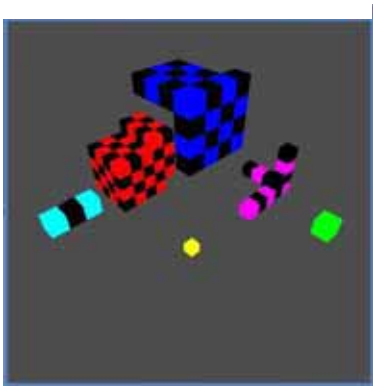

Figura 53.Resultado 5x5 22.

Figura 52Resultado 5x5 23.

Figura 51.Resultado 5x5 24.

Figura 55.Resultado5x5 25.

Figura 54.Resultado5x5 26.

8. Análisis y discusión de resultados

Los resultados obtenidos en el presente proyecto reflejan cómo se logró crear un conjunto de posibles soluciones para la generación de las piezas del rompecabezas como también la incorporación de varios métodos algorítmicos par la creación del mismo. Lamentablemente las limitaciones de este algoritmo se vieron reflejadas ya que en el conjunto de resultados para el cubo de 3x3x3 con un valor de 1435 posibles candidatos, se encontraron descomposiciones creadas que se repitieron aunque estas estaban referenciadas por coordenadas diferentes, también se notó que muchos de las coordenadas de los resultados para la máscara en 2D eran los mismos, pero con alturas diferentes aun así cumpliendo con el objetivo, generando piezas diferentes.

Para longitudes mayores a 3 se encontró que debido a la complejidad computacional del programa que se calculó con un valor de O((N^8)*(8^(N^2))) y la arquitectura del sistema, es muy lento procesar los datos, para obtener los resultados hasta que finalice el programa.

Se logró incluir restricciones para obtener un grupo reducido de candidatos para un cubo de 5x5x5 con resultados satisfactorios, obteniendo una lista de aproximadamente 30 candidatos, que en conjunto con resultados obtenidos de pruebas manuales para la búsqueda de candidatos con los valores de las coordenadas 2D y manipulando coordenadas de las alturas, se logró mostrar 26 candidatos que cumplen con las especificaciones del algoritmo.

El presente algoritmo está construido para calcular hasta una longitud máxima 13 debido a que se utilizó memoria estática, pero se considera que al mejorar el diseño del programa implementándolo con memoria dinámica se podría calcular sin problemas, una longitud de tamaño N.

También se pretende buscar mejoras en la eficiencia de los métodos para obtener la conectividad de las piezas ya que al utilizar el método de Flood Fill recursivo, influyó en que

se incrementaran los tiempos de ejecución. Esto mediante la sustitución de este algoritmo por su análogo iterativo como Scanline Fill o Bundary Fill.

También se buscar incrementar las restricciones para lograr reducir el conjunto de búsqueda y también evitar soluciones del tipo elementales, simétricas, repetidas, etc.

9. Conclusiones

Se pudo construir un conjunto de búsqueda en este algoritmo que sea de un tamaño lo bastante amplio para realizar la localización de los candidatos, también se logró implementar satisfactoriamente la mayoría de los objetivos principales que se postularon en el proyecto que fueron:

1. Diseñar un algoritmo que permita generar las piezas del rompecabezas en base a la teoría de POLICUBOS.
2. Diseñar las estructuras de datos para guardar las piezas resultantes del rompecabezas.
3. Diseñar algoritmo de búsqueda que permita descomponer el cubo en piezas con todas sus partes linealmente conectadas.
4. Visualizar el proceso de descomposición de cada combinación encontrada a través de un programa implementado mediante OpenGL.

Debido a las limitaciones implementadas mencionadas en el capítulo 2.2 que habla de la formulación del problema en su punto 5 sobre la construcción de las alturas, se encontró que el conjunto de soluciones se redujo, al grado de no incluir el conjunto de candidatos los cuales las piezas estén construidas entrelazadas, quitando la característica de las piezas que al ser sacudidas mecánicamente no se desprendan.

Con los objetivos alcanzados se logró construir los candidatos a piezas y estos son almacenados a un archivo.

9.1. Trabajos Futuros

Existen mejoras a realizar para generar resultados más óptimos los cuales se pretenden tratar para mejorar el presente proyecto.

1. Reducir el tiempo de ejecución modificando las funciones del programa para reducir el tiempo de ejecución por ejemplo Flood Fiil.
2. Modificar la construcción de las torres para que el conjunto de búsqueda incluya la combinación de distintas alturas en los policubos candidatos mencionado en el capítulo 2.2 en su punto 5.
3. Eliminar la restricción 3 del capítulo 2.2, en donde se limita, que de cada cara se obtiene una sola pieza, pudiendo obtener de una cara dos o más piezas.
4. Eliminar la restricción 4 del capítulo 2.2 cambiando la operación de sacado de las piezas de forma ortogonal a cualquier desplazamiento disponible para desenganchar las piezas si están entrelazadas.
5. Introducir una pieza con una forma inicial para que el algoritmo parta de esta entrada para realizar los cálculos correspondientes.

10. Bibliografía

[1] Policubos. Arq. Roberto H. Serrentino, Arq. Hernán Molina, *"Arquitectura modular basada en la teoría de policubos"*, Facultad de Arquitectura y Urbanismo, Universidad Nacional de Tucumán, http://cumincades.scix.net/data/works/att/8a44.content.pdf.

[2] Luis Joyanes Aguilar, *"Programación en C++. Algoritmos, estructuras de datos y objetos"*, Primera Edición, Editorial Mc. Graw Hill, España, 2000.

[3] S. Fazzini, *"Algoritmo de relleno por difusión (Flood Fill Algorithm)", http://es.wikipedia.org/wiki/Algoritmo_de_relleno_por_difusi%C3%B3n*, http://en.wikipedia.org/wiki/Flood_fill.

[4] OpenGl, *"The Industry's Foundation for High Performance Graphics",* http://www.opengl.org/sdk/docs/man3.

[5] *"Cubo Soma",*
http://es.wikipedia.org/wiki/Cubo_Soma.

[6] *"Cubo de Bedlam",*
http://es.wikipedia.org/wiki/Cubo_Soma.

[7] *"Slothouber-Graatsma puzzle",*
http://en.wikipedia.org/wiki/Slothouber%E2%80%93Graatsma_puzzle.

[8] Guerequeta y Vallecillo, *"Técnicas de diseño de algoritmos.",*
Universidad de Málaga.

[9] *"Combinaciones de bits",*
http://es.wikipedia.org/wiki/Bit.

[10] Alan H. Schoen, *"Set of blocks for packing a cube",*
Https://docs.google.com/viewer?url=patentimages.storage.googleapis.com/pdfs/US20010035606.pdf, Carbondale Illinois US.

[11] J. A. Rupérez Padrón y M. García Deníz, *"Graduación en la dificultad en el cubo Soma (I)",* Sociedad Canaria Issac Newton de profesores de matematicas, http://www.sinewton.org/numeros/numeros/75/Juegos_01.pdf.

[12] J. A. Rupérez Padrón y M. García Deníz, *"Graduación en la dificultad en el cubo Soma (II)",* Sociedad Canaria Issac Newton de profesores de matematicas, http://grupoalquerque.es/ferias/2011/archivos/pdf/matemagia_de_gardner.pdf.

[13] J. A. Rupérez Padrón y M. García Déniz, *"Las disecciones de cubos. Secuenciación en tamaños y dificultad como una propuesta didáctica. Estudio del cubo 2x2x2. Algunas presentaciones de cubos 3x3x3 y 4x4x4, y un reto."*, Sociedad Canaria Issac Newton de profesores de matemáticas, http://www.sinewton.org/numeros/numeros/72/Juegos_01.pdf.

[14] Franco Guidi Polanco, *"Diagrama de clases UML"*, Universidad Católica de Valparaíso, Chile, http://eii.ucv.cl/pers/guidi/cursos/estructuras/pdf/SE-DiagramasDeClasesUML.pdf.

11. Apéndices

11.1. Listado de archivos

Cubo.cpp
Policubo.cpp
Rompecabezas.cpp
Principal.cpp
AsignacionVariables.h
ChangeSize.h
Cubo.h
Display.h
Keyboard.h
Policubo.h
Rompecabezas.h
SetupRC.h
SpecialKeys.h
Stdafx.h
Targetver.h
Resultado.txt
ResultadoAutomatico.txt
ResultadoAutomatico3x3.txt
ResultadoAutomatico.5x5.txt
freeglut.dll
glew32.dll
glew32mx.dll
msvcr120d.dll
ProyectoTerminal.exe

11.2. Manual del Usuario

El programa ejecutable funciona en versiones de sistema operativo Windows 7 x64, Windows 8 x64 y Windows 8.1 x64.

11.2.1. Ejecución del programa

Localizar la carpeta con el nombre de Ejecutables en la carpeta del proyecto, en su interior se encuentra la carpeta Descomposición de cubo la cual en su interior, se encuentra el archivo ProyectoTerminal.exe al cual hay que dar doble clic para su ejecución.

Posteriormente se visualizara la pantalla de inicio en consola de comandos en donde tendrá que ingresar una Longitud N entre 1 y 13 debido a las restricciones del programa (Figura 56).

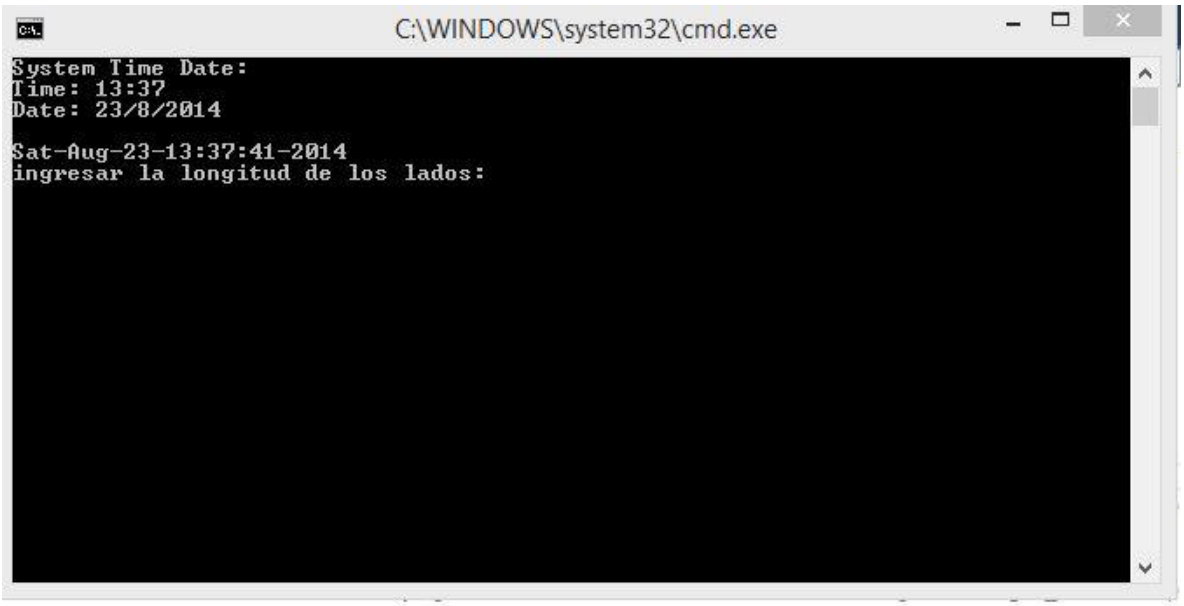

Figura 56.Ingreso de longitud.

Nota: Para cálculos automáticos ingresar solo el valor de tres o máximo cuatro debido a los tiempos de ejecución del programa.

11.2.2. Calculo de datos

Al ingresar la longitud deseada se muestra la opción para calcular los datos automáticamente o la opción de ingresar manualmente coordenadas para visualizar una descomposición en específico. En la cual debe ingresar “s” para realizar los cálculos automáticos o “n” para

pasar al modo manual, si usted presiona cualquier otra tecla se visualizara un cubo de una pieza (Figura 57).

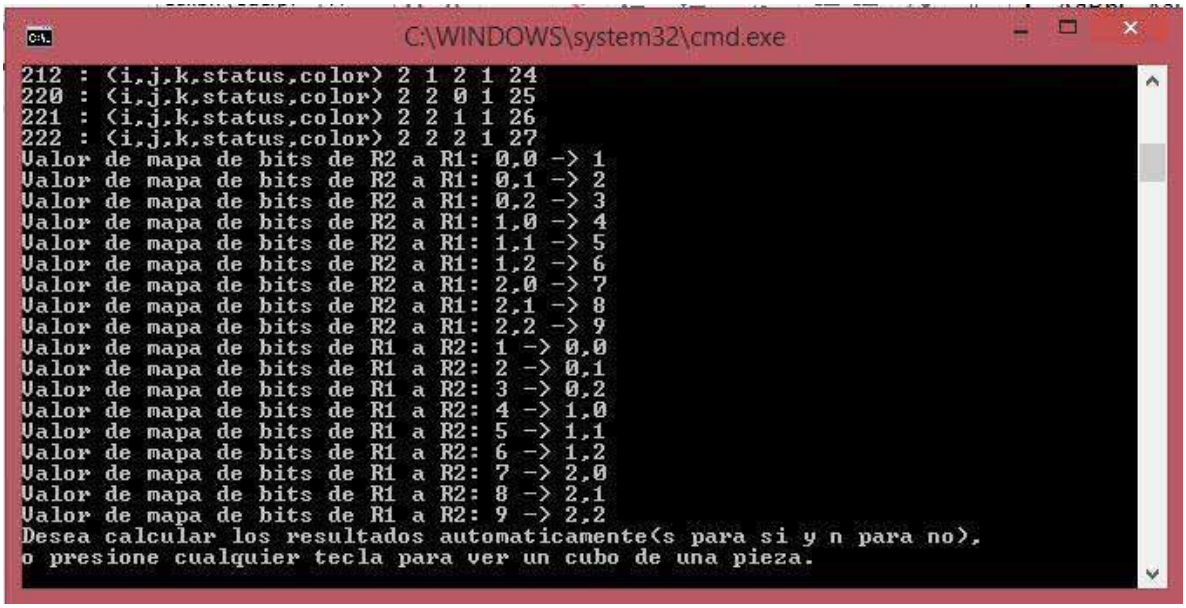

Figura 57.Calculo de datos.

11.2.2.1. Selección de cálculos automáticos

La selección de cálculos automáticos (Figura 58) se dio debido a que usted selecciono “s” en la pantalla del inciso anterior (Se recomienda no realizar cálculos automáticos para valores mayores a 3 debido al tiempo de ejecución del programa).

Figura 58.Cálculos automaticos.

En esta pantalla se muestra la ejecución de cálculos automáticos.

11.2.2.1.1. Cálculos automáticos con longitud mayor a tres

Cuando selecciona la opción de cálculos automáticos para valores mayores a tres se da la opción de ingresar las alturas manualmente (Figura 59) si usted presiona “s”, con el fin de en cuanto el algoritmo encuentre el primer resultado que sea candidato a piezas para el rompecabezas lo almacena en el archivo y así no se tiene que esperar a que se ejecute todo el algoritmo, las alturas tienen que ser valores entre uno y la longitud ingresada. En caso de que presionar cualquier otra tecla se ejecutara el algoritmo hasta que haya recorrido todas las posibilidades.

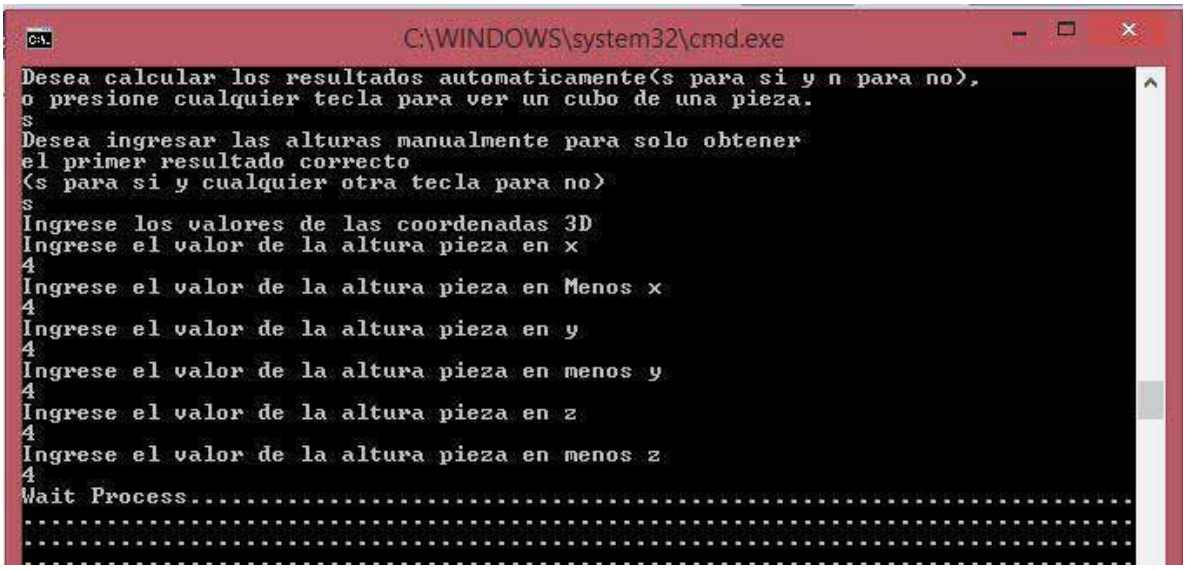

Figura 59.Calculos automáticos con longitud mayor a 3.

11.2.2.2. Selección de cálculos Manuales

La selección de cálculos Manuales se dio debido a que usted selecciono "n" en la pantalla de cálculo de datos. En esta parte del programa debe ingresar una valor numérico entero dentro del rango que está dado por valores entre uno y dos elevado al cuadrado de la longitud ingresada menos uno, en la coordenada 2D y posteriormente seis valores para las alturas de cada una de las piezas en un rango entre 1 y la longitud ingresada (Figura 60).

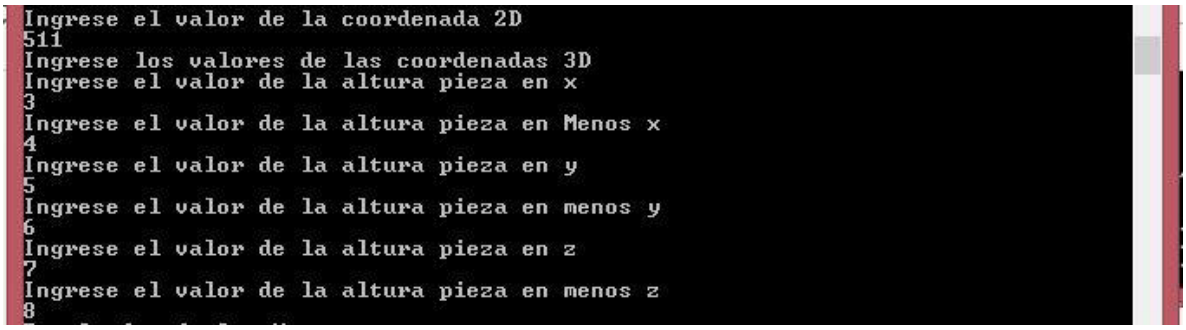

Figura 60.Ingreso manual de coordenadas.

Dando como resultado la graficación de la pieza con esas coordenadas sin importar que no cumplan con las restricciones de los cálculos automáticos (Figura 61).

En esta sección utilizar los valores de las coordenadas impresas en el archivo destino generadas en los cálculos automáticos, para visualizar las piezas con los resultados obtenidos.

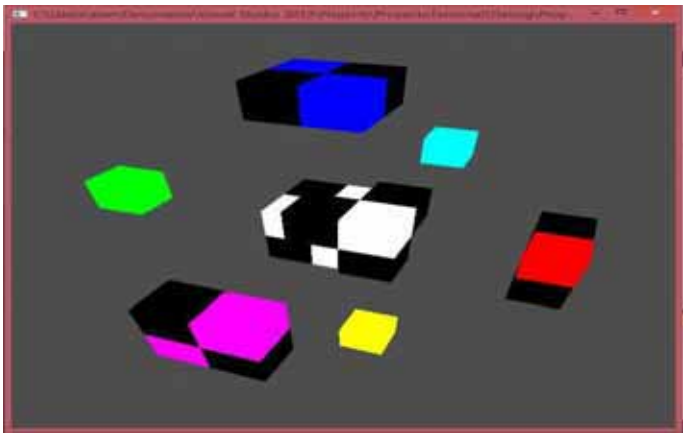

Figura 61.Cubo en piezas.

11.2.3. Visualización de Cubo

Al presionar cualquier tecla en la pantalla de modo de cálculo de resultados, no se realiza ningún cálculo y se visualiza un cubo de la longitud ingresada (Figura 62).

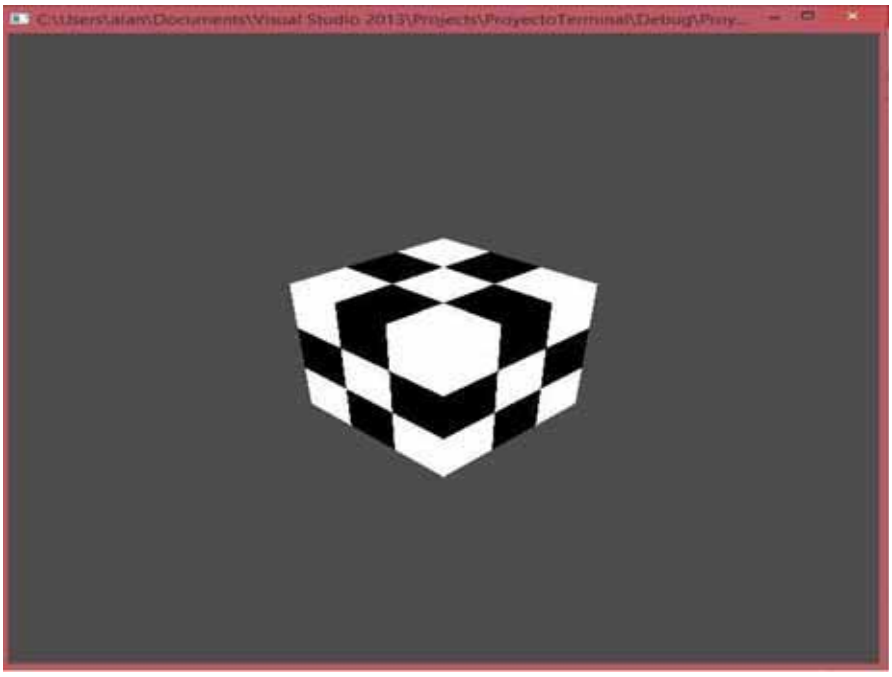

Figura 62.Cubo.

11.2.4. Efectos de visualización del Cubo

Al Visualizar el cubo inicial o al visualizar la descomposición del cubo en piezas se tienen varios efectos visuales.

- Presione la tecla ESC si desea cerrar el grafico del cubo.

11.2.4.1. Posicionamiento de Cámara

- Con la flecha arriba se rota la cámara alrededor del origen en el plano Oyz.
- Con la flecha abajo se rota la cámara alrededor del origen en el plano Oyz.
- Con la flecha izquierda se rota la cámara alrededor del origen en el plano Oxz.
- Con la flecha derecha se rota la cámara alrededor del origen en el plano Oxz.
- Con la tecla F6 se invierte el ojo de la cámara con respecto al eje x.
- Con la tecla F7 se invierte el ojo de la cámara con respecto al eje y.
- Con la tecla F8 se invierte el ojo de la cámara con respecto al eje z.
- Con la tecla F9 se invierte el ojo de la cámara con respecto a los tres ejes.
- Con la tecla W o w se reduce la distancia de la cámara al origen (close up).
- Con la tecla S o s se incrementa la distancia de la cámara al origen.
- Con la tecla 1 cambia la posición del vector de up en la cámara de a x.
- Con la tecla 2 cambia la posición del vector de up en la cámara de a y.
- Con la tecla 3 cambia la posición del vector de up en la cámara de a z.
- Con la tecla Q o q Desplazar piezas hacia afuera (Solo en el ingreso manual).
- Con la tecla A o a Desplazar piezas hacia el centro (Solo en el ingreso manual).
-

11.2.4.2. Cambios de color para la visualización del Cubo inicial

- Con la tecla 4 cambia el color del cubo a rojo.
- Con la tecla 5 cambia el color del cubo a verde.
- Con la tecla 6 cambia el color del cubo a azul.
- Con la tecla 7 cambia el color del cubo a Cian.
- Con la tecla 8 cambia el color del cubo a Magenta.
- Con la tecla 9 cambia el color del cubo a Amarillo.
- Con la tecla 0 cambia el color del cubo a Blanco.

11.2.4.3. Efectos de visualización especiales

- Con la tecla F1 se cambia si se visualiza la cara posterior o no (Figuras 63 y 64).

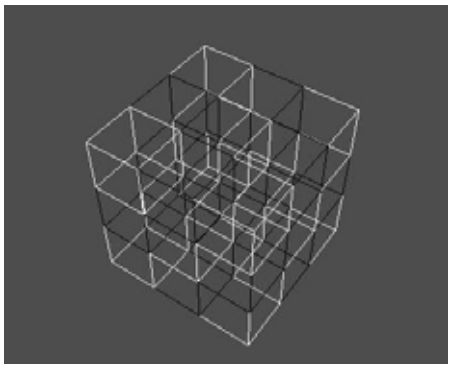

Figura 64.Vista Cara Posterior.

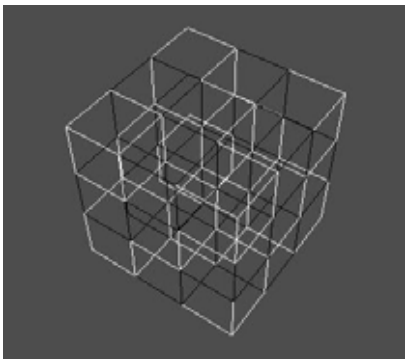

Figura 63.Sin vista cara posterior.

- Con la tecla F2 se visualiza el efecto de profundidad (Figura 65).

Figura 65.Profundidad.

- Con la tecla F3 se muestra el contorno del cubo (Figura 66).

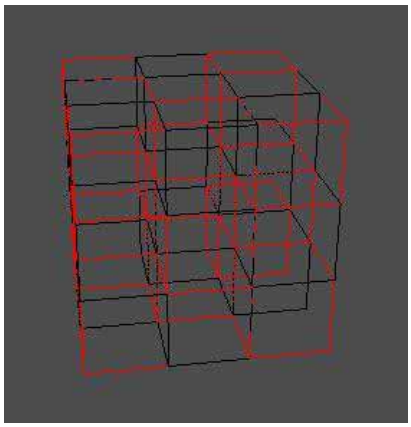

Figura 66.Contorno.

- Con la tecla F5 se muestran efectos de luz y sombra (Figura 67).

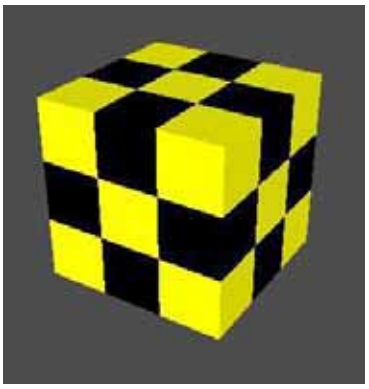

Figura 67.Efectos de luz.

11.3. Código fuente

Para poder compilar el código fuente es necesario crear un proyecto nuevo en Visual Studio 2013 con la versión más reciente de Visual C++, crear un proyecto nuevo en modo de consola de Windows 32 que incluya las bibliotecas de OpenGL, freeglut y glew.

11.3.1. Código clase principal

```
#include <iostream>
#include"AsignacionVariables.h"
#include "Cubo.h"
#include"stdafx.h"
#include"SpecialKeys.h"
#include"ChangeSize.h"
#include"Rompecabezas.h"
#include"Display.h"
#include"SetupRC.h"
#include"Keyboard.h"
#include<fstream>
#include<ctime>
//#include<pthread.h>
using namespace std;
//Psicion de camara virtual
#define EX                0.0f
#define EY                0.0f
#define EZ                5.0f
#define CX                0.0f
#define CY                0.0f
#define CZ                0.0f
#define UX                0.0f
#define UY                1.0f
#define UZ                0.0f
//Diferenciales para cambios en posicion de camara
#define DELTA            0.08f
#define DELTA_R            0.08f

int main(int argc, char ** argv)
{
	/* Optener System Time Date*/
	cout << "System Time Date:\n";
	time_t tSac;	// instante actual
	time(&tSac);
	struct tm tmP;
	localtime_s(&tmP,&tSac);

	int hour = tmP.tm_hour;
	int min = tmP.tm_min;
	int day = tmP.tm_mday;
	int mon = tmP.tm_mon + 1;
	int year = tmP.tm_year + 1900;
	//Construccion nombre de Archivo
	char Fecha[25];
	ctime_s(Fecha,26,&tSac);
	Fecha[24] = '\0';
	Fecha[19] = '-';
	Fecha[10] = '-';
```

```
        Fecha[7] = '-';
        Fecha[3] = '-';
        cout << "Time: " << hour << ":" << min << "\n";
        cout << "Date: " << day << "/" << mon << "/" << year << "\n";
        cout << "\n" << Fecha << "\n";
        //Archivo
        ofstream Archivo("Resultado/Resultado.txt", ios::app);
        Archivo <<
"////////////////////////////////////////////////////////////////////////////////
////////////////////////////////\n";
        Archivo << "Fecha: " << Fecha << ".\n";
        Archivo <<
"////////////////////////////////////////////////////////////////////////////////
////////////////////////////////\n";
        //Cierre de Archivo
        Archivo.close();
        //Numero de lados de policubo
        int Longitud = 0;
        //Tamaño ventana
        GLsizei wSize = 700, hSize = 700;
        //Efectos de Gaficos
        GLboolean    bCull = glIsEnabled(GL_CULL_FACE), bDepth =
glIsEnabled(GL_DEPTH_TEST), bOutline = (GLboolean)true,
        bLight = glIsEnabled(GL_LIGHTING);
        GLenum shademode = GL_FLAT;
        //Asignacion de valores predefinidos para incrementos de camara
        GLdouble    ex = EX, ey = EY, ez = EZ, cx = CX, cy = CY, cz = CZ,ux = UX, uy
= UY,uz = UZ,
        delta = DELTA, deltaR = DELTA_R;
        //Color de policubos
        GLfloat Color[3] = { 1.0, 1.0, 1.0 };
        //Desplazamiento piezas
        GLfloat Desp = 0;
        cout << "ingresar la longitud de los lados:\n";
        cin >> Longitud; cout << "\n";
        //ojo de la camara posicionado dependiendo el tamaño del cubo
        ex = Longitud * 2.5;
        ey = Longitud * 2.5;
        ez = Longitud * 2.5;
        //Centra el ojo de la camara a la posicion dependiendo la longitud
        cx = Longitud / 2;
        cy = Longitud / 2;
        cz = Longitud / 2;

        TeclasEspeciales(cx, cy, cz, ex, ey, ez, ux, uy, uz, deltaR, delta, bCull,
bDepth, bOutline, shademode, bLight,Color,Desp);
        /*TeclasEspecialesDisplay(cx, cy, cz, ex, ey, ez, deltaR, delta, bCull,
bDepth, bOutline, shademode);*/

        //Creacion objeto cubo
        Cubo Volumen = Cubo(Longitud, Color);
        //Crando Cubo de dimensiones especificadas
        Volumen.CrearCubo();
        //Visualixzacion de datos
        Volumen.ResultCubo();
        //Graficar cubo mediante policubos en la funcion display
        RenderizarPolicubo(Volumen);
```

```
	// Separa el cubo en piezas con la direccion del cubo simula que dscompone el
mismo cubo
	Rompecabezas rompecabezas = Rompecabezas(&Volumen, Longitud/*,
Color*/,Fecha);
	rompecabezas.CrearRompecabezas();
	//dibuja las piezas
	if (rompecabezas.tipo == 'n'){
		RenderizarDescCubo(rompecabezas);
		RenderizarPolicubo(Volumen);
	}
	//REsultado de piezas
	rompecabezas.ResultRompecabezas();
	//prueba para la funcion
	Volumen.ResultCubo();
	//Funciones para graficacion
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
	glutInitWindowSize(wSize, hSize);
	glutInitWindowPosition(1, 1);
	glutCreateWindow(argv[0]);
	SetupRC();
	glutDisplayFunc(Display);
	glutReshapeFunc(ChangeSize);
	glutSpecialFunc(SpecialKeys);
	glutKeyboardFunc(keyboard);
	glutMainLoop();
	return 0;
}
```

11.3.2. Código clase policubo

```
#include"Policubo.h"
#include"stdafx.h"

// asigna los valores publicos a los privados
Policubo::Policubo(int x, int y, int z, bool Ocupado){
	posicionx = x;
	posiciony = y;
	posicionz = z;
	ocupado = Ocupado;
}
//constructor por default
Policubo::Policubo(){

}
// destructor
Policubo::~Policubo(){
	//cout << "\nBorrando Datos....\n";
}
//Leer posicion de policubo
void Policubo::GetPosicion(int &x, int &y, int &z){
	x = posicionx;
	y = posiciony;
	z = posicionz;
}
//Guardar posicion de policubo
```

```
void Policubo::SetPosicion(int &x, int &y, int &z){
	posicionx = x;
	posiciony = y;
	posicionz = z;

}
//Leer si esta ocupada esa posicion
bool Policubo::GetStatus(){
	return ocupado;
}
//Guarda si se ocupa esa posicion
void Policubo::SetStatus(bool &Ocupado){
	ocupado = Ocupado;
}
//Guarda el color
int Policubo::GetColor(){
	return color;
}
//Regresa el color
void Policubo::SetColor(int &Color){
	color = Color;
}
void Policubo::Cuadrado(){
	glBegin(GL_QUADS);		//cara frontal
	glVertex3f(-0.5f, -0.5f, 0.5f); //V1
	glVertex3f(0.5f, -0.5f, 0.5f);//v2
	glVertex3f(0.5f, 0.5f, 0.5f);//v3
	glVertex3f(-0.5f, 0.5f, 0.5f);//v4
	glEnd();
}
//Grafica un Policubo
void Policubo::GrafPolicubo(){

	//GRafica el policubo mediante un cuadrado mediante transformaciones
	glPushMatrix();
	//Translaciones para formar el cubo mediante policubos
	glTranslatef(posicionx, posiciony, posicionz);
	//Cara Frontal/////////////
	glPushMatrix();
	Cuadrado();
	glPopMatrix();
	//Cara trasera////////////
	glPushMatrix();
	glRotatef(180, 0, 1, 0);
	Cuadrado();
	glPopMatrix();
	//Cara izquierda//////////
	glPushMatrix();
	glRotatef(90, 0, 1, 0);
	Cuadrado();
	glPopMatrix();
	//Cara Derecha//////////
	glPushMatrix();
	glRotatef(270, 0, 1, 0);
	Cuadrado();
	glPopMatrix();
	//Cara Abajo//////////
	glPushMatrix();
```

```
	glRotatef(90, 1, 0, 0);
	Cuadrado();
	glPopMatrix();
	//Cara Arriba//////////
	glPushMatrix();
	glRotatef(270, 1, 0, 0);
	Cuadrado();
	glPopMatrix();
	glPopMatrix();
}
//Borra los datos de cada policubo
void Policubo::Reset(){
	posicionx = 0;
	posiciony = 0;
	posicionz = 0;
	ocupado = false;
}
```

11.3.3. Código clase cubo

```
#include "Policubo.h"
#include "Cubo.h"
#include"stdafx.h"
#include<iostream>
#include<fstream>
using namespace std;

//Funcion cunstructor
Cubo::Cubo(int longitud, GLfloat *Color){
	Longitud = longitud;
	color = Color;
}
//Constructor por default
Cubo::Cubo(){

}
//obtencion de policubo
Policubo Cubo::GetPolicubo(int &i, int &j, int &k){
	return cubo[i][j][k];
}
//Regresa el estado del policubo
void Cubo::SetPolicubo(int &i, int &j, int &k, bool &Status, int &Color){
	 cubo[i][j][k] = Policubo(i,j,k,Status);
	 cubo[i][j][k].SetColor(Color);
}
//destructor
Cubo::~Cubo(){

}
//funcion q crea el cubo
void Cubo::CrearCubo(){
	int i = 0, j = 0, k = 0, iPivot = 1;
	bool Status = true;
	for (i = 0; i < Longitud; i++){
		for (j = 0; j < Longitud; j++){
```

```
                        for (k = 0; k < Longitud; k++){
                                cubo[i][j][k] = Policubo(i, j, k, Status);
                                cubo[i][j][k].SetColor(iPivot);
                                iPivot++;
                        }
                }
        }
}
        //muestra los resultados de la creacion del cubo logico
void Cubo::ResultCubo(){
        int i = 0, j = 0, k = 0;
        int iaux = 0, jaux = 0, kaux = 0;
        ofstream Archivo("Resultado/Resultado.txt",ios::app);
        cout << "Resultados creacion del cubo: \n";
        Archivo <<
"////////////////////////////////////////////////////////////////////////////////
////////////////////////////////\n";
        Archivo << "Resultados creacion del cubo: \n";
        for (i = 0; i < Longitud; i++){
                for (j = 0; j < Longitud; j++){
                        for (k = 0; k < Longitud; k++){
                                cubo[i][j][k].GetPosicion(iaux, jaux, kaux);

                                cout << i << j << k << " " << ": (i,j,k,status,color) "
<< iaux << " " << jaux << " " << kaux << " " << cubo[i][j][k].GetStatus() << " " <<
cubo[i][j][k].GetColor() << "\n";
                                Archivo << i << j << k << " " << ": (i,j,k,status,color)
" << iaux << " " << jaux << " " << kaux << " " << cubo[i][j][k].GetStatus() << " "
<< cubo[i][j][k].GetColor() << "\n";
                        }
                }
        }
        Archivo.close();
}
//Grafica el cubo
void Cubo::GrafCubo(){
        int iPivot;
        int i = 0, j = 0, k = 0;
        for (i = 0; i < Longitud; i++){
                for (j = 0; j < Longitud; j++){
                        for (k = 0; k < Longitud; k++){
                                if (cubo[i][j][k].GetStatus() == true){
                                        iPivot = cubo[i][j][k].GetColor();
                                        if ((iPivot % 2) == 0)
                                                glColor3f(0.0f, 0.0f, 0.0f);//color negro
                                        else
                                                glColor3fv(color);//color rojo
                                        //              Incrementa el pivote para cambiar el
color del siguiente policubo
                                        cubo[i][j][k].GrafPolicubo();
                                }
                                else{}

                        }
                }
        }
}
```

11.3.4. Código clase Rompecabezas

```
#include "Rompecabezas.h"
#include"Cubo.h"
#include"stdafx.h"
#include<math.h>
#include<iostream>
#include <string>         // std::string
#include <bitset>         // std::bitset
#include <fstream>
#include<ctime>

using namespace std;

//Variables para graficar las Piezas
GLfloat colorx[3] = { 1.0, 0.0, 0.0 }, colorMenosx[3] = { 0.0, 1.0, 0.0 }, colory[3]
= { 0.0, 0.0, 1.0 };
GLfloat colorMenosy[3] = { 1.0, 1.0, 0.0 }, colorz[3] = { 1.0, 0.0, 1.0 },
colorMenosz[3] = { 0.0, 1.0, 1.0 };
//Funcion cunstructor
Rompecabezas::Rompecabezas(Cubo *Cubo, int longitud, char *fecha){
	cubo = Cubo;
	Longitud = longitud;
	Fecha = fecha;
}
//Constructor por default
Rompecabezas::Rompecabezas(){

}
//destructor
Rompecabezas::~Rompecabezas(){
}
//Pone el cubo al valor boleano de la variable status
void Rompecabezas::ResetCubo(bool status){
	bool Status = false;
	for (int i = 0; i < Longitud; i++){
		for (int j = 0; j < Longitud; j++){
			for (int k = 0; k < Longitud; k++){
				cubo->cubo[i][j][k].SetStatus(status);
				CajaDePolicubosX.cubo[i][j][k].SetStatus(Status);
				CajaDePolicubosMenosX.cubo[i][j][k].SetStatus(Status);
				CajaDePolicubosY.cubo[i][j][k].SetStatus(Status);
				CajaDePolicubosMenosY.cubo[i][j][k].SetStatus(Status);
				CajaDePolicubosZ.cubo[i][j][k].SetStatus(Status);
				CajaDePolicubosMenosZ.cubo[i][j][k].SetStatus(Status);
			}
		}
	}
}

//Funcion que mapea las coordenadas de las piezas a la coordenada de la mascara
int Rompecabezas::MapeaDeR2aR1(int x, int y){

	int contX = 1;
```

```
		for (int i = 0; i < Longitud; i++){
			for (int j = 0; j < Longitud; j++){
				if (i == x && j == y){
					return contX;
				}
				contX++;
			}
		}
		return 0;
}

//Funcion que mapea la coordenada de la mascara a las cordenadas de la pieza
int Rompecabezas::MapeaDeR1aR2(int x,int Coordenadas[2]){

	int contX = 1;
	for (int i = 0; i < Longitud; i++){
		for (int j = 0; j < Longitud; j++){
			if (contX ==x){
				Coordenadas[0] = i;
				Coordenadas[1] = j;
			}
			contX++;
		}
	}
	return 0;
}
//Crea rompecabezas
void Rompecabezas::CrearRompecabezas(){

	int i = 0, j = 0, k = 0, imascara = 0;
	int Coordenadas[2], imascarax = 0, imascaraMenosx = 0, imascaray = 0,
imascaraMenosy = 0, imascaraz = 0, imascaraMenosz = 0;
	bool target_color = true, filling = false, replacement_color = false, Volumen
= false;
	int ix = 0, iMx = 0, iy = 0, iMy = 0, iz = 0, iMz = 0;
	int ixAux = 0, iMxAux = 0, iyAux = 0, iMyAux = 0, izAux = 0, iMzAux = 0;
	char NombreArchivo = '\0';
	//Escribir en fichero
	ofstream Archivo("Resultado/Resultado.txt", ios::app);
	Archivo <<
"////////////////////////////////////////////////////////////////////////////////
////////////////////////////////\n";
	Archivo << "Resultados cubo de Longitud: " << Longitud << "x" << Longitud <<
".\n";
	Archivo <<
"////////////////////////////////////////////////////////////////////////////////
////////////////////////////////\n";
	Archivo << "Valores de mapeo: \n";
	for (int i = 0; i < Longitud; i++){
		for (int j = 0; j < Longitud; j++){
			cout << "Valor de mapa de bits de R2 a R1: " << i << "," << j <<
" -> " << MapeaDeR2aR1(i, j) << "\n";
			Archivo << "Valor de mapa de bits de R2 a R1: " << i << "," << j
<< " -> " << MapeaDeR2aR1(i, j) << "\n";
		}
	}
```

```
	Archivo <<
"////////////////////////////////////////////////////////////////////////////////
////////////////////////////////\n";
	for (int i = 1; i <= Longitud * Longitud; i++){
		MapeaDeR1aR2(i, Coordenadas);
		cout << "Valor de mapa de bits de R1 a R2: " << i << " -> " <<
Coordenadas[0] << "," << Coordenadas[1] << "\n";
		Archivo << "Valor de mapa de bits de R1 a R2: " << i << " -> " <<
Coordenadas[0] << "," << Coordenadas[1] << "\n";
	}
	cout << "Desea calcular los resultados automaticamente(s para si y n para
no),\no presione cualquier tecla para ver un cubo de una pieza.\n";
	cin >> tipo;
	if (tipo == 'n'){
		//////////////////////////////////////////7///////Ingreso manual de
valores ////////////////////////////////////////
		cout << "Ingrese el valor de la coordenada 2D\n";
		cin >> imascara;
		cout << "Ingrese los valores de las coordenadas 3D\n";
		cout << "Ingrese el valor de la altura pieza en x\n";
		cin >> ix;
		cout << "Ingrese el valor de la altura pieza en Menos x\n";
		cin >> iMx;
		cout << "Ingrese el valor de la altura pieza en y\n";
		cin >> iy;
		cout << "Ingrese el valor de la altura pieza en menos y\n";
		cin >> iMy;
		cout << "Ingrese el valor de la altura pieza en z\n";
		cin >> iz;
		cout << "Ingrese el valor de la altura pieza en menos z\n";
		cin >> iMz;
		//indices de las  mascaras
		imascarax = imascara; imascaraMenosx = imascara; imascaray = imascara;
imascaraMenosy = imascara; imascaraz = imascara; imascaraMenosz = imascara;
		//Proceso de generacion de plas piezas para el rompecabezas
		//pone las piezas en falso
		for (int i = 1; i <= Longitud * Longitud; i++){
			Piezas[i] = false;
		}
		IndiceVolumen = 0;
		ResetCubo(target_color);
		//Proceso para la cara X
		filling = false;
		filling = ProcesaFilling2D(CajaDePolicubosX, Indicex, Longitud - 1, 0,
0, imascarax, mascarax, target_color, replacement_color, 'X');
		//construccion de torres
		filling = false;
		alturax = Longitud - ix;
		Piezas[0] = ProcesaConstruccionDeTorres(CajaDePolicubosX,
IndiceTorresx, alturax, Longitud - 1, 0, 0, imascarax, mascarax, target_color,
replacement_color, 'X');
		filling = ProcesaFilling3D(CajaDePolicubosX, IndiceTorresx, Longitud -
1, 0, 0, imascarax, mascarax, target_color, replacement_color, 'X');
		////Proceso para la cara Menos X
		filling = false;
		filling = ProcesaFilling2D(CajaDePolicubosMenosX, IndiceMenosx, 0, 0,
0, imascaraMenosx, mascaraMenosx, target_color, replacement_color, 'x');
		//construccion de torres
```

```
            filling = false;
            alturaMenosx = Longitud - iMx;
            Piezas[1] = ProcesaConstruccionDeTorres(CajaDePolicubosMenosX,
IndiceTorresMenosx, alturaMenosx, 0, 0, 0, imascaraMenosx, mascaraMenosx,
target_color, replacement_color, 'x');
            filling = ProcesaFilling3D(CajaDePolicubosMenosX, IndiceTorresMenosx,
0, 0, 0, imascaraMenosx, mascaraMenosx, target_color, replacement_color, 'x');
            ////Proceso para la cara Y
            filling = false;
            filling = ProcesaFilling2D(CajaDePolicubosY, Indicey, 0, Longitud - 1,
0, imascaray, mascaray, target_color, replacement_color, 'Y');
            //construccion de torres
            filling = false;
            alturay = Longitud - iy;
            Piezas[2] = ProcesaConstruccionDeTorres(CajaDePolicubosY,
IndiceTorresy, alturay, 0, Longitud - 1, 0, imascaray, mascaray, target_color,
replacement_color, 'Y');
            filling = ProcesaFilling3D(CajaDePolicubosY, IndiceTorresy, 0, Longitud
- 1, 0, imascaray, mascaray, target_color, replacement_color, 'Y');
            ////Proceso para la cara menos Y
            filling = false;
            filling = ProcesaFilling2D(CajaDePolicubosMenosY, IndiceMenosy, 0, 0,
0, imascaraMenosy, mascaraMenosy, target_color, replacement_color, 'y');
            //construccion de torres
            filling = false;
            alturaMenosy = Longitud - iMy;
            Piezas[3] = ProcesaConstruccionDeTorres(CajaDePolicubosMenosY,
IndiceTorresMenosy, alturaMenosy, 0, 0, 0, imascaraMenosy, mascaraMenosy,
target_color, replacement_color, 'y');
            filling = ProcesaFilling3D(CajaDePolicubosMenosY, IndiceTorresMenosy,
0, 0, 0, imascaraMenosy, mascaraMenosy, target_color, replacement_color, 'y');
            ////Proceso para la cara Z
            filling = false;
            filling = ProcesaFilling2D(CajaDePolicubosZ, Indicez, 0, 0, Longitud -
1, imascaraz, mascaraz, target_color, replacement_color, 'Z');
            //construccion de torres
            filling = false;
            alturaz = Longitud - iz;
            Piezas[4] = ProcesaConstruccionDeTorres(CajaDePolicubosZ,
IndiceTorresz, alturaz, 0, 0, Longitud - 1, imascaraz, mascaraz, target_color,
replacement_color, 'Z');
            filling = ProcesaFilling3D(CajaDePolicubosZ, IndiceTorresz, 0, 0,
Longitud - 1, imascaraz, mascaraz, target_color, replacement_color, 'Z');
            ////Proceso para la cara menos Z
            filling = false;
            filling = ProcesaFilling2D(CajaDePolicubosMenosZ, IndiceMenosz, 0, 0,
0, imascaraMenosz, mascaraMenosz, target_color, replacement_color, 'z');
            //construccion de torres
            filling = false;
            alturaMenosz = Longitud - iMz;
            Piezas[5] = ProcesaConstruccionDeTorres(CajaDePolicubosMenosZ,
IndiceTorresMenosz, alturaMenosz, 0, 0, 0, imascaraMenosz, mascaraMenosz,
target_color, replacement_color, 'z');
            filling = ProcesaFilling3D(CajaDePolicubosMenosZ, IndiceTorresMenosz,
0, 0, 0, imascaraMenosz, mascaraMenosz, target_color, replacement_color, 'z');
            Archivo <<
"////////////////////////////////////////////////////////////////////////////////
////////////////////////////////\n";
```

```
            Archivo << "Valores de las Coordenadas" << Longitud << "x" << Longitud
<< "x" << Longitud << ": \n";
            Archivo << "Resultado Coordenadas 2D = " << imascarax << " Resultado
coordenadas 3D = " << ix << ":" << iMx << ":" << iy << ":" << iMy << ":" << iz <<
":" << iMz << ".\n";
      }
      if (tipo == 's'){
            //Tiempo de ejecucion del programa
            clock_t t;
            //Variable para ingreso manual en el cubo de 5x5
            char ingresoManual = '\0';
            t = clock();
            //Archivo de resultados
            ofstream Archivo;
            if (Longitud == 3 ){
                  Archivo.open("Resultado/ResultadoAutomatico3x3.txt", ios::app);
                  ixAux = 1, iMxAux = 1, iyAux = 1, iMyAux = 1, izAux = 1, iMzAux
= 1;
            }
            else if (Longitud == 5){
                  Archivo.open("Resultado/ResultadoAutomatico5x5.txt", ios::app);
                  cout << "Desea ingresar las alturas manualmente para solo
obtener\nel primer resultado correcto\n(s para si y cualquier otra tecla para
no)\n";
                  cin >> ingresoManual;
                  if (ingresoManual == 's'){
                        cout << "Ingrese los valores de las coordenadas 3D\n";
                        cout << "Ingrese el valor de la altura pieza en x\n";
                        cin >> ixAux;
                        cout << "Ingrese el valor de la altura pieza en Menos
x\n";
                        cin >> iMxAux;
                        cout << "Ingrese el valor de la altura pieza en y\n";
                        cin >> iyAux;
                        cout << "Ingrese el valor de la altura pieza en menos
y\n";
                        cin >> iMyAux;
                        cout << "Ingrese el valor de la altura pieza en z\n";
                        cin >> izAux;
                        cout << "Ingrese el valor de la altura pieza en menos
z\n";
                        cin >> iMzAux;
                  }
                  else{
                        ixAux = 1, iMxAux = 1, iyAux = 1, iMyAux = 1, izAux = 1,
iMzAux = 1;
                  }
            }
            else if (Longitud > 3 || Longitud != 5){
                  Archivo.open("Resultado/ResultadoAutomatico.txt", ios::app);
                  cout << "Desea ingresar las alturas manualmente para solo
obtener\nel primer resultado correcto\n(s para si y cualquier otra tecla para
no)\n";
                  cin >> ingresoManual;
                  if (ingresoManual == 's'){
                        cout << "Ingrese los valores de las coordenadas 3D\n";
                        cout << "Ingrese el valor de la altura pieza en x\n";
                        cin >> ixAux;
```

```
                        cout << "Ingrese el valor de la altura pieza en Menos
x\n";
                        cin >> iMxAux;
                        cout << "Ingrese el valor de la altura pieza en y\n";
                        cin >> iyAux;
                        cout << "Ingrese el valor de la altura pieza en menos
y\n";
                        cin >> iMyAux;
                        cout << "Ingrese el valor de la altura pieza en z\n";
                        cin >> izAux;
                        cout << "Ingrese el valor de la altura pieza en menos
z\n";
                        cin >> iMzAux;
                }
                else{
                        ixAux = 1, iMxAux = 1, iyAux = 1, iMyAux = 1, izAux = 1,
iMzAux = 1;
                }
        }
        else{
                Archivo.open("Resultado/ResultadoAutomatico.txt", ios::app);
                ixAux = 1, iMxAux = 1, iyAux = 1, iMyAux = 1, izAux = 1, iMzAux
= 1;
        }
        cout << "Wait Process......";
        Archivo <<
"////////////////////////////////////////////////////////////////////////////////
////////////////////////////////\n";
        Archivo << "Fecha: " << Fecha << ".\n";
        Archivo <<
"////////////////////////////////////////////////////////////////////////////////
////////////////////////////////\n";
        Archivo << "Cooredenadas de los resultados " << Longitud << "x" <<
Longitud << "x" << Longitud << ": \n";
        int I = 1;
        bool salida = false;
        for (int ix = ixAux; ix <= Longitud; ix++){
                if (salida)
                        break;
                for (int iMx = iMxAux; iMx <= Longitud; iMx++){
                        if (salida)
                                break;
                        for (int iy = iyAux; iy <= Longitud; iy++){
                                if (salida)
                                        break;
                                for (int iMy = iMyAux; iMy <= Longitud; iMy++){
                                        if (salida)
                                                break;
                                        for (int iz = izAux; iz <= Longitud; iz++){
                                                if (salida)
                                                        break;
                                                for (int iMz = iMzAux; iMz <=
Longitud; iMz++){
                                                        if (salida)
                                                                break;
                                                        if (ix > Longitud / 2 || iMx >
Longitud / 2 || iy > Longitud / 2 || iMy > Longitud / 2 || iz > Longitud / 2 || iMz
> Longitud / 2){
```

```
                                                        imascara = 0;
                                                        //indices de las
mascaras
                                                        imascarax = imascara;
imascaraMenosx = imascara; imascaray = imascara; imascaraMenosy = imascara;
imascaraz = imascara; imascaraMenosz = imascara;
                                                        do{
                                                                //Proceso de
generacion de plas piezas para el rompecabezas
                                                                do{
                                                                        //pone las
piezas en falso
                                                                        for (int i
= 1; i <= Longitud * Longitud; i++){

        Piezas[i] = false;
                                                                        }


        IndiceVolumen = 0;

        ResetCubo(target_color);
                                                                        //Proceso
para la cara X
                                                                        do{

        filling = false;

        imascarax++;

        filling = ProcesaFilling2D(CajaDePolicubosX, Indicex, Longitud - 1, 0, 0,
imascarax, mascarax, target_color, replacement_color, 'X');
                                                                        } while
(!filling);


        //construccion de torres
                                                                        alturax =
Longitud - ix;
                                                                        do{

        filling = false;

        Piezas[0] = ProcesaConstruccionDeTorres(CajaDePolicubosX, IndiceTorresx,
alturax, Longitud - 1, 0, 0, imascarax, mascarax, target_color, replacement_color,
'X');

        filling = ProcesaFilling3D(CajaDePolicubosX, IndiceTorresx, Longitud - 1, 0,
0, imascarax, mascarax, target_color, replacement_color, 'X');

        alturax--;
                                                                        } while
(!filling);


        ////Proceso para la cara Menos X

                                                                        do{
```

```
        filling = false;

        imascaraMenosx++;

        filling = ProcesaFilling2D(CajaDePolicubosMenosX, IndiceMenosx, 0, 0, 0,
imascaraMenosx, mascaraMenosx, target_color, replacement_color, 'x');
                                                                        } while
(!filling);


        //construccion de torres

        alturaMenosx = Longitud - iMx;
                                                                        do{

        filling = false;


        Piezas[1] = ProcesaConstruccionDeTorres(CajaDePolicubosMenosX,
IndiceTorresMenosx, alturaMenosx, 0, 0, 0, imascaraMenosx, mascaraMenosx,
target_color, replacement_color, 'x');

        filling = ProcesaFilling3D(CajaDePolicubosMenosX, IndiceTorresMenosx, 0, 0,
0, imascaraMenosx, mascaraMenosx, target_color, replacement_color, 'x');

        alturaMenosx--;
                                                                        } while
(!filling);


        ////Proceso para la cara Y

                                                                        do{

        filling = false;

        imascaray++;

        filling = ProcesaFilling2D(CajaDePolicubosY, Indicey, 0, Longitud - 1, 0,
imascaray, mascaray, target_color, replacement_color, 'Y');
                                                                        } while
(!filling);


        //construccion de torres
                                                                        alturay =
Longitud - iy;
                                                                        do{

        filling = false;

        Piezas[2] = ProcesaConstruccionDeTorres(CajaDePolicubosY, IndiceTorresy,
alturay, 0, Longitud - 1, 0, imascaray, mascaray, target_color, replacement_color,
'Y');

        filling = ProcesaFilling3D(CajaDePolicubosY, IndiceTorresy, 0, Longitud - 1,
0, imascaray, mascaray, target_color, replacement_color, 'Y');
```

```
        alturay--;
                                                                    } while
(!filling);


        ////Proceso para la cara menos Y
                                                                    do{

        filling = false;

        imascaraMenosy++;

        filling = ProcesaFilling2D(CajaDePolicubosMenosY, IndiceMenosy, 0, 0, 0,
imascaraMenosy, mascaraMenosy, target_color, replacement_color, 'y');
                                                                    } while
(!filling);


        //construccion de torres

        alturaMenosy = Longitud - iMy;
                                                                    do{

        filling = false;

        Piezas[3] = ProcesaConstruccionDeTorres(CajaDePolicubosMenosY,
IndiceTorresMenosy, alturaMenosy, 0, 0, 0, imascaraMenosy, mascaraMenosy,
target_color, replacement_color, 'y');

        filling = ProcesaFilling3D(CajaDePolicubosMenosY, IndiceTorresMenosy, 0, 0,
0, imascaraMenosy, mascaraMenosy, target_color, replacement_color, 'y');

        alturaMenosy--;
                                                                    } while
(!filling);


        ////Proceso para la cara Z
                                                                    do{

        filling = false;

        imascaraz++;

        filling = ProcesaFilling2D(CajaDePolicubosZ, Indicez, 0, 0, Longitud - 1,
imascaraz, mascaraz, target_color, replacement_color, 'Z');
                                                                    } while
(!filling);


        //construccion de torres
                                                                    alturaz =
Longitud - iz;
                                                                    do{
```

```
        filling = false;

        Piezas[4] = ProcesaConstruccionDeTorres(CajaDePolicubosZ, IndiceTorresz,
alturaz, 0, 0, Longitud - 1, imascaraz, mascaraz, target_color, replacement_color,
'Z');

        filling = ProcesaFilling3D(CajaDePolicubosZ, IndiceTorresz, 0, 0, Longitud -
1, imascaraz, mascaraz, target_color, replacement_color, 'Z');

        alturaz--;
                                                                        } while
(!filling);


        ////Proceso para la cara menos Z

                                                                        do{

        filling = false;

        imascaraMenosz++;

        filling = ProcesaFilling2D(CajaDePolicubosMenosZ, IndiceMenosz, 0, 0, 0,
imascaraMenosz, mascaraMenosz, target_color, replacement_color, 'z');
                                                                        } while
(!filling);


        //construccion de torres

        alturaMenosz = Longitud - iMz;
                                                                        do{

        filling = false;

        Piezas[5] = ProcesaConstruccionDeTorres(CajaDePolicubosMenosZ,
IndiceTorresMenosz, alturaMenosz, 0, 0, 0, imascaraMenosz, mascaraMenosz,
target_color, replacement_color, 'z');


        filling = ProcesaFilling3D(CajaDePolicubosMenosZ, IndiceTorresMenosz, 0, 0,
0, imascaraMenosz, mascaraMenosz, target_color, replacement_color, 'z');

        alturaMenosz--;
                                                                        } while
(!filling);
                                                                    } while
((!Piezas[0] || !Piezas[1] || !Piezas[2] || !Piezas[3] || !Piezas[4] ||
!Piezas[5]));
                                                                    if
(IndiceVolumen == (int)pow(Longitud, 3)){
                                                                        Archivo <<
I << ": " << "Resultado Coordenadas 2D = " << imascarax  << " Resultado coordenadas
3D = " << ix << ":" << iMx << ":" << iy << ":" << iMy << ":" << iz << ":" << iMz <<
".\n";
                                                                        I++;
```

```
                                                                        if
(Longitud > 3){
        cout << "/";
                                                                            if
(ingresoManual == 's'){
        salida = true;
        break;
                                                                            }
                                                                        }
                                                                    }
                                                                    if (Longitud >
3)
                                                                        cout <<
".";
                                                                } while (imascarax <
((int)pow(2, pow(Longitud, 2))) - 1);
                                                            }
                                                            cout << ".";
                                                        }
                                                    }
                                                }
                                            }
                                        }
                                    }
                Archivo <<
"////////////////////////////////////////////////////////////////////////////////
////////////////////////////////\n";
                Archivo << "Numero de soluciones: " << I - 1 << "\n";
                t = clock() - t;
                Archivo << "Tiempo de Ejecucion: " << (float)t/CLOCKS_PER_SEC <<"
segundos." << "\n";
                Archivo << "Tiempo de Ejecucion: " << ((float)t / CLOCKS_PER_SEC)/60 <<
" minutos." << "\n";
                Archivo << "Tiempo de Ejecucion: " << (((float)t / CLOCKS_PER_SEC) /
60) / 60 << " horas." << "\n";
                cout << "\nNumero de soluciones: " << I - 1 << "\n";
                Archivo.close();
        }
        Archivo <<
"////////////////////////////////////////////////////////////////////////////////
////////////////////////////////\n";
        cout << "Resultados de las Mascaras: \n";
        Archivo << "Resultados de las Mascaras: \n";
        cout << "\nMascara X: " << imascarax << "\n";
        Archivo << "Mascara X: " << imascarax << "\n";
        cout << "Mascara menos X: " << imascaraMenosx << "\n";
        Archivo << "Mascara menos X: " << imascaraMenosx << "\n";
        cout << "Mascara Y: " << imascaray << "\n";
        Archivo << "Mascara Y: " << imascaray << "\n";
        cout << "Mascara Menos y: " << imascaraMenosy << "\n";
        Archivo << "Mascara Menos y: " << imascaraMenosy << "\n";
        cout << "Mascara Z: " << imascaraz << "\n";
        Archivo << "Mascara Z: " << imascaraz << "\n";
        cout << "Mascara Menos z: " << imascaraMenosz << "\n";
```

```
	Archivo << "Mascara Menos z: " << imascaraMenosz << "\n";
	Archivo <<
"////////////////////////////////////////////////////////////////////////////////
////////////////////////////////\n";
	Archivo << "Resultados creacion de las piezas: \n";
	Archivo.close();
}
//Procesa cada piezas para la verificacion de 3D
bool Rompecabezas::ProcesaFilling3D(Cubo &CajaDePolicubos, int &Indice, int i, int
j, int k, int imascara, bool mascara[], bool target_color, bool replacement_color,
char cara){

	//Pone la vandera de prueba de conectividad en falso
	bool filling = false;
	//Coordenadas de mapeo de R1 a R2
	int Coordenadas[2];

	//prueba de conectividad
	//Resetea el valor del indice
	int indice = 0;
	for (int I = 1; I <= (int)pow(Longitud, 2); I++){
		if (mascara[I] == true){
			//Resetea el valor del indice
			//indice = 0;
			MapeaDeR1aR2(I, Coordenadas);
			if (cara == 'X'){
				Filling3D(CajaDePolicubos, indice, i, Coordenadas[0],
Coordenadas[1], target_color, replacement_color, cara);
			}
			if (cara == 'x'){
				Filling3D(CajaDePolicubos, indice, i, Coordenadas[0],
Coordenadas[1], target_color, replacement_color, cara);
			}
			if (cara == 'Y'){
				Filling3D(CajaDePolicubos, indice, Coordenadas[0], j,
Coordenadas[1], target_color, replacement_color, cara);
			}
			if (cara == 'y'){
				Filling3D(CajaDePolicubos, indice, Coordenadas[0], j,
Coordenadas[1], target_color, replacement_color, cara);
			}
			if (cara == 'Z'){
				Filling3D(CajaDePolicubos, indice, Coordenadas[0],
Coordenadas[1], k, target_color, replacement_color, cara);
			}
			if (cara == 'z'){
				Filling3D(CajaDePolicubos, indice, Coordenadas[0],
Coordenadas[1], k, target_color, replacement_color, cara);
			}
			//break;
		}
	}

	if (Indice == indice){
		filling = true;
	}
	return filling;
}
```

```
////Funcion de algoritmo para verificar la conectividad de cada pieza #D por medio
de flood fill
void Rompecabezas::Filling3D(Cubo &CajaDePolicubos, int &Indice, int i, int j, int
k, bool target_color, bool replacement_color, char cara){
	if (i >= 0 && j >= 0 && k >= 0 && i <= Longitud - 1 && j <= Longitud - 1 && k
<= Longitud - 1){
		if (target_color == replacement_color)
			return;
		if (CajaDePolicubos.cubo[i][j][k].GetStatus() != target_color)
			return;
			Indice++;
		if (cara == 'X'){//Aplica la funcion a la cara x con 'X'
			Filling3D(CajaDePolicubos, Indice, i - 1, j, k, target_color,
replacement_color, cara);
		}
		if (cara == 'x'){//Aplica la funcion a la cara menos x con 'x'
			Filling3D(CajaDePolicubos, Indice, i + 1, j, k, target_color,
replacement_color, cara);
		}
		if (cara == 'Y'){//Aplica la funcion a la cara y con 'Y'
			Filling3D(CajaDePolicubos, Indice, i, j - 1, k, target_color,
replacement_color, cara);
		}
		if (cara == 'y'){//Aplica la funcion a la cara menos y con 'y'
			Filling3D(CajaDePolicubos, Indice, i, j + 1, k, target_color,
replacement_color, cara);
		}
		if (cara == 'Z'){//Aplica la funcion a la cara z con 'Z'
			Filling3D(CajaDePolicubos, Indice, i, j, k - 1, target_color,
replacement_color, cara);
		}
		if (cara == 'z'){//Aplica la funcion a la cara  menos z con 'z'
			Filling3D(CajaDePolicubos, Indice, i, j, k + 1, target_color,
replacement_color, cara);
		}
	}
	else{
		return;
		}

}
//Procesa para la comstruccion de torres
bool Rompecabezas::ProcesaConstruccionDeTorres(Cubo &CajaDePolicubos, int &Indice,
int altura, int i, int j, int k, int imascara, bool mascara[], bool target_color,
bool replacement_color, char cara){

	//Pone la vandera de prueba de conectividad en falso
	bool filling = false;
	//Coordenadas de mapeo de R1 a R2
	int Coordenadas[2];
	Indice = 0;
	for (int I = 1; I <= (int)pow(Longitud, 2); I++){
		if (mascara[I] == true){
			//Resetea el valor del indice
			MapeaDeR1aR2(I, Coordenadas);
			if (cara == 'X'){
				ConstruccionTorres(CajaDePolicubos, Indice, altura, i,
Coordenadas[0], Coordenadas[1], target_color, replacement_color, cara);
```

```
                        }
                        if (cara == 'x'){
                                ConstruccionTorres(CajaDePolicubos, Indice, altura, i,
Coordenadas[0], Coordenadas[1], target_color, replacement_color, cara);
                        }
                        if (cara == 'Y'){
                                ConstruccionTorres(CajaDePolicubos, Indice, altura,
Coordenadas[0], j, Coordenadas[1], target_color, replacement_color, cara);
                        }
                        if (cara == 'y'){
                                ConstruccionTorres(CajaDePolicubos, Indice, altura,
Coordenadas[0], j, Coordenadas[1], target_color, replacement_color, cara);
                        }
                        if (cara == 'Z'){
                                ConstruccionTorres(CajaDePolicubos, Indice, altura,
Coordenadas[0], Coordenadas[1], k, target_color, replacement_color, cara);
                        }
                        if (cara == 'z'){
                                ConstruccionTorres(CajaDePolicubos, Indice, altura,
Coordenadas[0], Coordenadas[1], k, target_color, replacement_color, cara);
                        }

                }
        }
        ////////////////////////////////////// controlar construccion de torres y
prueba de filling 3D//////////////////////////////////
        if (Indice > 0){
                filling = true;
        }
        return filling;
}
//Funcion de algoritmo para generar las torres en x
void Rompecabezas::ConstruccionTorres(Cubo &CajaDePolicubos, int &Indice, int
altura,  int i, int j, int k, bool target_color, bool replacement_color, char cara){
        if (i >= 0 && j >= 0 && k >= 0 && i <= Longitud - 1 && j <= Longitud - 1 && k
<= Longitud - 1){
                if (target_color == replacement_color)
                        return;
                if (cubo->cubo[i][j][k].GetStatus() != target_color)
                        return;
                if (altura < Longitud){
                        CajaDePolicubos.cubo[i][j][k] = cubo->cubo[i][j][k];
                        cubo->cubo[i][j][k].SetStatus(replacement_color);
                        Indice++;
                        IndiceVolumen++;
                        altura++;
                        if (cara == 'X'){//Aplica la funcion a la cara x con 'X'
                                ConstruccionTorres(CajaDePolicubos, Indice, altura, i -
1, j, k, target_color, replacement_color, cara);
                        }
                        if (cara == 'x'){//Aplica la funcion a la cara menos x con 'x'
                                ConstruccionTorres(CajaDePolicubos, Indice, altura, i +
1, j, k, target_color, replacement_color, cara);
                        }
                        if (cara == 'Y'){//Aplica la funcion a la cara y con 'Y'
                                ConstruccionTorres(CajaDePolicubos, Indice, altura, i, j
- 1, k, target_color, replacement_color, cara);
                        }
```

```
                    if (cara == 'y'){//Aplica la funcion a la cara menos y con 'y'
                          ConstruccionTorres(CajaDePolicubos, Indice, altura, i, j
+ 1, k, target_color, replacement_color, cara);
                    }
                    if (cara == 'Z'){//Aplica la funcion a la cara z con 'Z'
                          ConstruccionTorres(CajaDePolicubos, Indice, altura, i, j,
k - 1, target_color, replacement_color, cara);
                    }
                    if (cara == 'z'){//Aplica la funcion a la cara  menos z con 'z'
                          ConstruccionTorres(CajaDePolicubos, Indice, altura, i, j,
k + 1, target_color, replacement_color, cara);
                    }
             }
       }
       else{
             return;
       }
}
////Funcion de algoritmo para verificar la conectividad de cada pieza en 2D por
medio de flood fill
void Rompecabezas::Filling2D(Cubo &CajaDePolicubos, int &Indice, int i, int j, int
k, bool target_color, bool replacement_color, char cara){

       if (i >= 0 && j >= 0 && k >= 0 && i <= Longitud - 1 && j <= Longitud - 1 && k
<= Longitud - 1){
             if (target_color == replacement_color)
                    return;
             if (CajaDePolicubos.cubo[i][j][k].GetStatus() != target_color)
                    return;
             CajaDePolicubos.cubo[i][j][k].SetStatus(replacement_color);
             Indice++;

             if (cara == 'X' || cara == 'x'){//Aplica la funcion a la cara x con 'X'
                    Filling2D(CajaDePolicubos, Indice, i, j, k - 1, target_color,
replacement_color, cara);
                    Filling2D(CajaDePolicubos, Indice, i, j, k + 1, target_color,
replacement_color, cara);
                    Filling2D(CajaDePolicubos, Indice, i, j + 1, k, target_color,
replacement_color, cara);
                    Filling2D(CajaDePolicubos, Indice, i, j - 1, k, target_color,
replacement_color, cara);
             }
             if (cara == 'Y' || cara == 'y'){//Aplica la funcion a la cara y con 'Y'
                    Filling2D(CajaDePolicubos, Indice, i - 1, j, k, target_color,
replacement_color, cara);
                    Filling2D(CajaDePolicubos, Indice, i + 1, j, k, target_color,
replacement_color, cara);
                    Filling2D(CajaDePolicubos, Indice, i, j, k + 1, target_color,
replacement_color, cara);
                    Filling2D(CajaDePolicubos, Indice, i, j, k - 1, target_color,
replacement_color, cara);
             }
             if (cara == 'Z' || cara == 'z'){//Aplica la funcion a la cara z con 'Z'
                    Filling2D(CajaDePolicubos, Indice, i - 1, j, k, target_color,
replacement_color, cara);
                    Filling2D(CajaDePolicubos, Indice, i + 1, j, k, target_color,
replacement_color, cara);
```

```
                    Filling2D(CajaDePolicubos, Indice, i, j + 1, k, target_color,
replacement_color, cara);
                    Filling2D(CajaDePolicubos, Indice, i, j - 1, k, target_color,
replacement_color, cara);
             }
      }
      else{
             return;
      }
}
//Procesa cada piezas para la verificacion de 2D
bool Rompecabezas::ProcesaFilling2D(Cubo &CajaDePolicubos, int &Indice, int i, int
j, int k, int imascara, bool mascara[], bool target_color, bool replacement_color,
char cara){
      //mascara de bits
      bitset<170> imascaraBits(imascara);

      //pone las banderas en falso
      for (int I = 1; I <= Longitud * Longitud; I++){
             mascarax[I] = false; mascaraMenosx[I] = false;
             mascaray[I] = false; mascaraMenosy[I] = false;
             mascaraz[I] = false; mascaraMenosz[I] = false;
      }
      //Pone la vandera de prueba de conectividad en falso
      bool filling = false;
      //Coordenadas de mapeo de R1 a R2
      int Coordenadas[2];
      //Cambia el valor de la mascara a utilizar
      if (cara == 'x' || cara == 'y' || cara == 'z'){
             for (int J = 0; J < (int)pow(Longitud, 2); J++){
                    bitset<1> imascaraBitsaux = imascaraBits[J] & 1;
                    if (imascaraBitsaux == 1)
                           mascara[(int)pow(Longitud, 2) - j] = true;
             }
      }
      else{
             for (int J = 0; J < (int)pow(Longitud, 2); J++){
                    bitset<1> imascaraBitsaux = imascaraBits[J] & 1;
                    if (imascaraBitsaux == 1)
                           mascara[J + 1] = true;
             }
      }
      //Verifica si la bandera esta en verdadero para utilización de la misma
      for (int I = 1; I <= Longitud * Longitud; I++){
             if (mascara[I] == true){
                    //Pone las coordenadas de la mascara de a las de la pieza
                    MapeaDeR1aR2(I, Coordenadas);
                    if (cara == 'X' || cara == 'x'){

      CajaDePolicubos.cubo[i][Coordenadas[0]][Coordenadas[1]].SetStatus(target_colo
r);
                    }
                    if (cara == 'Y' || cara == 'y'){

      CajaDePolicubos.cubo[Coordenadas[0]][j][Coordenadas[1]].SetStatus(target_colo
r);
                    }
                    if (cara == 'Z' || cara == 'z'){
```

```
	CajaDePolicubos.cubo[Coordenadas[0]][Coordenadas[1]][k].SetStatus(target_colo
r);
				}
			}
		}
		//prueba de conectividad
		for (int I = 1; I <= (int)pow(Longitud,2); I++){
			if (mascara[I] == true){
				//Resetea el valor del indice
				Indice = 0;
				MapeaDeR1aR2(I, Coordenadas);
				if (cara == 'X' || cara == 'x'){
					Filling2D(CajaDePolicubos, Indice, i, Coordenadas[0],
Coordenadas[1], target_color, replacement_color, cara);
				}
				if (cara == 'Y' || cara == 'y'){
					Filling2D(CajaDePolicubos, Indice, Coordenadas[0], j,
Coordenadas[1], target_color, replacement_color, cara);
				}
				if (cara == 'Z' || cara == 'z'){
					Filling2D(CajaDePolicubos, Indice, Coordenadas[0],
Coordenadas[1], k, target_color, replacement_color, cara);
				}
				break;
			}
		}

		// si hay conectivadad saca del ciclo
		int count = 0;
		for (int I = 1; I <= Longitud * Longitud; I++){
			if (mascara[I] == true)
				count++;
		}
		if (Indice == count){
			filling = true;
		}
		return filling;
}
//Pocesa los datos de las piezas para ver los resultados obtenidos
void Rompecabezas::ProcesarDatos(Cubo CajaDePolicubos, int Indice, char Cara[]){
		int i, j, k, iPivote = 1;
		int iaux = 0, jaux = 0, kaux = 0;
		int m = 0;
		ofstream Archivo("Resultado/Resultado.txt", ios::app);
		Archivo <<
"////////////////////////////////////////////////////////////////////////////////
////////////////////////////////\n";
		for (i = 0; i <= Longitud; i++){
			for (j = 0; j <= Longitud; j++){
				for (k = 0; k <= Longitud; k++){
					if (CajaDePolicubos.cubo[i][j][k].GetStatus() == true){
						iPivote =
CajaDePolicubos.cubo[i][j][k].GetColor();
						CajaDePolicubos.cubo[i][j][k].GetPosicion(iaux,
jaux, kaux);
						CajaDePolicubos.cubo[i][j][k].GetStatus();
						m += 1;
```

```
                                cout << Cara << ": (color,i,j,k,status)     " <<
iPivote << " " << iaux << " " << jaux << " " << kaux << " " <<
CajaDePolicubos.cubo[i][j][k].GetStatus() << "\n";
                                Archivo << Cara << ": (color,i,j,k,status)     "
<< iPivote << " " << iaux << " " << jaux << " " << kaux << " " <<
CajaDePolicubos.cubo[i][j][k].GetStatus() << "\n";
                        }
                        else{}
                }
        }
    }
    Archivo.close();
}
//Pocesa los datos de las piezas para graficarlas
void Rompecabezas::ProcesarGrafica(Cubo CajaDePolicubos, int Indice, GLfloat
color[]){
    int i, j, k, iPivote = 1;
    //Grafica de la cara x
    glPushMatrix();
    for (i = 0; i <= Longitud; i++){
        for (j = 0; j <= Longitud; j++){
            for (k = 0; k <= Longitud; k++){
                if (CajaDePolicubos.cubo[i][j][k].GetStatus() == true){
                    iPivote =
CajaDePolicubos.cubo[i][j][k].GetColor();
                    if ((iPivote % 2) == 0)
                        glColor3f(0.0f, 0.0f, 0.0f);//color negro
                    else
                        glColor3fv(color);//arreglo de colores
                    CajaDePolicubos.cubo[i][j][k].GrafPolicubo();
                }
                else{}
            }
        }
    }
    glPopMatrix();
}

//Grafica las piezas obtenidas
void Rompecabezas::GrafCaraX(){
    ProcesarGrafica(CajaDePolicubosX, Indicex,colorx);
}
void Rompecabezas::GrafCaraMenosX(){
    ProcesarGrafica(CajaDePolicubosMenosX, IndiceMenosx,colorMenosx);
}
void Rompecabezas::GrafCaraY(){
    ProcesarGrafica(CajaDePolicubosY, Indicey,colory);
}
void Rompecabezas::GrafCaraMenosY(){
    ProcesarGrafica(CajaDePolicubosMenosY, IndiceMenosy,colorMenosy);
}
void Rompecabezas::GrafCaraZ(){
    ProcesarGrafica(CajaDePolicubosZ, Indicez,colorz);
}
void Rompecabezas::GrafCaraMenosZ(){
    ProcesarGrafica(CajaDePolicubosMenosZ, IndiceMenosz,colorMenosz);
}
//muestra los resultados de la creacion del cubo logico
```

```
void Rompecabezas::ResultRompecabezas(){
	ofstream Archivo("Resultado/Resultado.txt", ios::app);
	//Imprime los datos de las piezas
	cout << "Resultados creacion de las piezas: \n";
	Archivo <<
"////////////////////////////////////////////////////////////////////////////////
////////////////////////////////\n";
	Archivo << "Resultados creacion de las piezas: \n";
	ProcesarDatos(CajaDePolicubosX, Indicex, "Cara x");
	ProcesarDatos(CajaDePolicubosMenosX, IndiceMenosx, "Cara Menosx");
	ProcesarDatos(CajaDePolicubosY, Indicey, "Cara y");
	ProcesarDatos(CajaDePolicubosMenosY, IndiceMenosy, "Cara Menosy");
	ProcesarDatos(CajaDePolicubosZ, Indicez, "Cara z");
	ProcesarDatos(CajaDePolicubosMenosZ, IndiceMenosz, "Cara Menosz");
	//Imprime los indices para la veridicacion
	Archivo <<
"////////////////////////////////////////////////////////////////////////////////
////////////////////////////////\n";
	cout << "Valores de los indices: \n";
	Archivo << "Valores de los indices: \n";
	cout << "Indicex:" << Indicex << "\n";
	Archivo << "Indicex:" << Indicex << "\n";
	cout << "IndiceMenosx:" << IndiceMenosx << "\n";
	Archivo << "IndiceMenosx:" << IndiceMenosx << "\n";
	cout << "Indicey:" << Indicey << "\n";
	Archivo << "Indicey:" << Indicey << "\n";
	cout << "IndiceMenosy:" << IndiceMenosy << "\n";
	Archivo << "IndiceMenosy:" << IndiceMenosy << "\n";
	cout << "Indicez:" << Indicez << "\n";
	Archivo << "Indicez:" << Indicez << "\n";
	cout << "IndiceMenosz:" << IndiceMenosz << "\n";
	Archivo << "IndiceMenosz:" << IndiceMenosz << "\n";
	cout << "IndiceVolumen:" << IndiceVolumen << "\n";
	Archivo << "IndiceVolumen:" << IndiceVolumen << "\n";
	Archivo.close();

}
```

11.3.5. Código función AsignacionVariables

```
#include"stdafx.h"

//Variables globales para utilizacion y modificaciones especiales
GLdouble    ex, ey, ez, cx, cy, cz, ux, uy, uz, delta, deltaR;
GLfloat *Color;
GLboolean    bCull, bDepth, bOutline, bShade, bLight;
GLenum shademode;
GLfloat desp;
//Funcion para asignación de valores para las teclas especiales
void TeclasEspeciales(GLdouble CX, GLdouble CY, GLdouble CZ, GLdouble EX, GLdouble
EY, GLdouble EZ, GLdouble UX, GLdouble UY, GLdouble UZ,
	GLdouble DELTA_R, GLdouble DELTA, GLboolean bCULL, GLboolean bDEPTH,
GLboolean bOUTLINE, GLenum bSHADE, GLboolean bLIGHT, GLfloat *COLOR
	,GLfloat Desp){
	//Efectos de Gaficos
```

```
	bCull = bCULL; bDepth = bDEPTH; bOutline = bOUTLINE; shademode = bSHADE;
Color = COLOR;
	//Asignacion de valores predefinidos para incrementos de camara
	ex = EX; ey = EY; ez = EZ; cx = CX; cy = CY; cz = CZ; ux = UX; uy = UY; uz =
UZ; delta = DELTA; deltaR = DELTA_R; bLight = bLIGHT;
	desp = Desp;

}
```

11.3.6. Código función ChangeSize

```
#include"stdafx.h"

void ChangeSize(GLsizei w, GLsizei h){
	// Set Viewport to window dimensions
	glViewport(0, 0, (GLsizei)w, (GLsizei)h);

	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluPerspective(60.0, 1.0, 0.1, 500.0);
	glMatrixMode(GL_MODELVIEW);
}
```

11.3.7. Código función Display

```
#include"stdafx.h"
//#include"Policubo.h"
//objeto cubo
Cubo DesplegarCubo;
Rompecabezas DesplegarDescCubo;
//Asigan el parametro de la funcion principal y lo asigna a DesplegarCubo para
graficar el cubo
void RenderizarPolicubo(Cubo Cubo){
	DesplegarCubo = Cubo;
}
//Asigan el parametro de la funcion principal y lo asigna a DesplegarDescCubo para
graficar la descomposicion del cubo
void RenderizarDescCubo(Rompecabezas piezas){
	DesplegarDescCubo = piezas;
}
//Funcion para renderizacion de imagen
void Display(void)
{
	glLoadIdentity();//Carga la matriz identidad
	gluLookAt((GLdouble)ex, (GLdouble)ey, (GLdouble)ez, (GLdouble)cx,
(GLdouble)cy, (GLdouble)cz, (GLdouble)ux, (GLdouble)uy, (GLdouble)uz);// Uso de
camara

	// Limpia la vewntana y el bufer de profundidad
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	//ACtiva el  para la parte frontal y treasera
	glCullFace(GL_BACK);
	//Activa las propiedades para depth test
	glDepthFunc(GL_LEQUAL);
	//Sirve para utilizar las teclas especiales para los efectos////////////////
	// Cambia culling a encendido so la bandera esta activa
```

```
        if (bCull)   glEnable(GL_CULL_FACE);
        else        glDisable(GL_CULL_FACE);

        // Activa prueba de profundidad si la bandera esta activa
        if (bDepth)  glDisable(GL_DEPTH_TEST);
        else        glEnable(GL_DEPTH_TEST);

        // Dibuja la parte de atras como malla, si la bandera esta activa
        if (bOutline)glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
        else        glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
        //Enciende y apaga la luz
        if (bLight)glEnable(GL_LIGHTING);
        else        glDisable(GL_LIGHTING);
        ////////////////////////////////////////////////////////////////////////
        // Guarda el estado de la matriz
        glPushMatrix();
        DesplegarCubo.GrafCubo();
        // Restaura la transformacion
        glPopMatrix();
        glPushMatrix();
        glTranslatef(desp, 0, 0);
        DesplegarDescCubo.GrafCaraX();
        glPopMatrix();
        glPushMatrix();
        glTranslatef(-desp, 0, 0);
        DesplegarDescCubo.GrafCaraMenosX();
        glPopMatrix();
        glPushMatrix();
        glTranslatef(0, desp, 0);
        DesplegarDescCubo.GrafCaraY();
        glPopMatrix();
        glPushMatrix();
        glTranslatef(0, -desp, 0);
        DesplegarDescCubo.GrafCaraMenosY();
        glPopMatrix();
        glPushMatrix();
        glTranslatef(0, 0, desp);
        DesplegarDescCubo.GrafCaraZ();
        glPopMatrix();
        glPushMatrix();
        glTranslatef(0, 0, -desp);
        DesplegarDescCubo.GrafCaraMenosZ();
        glPopMatrix();
        // Restaura la transformacion
        glPopMatrix();

        // Limpia y dibuja los comandos
        glutSwapBuffers();
}
```

11.3.8. Código función Keyboard

```
#include"stdafx.h"

void keyboard(unsigned char key, int x, int y){
```

```
	switch (key) {
	case 27:  /*  Tecla de Escape para salir de la aplicación */
		exit(0);
		break;
	case'W'://reduce la distancia de la camara al origen (close up)
		if (ex <= 5 && ey <= 5 && ez <= 5){
			ex = ex; ey = ey; ez = ez;
		}
		else{
			ex *= (1.0f - deltaR); ey *= (1.0f - deltaR);   ez *= (1.0f -
deltaR);
		}
		glutPostRedisplay(); return;
		break;
	case'w'://reduce la distancia de la camara al origen (close up)
		if (ex <= 5 && ey <= 5 && ez <= 5){
			ex = ex; ey = ey; ez = ez;
		}
		else{
			ex *= (1.0f - deltaR); ey *= (1.0f - deltaR);   ez *= (1.0f -
deltaR);
		}
		glutPostRedisplay(); return;
		break;
	case'S'://incrementa la distacncia de la camara al origen
		if (ex >= 30 || ey >= 30 || ez >= 30){
			ex = ex; ey = ey; ez = ez;
		}
		else{
			ex *= (1.0f + deltaR);  ey *= (1.0f + deltaR);  ez *= (1.0f +
deltaR);
		}
		glutPostRedisplay(); return;
		break;
	case's'://incrementa la distacncia de la camara al origen
		if (ex >= 30 || ey >= 30 || ez >= 30){
			ex = ex; ey = ey; ez = ez;
		}
		else{
			ex *= (1.0f + deltaR);  ey *= (1.0f + deltaR);  ez *= (1.0f +
deltaR);
		}
		glutPostRedisplay(); return;
		break;
	case'1'://cambia la posicion del vector de up en la camara de a x
		ux = 1; uy = 0; uz = 0;
		glutPostRedisplay(); return;
		break;
	case'2'://cambia la posicion del vector de up en la camara de a y
		ux = 0; uy = 1; uz = 0;
		glutPostRedisplay(); return;
		break;
	case'3'://cambia la posicion del vector de up en la camara de a z
		ux = 0; uy = 0; uz = 1;
		glutPostRedisplay(); return;
		break;
	case'4'://cambia el color del cubo a rojo
		Color[0] = 1.0;
```

```
		Color[1] = 0.0;
		Color[2] = 0.0;
		glutPostRedisplay(); return;
		break;
	case'5'://cambia el color del cubo a verde
		Color[0] = 0.0;
		Color[1] = 1.0;
		Color[2] = 0.0;
		glutPostRedisplay(); return;
		break;
	case'6'://cambia el color del cubo a azul
		Color[0] = 0.0;
		Color[1] = 0.0;
		Color[2] = 1.0;
		glutPostRedisplay(); return;
		break;
	case'7'://cambia el color del cubo a Cyan
		Color[0] = 0.0;
		Color[1] = 1.0;
		Color[2] = 1.0;
		glutPostRedisplay(); return;
		break;
	case'8'://cambia el color del cubo a Magenta
		Color[0] = 1.0;
		Color[1] = 0.0;
		Color[2] = 1.0;
		glutPostRedisplay(); return;
		break;
	case'9'://cambia el color del cubo a Amarillo
		Color[0] = 1.0;
		Color[1] = 1.0;
		Color[2] = 0.0;
		glutPostRedisplay(); return;
		break;
	case'0'://cambia el color del cubo a Blanco
		Color[0] = 1.0;
		Color[1] = 1.0;
		Color[2] = 1.0;
		glutPostRedisplay(); return;
		break;
	case'Q'://Desplazar piezas hacia afuera
		if (desp == 20)
			desp = desp;
		else
			desp++;
		glutPostRedisplay(); return;
		break;
	case'q'://Desplazar piezas hacia afuera
		if (desp == 20)
			desp = desp;
		else
			desp++;
		glutPostRedisplay(); return;
		break;
	case'A'://Desplazar piezas hacia el centro
		if (desp == 0)
			desp = desp;
		else
```

```
			desp--;
			glutPostRedisplay(); return;
			break;
		case'a'://Desplazar piezas hacia el centro
			if (desp == 0)
				desp = desp;
			else
				desp--;
			glutPostRedisplay(); return;
			break;
		default:
			break;
		}
}
```

11.3.9. Código función SetupRC

```
#include"stdafx.h"

GLfloat  ambientLight[] = { 0.3f, 0.3f, 0.3f, 1.0f }, diffuseLight[] = { 0.7f, 0.7f,
0.7f, 1.0f },
specular[] = { 1.0f, 1.0f, 1.0f, 1.0f }, specref[] = { 1.0f, 1.0f, 1.0f, 1.0f };

void SetupRC()
{
	// Blue background
	glClearColor(0.3f, 0.3f, 0.3f, 1.0f);

	// Set color shading model to flat
	glShadeModel(shademode);
	//da la iluminacion al entorno
	glLightfv(GL_LIGHT0, GL_AMBIENT, ambientLight);
	glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuseLight);
	glLightfv(GL_LIGHT0, GL_SPECULAR, specular);
	glEnable(GL_LIGHT0);
	glEnable(GL_COLOR_MATERIAL);
	// Clockwise-wound polygons are front facing; this is reversed
	// because we are using triangle fans
	glFrontFace(GL_CCW);
}
```

11.3.10. Código función SpecialKeys

```
#include"stdafx.h"
//Funcion de teclas especiales
void SpecialKeys(int key, int x, int y){

	GLdouble dx, dy, dz, dx0, dy0, dz0;
```

```
if (key == GLUT_KEY_UP){
       //Rota la camara alrededor del origen  en el plano Oyz  tecla arriba
       dy0 = cz - ez;  dz0 = ey - cy;
       dy = cy - ey;    dz = cz - ez;
       GLdouble s = sqrtl(dy*dy + dz*dz);
       dy += delta*dy0;       dz += delta*dz0;
       GLdouble s1 = sqrtl(dy*dy + dz*dz) / s;
       dy /= s1;           dz /= s1;
       ey = cy - dy; ez = cz - dz;

       glutPostRedisplay(); return;
}
if (key == GLUT_KEY_DOWN){
       //Rota la camara alrededor del origen  en el plano Oyz  tecla abajo
       dy0 = cz - ez;  dz0 = ey - cy;
       dy = cy - ey;    dz = cz - ez;
       GLdouble s = sqrtl(dy*dy + dz*dz);
       dy -= delta*dy0;       dz -= delta*dz0;
       GLdouble s1 = sqrtl(dy*dy + dz*dz) / s;
       dy /= s1;           dz /= s1;
       ey = cy - dy; ez = cz - dz;

       glutPostRedisplay(); return;
}
if (key == GLUT_KEY_LEFT){
       //Rota la camara alrededor del origen  en el plano Oxz  tecla izquierda
       dx0 = cz - ez;  dz0 = ex - cx;
       dx = cx - ex;  dz = cz - ez;
       GLdouble s = sqrtl(dx*dx + dz*dz);
       dx -= delta*dx0;       dz -= delta*dz0;
       GLdouble s1 = sqrtl(dx*dx + dz*dz) / s;
       dx /= s1;           dz /= s1;
       ex = cx - dx; ez = cz - dz;

       glutPostRedisplay(); return;
}
if (key == GLUT_KEY_RIGHT){
       //Rota la camara alrededor del origen  en el plano Oxz  tecla derecha
       dx0 = cz - ez;  dz0 = ex - cx;
       dx = cx - ex;  dz = cz - ez;
       GLdouble s = sqrtl(dx*dx + dz*dz);
       dx += delta*dx0;       dz += delta*dz0;
       GLdouble s1 = sqrtl(dx*dx + dz*dz) / s;
       dx /= s1;           dz /= s1;
       ex = cx - dx; ez = cz - dz;

       glutPostRedisplay(); return;
}
if (key == GLUT_KEY_F1){
       bCull = !bCull; glutPostRedisplay();
}
if (key == GLUT_KEY_F2){
       bDepth = !bDepth; glutPostRedisplay();
}
if (key == GLUT_KEY_F3){
       bOutline = !bOutline; glutPostRedisplay();
}
if (key == GLUT_KEY_F4){
```

```
            if (shademode == GL_FLAT){
                  shademode = GL_SMOOTH;
            }
            else{
                  if (shademode == GL_SMOOTH){
                        shademode = GL_FLAT;
                  }
            };
            glShadeModel(shademode);
            glutPostRedisplay();
      }
      if (key == GLUT_KEY_F5){
            bLight = !bLight; glutPostRedisplay();
      }
      if (key == GLUT_KEY_F6){
            ex = -ex; ey = ey; ez = ey; glutPostRedisplay();
      }
      if (key == GLUT_KEY_F7){
            ex = ex; ey = -ey; ez = ez; glutPostRedisplay();
      }
      if (key == GLUT_KEY_F8){
            ex = ez; ey = ey; ez = -ez; glutPostRedisplay();
      }
      if (key == GLUT_KEY_F9){
            ex = -ez; ey = -ey; ez = -ez; glutPostRedisplay();
      }
      // Refresca la ventana
      glutPostRedisplay();
}
```

12. Entregables comprometidos en la propuesta

Se entregarán un disco compacto al Coordinador de Estudios de Ingeniería en Computación que incluirán el reporte final del proyecto terminal en un archivo PDF (sin restricciones) y el código fuente de la aplicación en un archivo comprimido (sin restricciones). El reporte final contendrá: portada, resumen, tabla de contenido, objetivos, introducción, desarrollo del proyecto, conclusiones, bibliografía, apéndices y manual de usuario. Los apéndices contendrán al menos un listado del código fuente desarrollado.