Simulación de mecanizado incremental adaptativa

Rafael Molina Carmona, Ramón Rizo Aldeguer, Juan Antonio Puchol García

Grupo de Informática Industrial e Inteligencia Artificial (i3a)
Universidad de Alicante
Campus de San Vicente. Apdo. Correos 99, 03080, Alicante
Tlf: 965 90 39 00 Fax: 965 90 39 02 e-mail: {rmolina, rizo, puchol}@dccia.ua.es

Resumen

La simulación de los procesos de mecanizado tiene un gran interés en el ámbito de la fabricación por ordenador. Presentamos un método de poligonalización para objetos generados por desbaste basado en una partición espacial en vóxeles, adaptando a este problema los algoritmos Marching Cubes y Postproceso de Rectificación de Metapolígonos. Sus características permiten aprovechar la coherencia espacial procesando sólo los vóxeles en un entorno de vecindad. Además, se trata de un método incremental, es decir, los cálculos realizados se reutilizan en pasos sucesivos. También se han introducido propiedades que lo hacen adaptativo, de tal manera que la densidad de la malla se adapta localmente a la superficie. Con estas características comprobamos que los tiempos de proceso se reducen drásticamente si los comparamos con otros métodos tradicionales. Esto permite realizar una visualización en tiempo real, especialmente adecuada para la simulación del mecanizado de piezas.

Palabras clave: simulación, poligonalización, Marching Cubes.

1 Introducción

En los procesos de diseño y fabricación por ordenador, cada vez es más importante contar con una representación visual lo más cercana posible a la realidad. En particular, visualizar el proceso de mecanizado de una pieza por control numérico es imprescindible para detectar errores que no es posible determinar con la representación matemática. Además, esta visualización tiene otras ventajas: ayuda a comprender el modelo y facilita procesos como la optimización de trayectorias.

Visualizar un objeto obtenido por desbaste de material tiene la particularidad de que no es posible basarse en las superficies que forman el objeto sino en las trayectorias de mecanizado, con lo que no se puede realizar un mallado directo. Además, cada vez más se demandan representaciones en tiempo real, que permitan visualizar el proceso completo de mecanizado y no sólo el resultado final.

Para este proceso encontramos tres aproximaciones principales en la literatura: visualización directa, utilización de geometría sólida constructiva (CSG) y uso de métodos de partición espacial [1].

La visualización directa (por ejemplo, mediante trazado de rayos) puede proporcionar imágenes de gran realismo, pero tiene un alto coste computacional. Además, un cambio en la posición del observador conlleva el recálculo de todos los píxeles de la escena.

El desbaste de material puede ser interpretado como un conjunto de operaciones booleanas regularizadas, por lo que el modelo CSG puede adaptarse convenientemente. Sin embargo, en la práctica, su aplicación se encuentra muy limitada puesto que la formulación de los volúmenes de barrido no suele ser sencilla, especialmente si tratamos con máquinas de más de dos ejes.

Los métodos de partición espacial se basan en la descomposición del espacio en un conjunto de elementos geométricos sencillos que facilitan la realización de las operaciones booleanas. Las distintas aproximaciones utilizan estructuras de tipo z-buffer, árboles octales o vóxeles [1]. En algunos casos se opta por representar directamente la partición espacial y en otros por obtener, a partir de ésta, un modelo secundario poligonal para representar el objeto. Esto proporciona algunas ventajas: el modelo poligonal produce una visualización de gran calidad, su representación es rápida y es independiente de la vista. A cambio, la obtención de la malla a partir de las trayectorias de mecanizado no resulta una tarea sencilla.

En el presente trabajo hemos optado por desarrollar un método de obtención de un modelo secundario poligonal a partir de una partición espacial en vóxeles. Para ello se ha partido de los algoritmos Marching Cubes [2][3] y Postproceso de Rectificación de Metapolígonos (PRM) [4][5], a los que se le han incorporado nuevas características de incrementalidad y de adaptación local. Así podemos aprovechar los cálculos de un paso de la simulación en el siguiente, obtener una malla menos densa pero más aproximada y realizar la visualización en tiempo real.

En el apartado 2 del documento se presentan los algoritmos Marching Cubes y PRM en su versión original. En el apartado 3 se propone una extensión del método para emplearlo en la simulación de procesos de mecanizado. Las pruebas realizadas y los resultados obtenidos sobre distintos modelos centran el contenido del apartado 4. Por último, en el apartado 5 se presentan las conclusiones.

2 Los algoritmos Marching Cubes y PRM

2.1 El algoritmo Marching Cubes

El algoritmo Marching Cubes aparece descrito en los artículos de Wyvill y McPheeters [2] y de Lorensen y Cline [3]. Se trata de un algoritmo diseñado para obtener una malla poligonal a partir de un volumen en el que no existe información superficial. Wyvill y McPheeters lo utilizan para visualizar una isosuperficie, mientras que Lorensen y Cline muestran datos médicos tridimensionales.

La idea general del algoritmo consiste en dividir el volumen de un paralelepípedo que encierra al objeto a representar en un número de cubos de igual tamaño (la precisión de la malla final depende del tamaño del cubo). A continuación se analiza cada cubo por separado determinando qué vértices del cubo son interiores al objeto y cuáles exteriores. A partir de la información sobre la colocación del cubo con respecto al objeto, se procede a estudiar qué forma tendrá la porción de superficie asociada a dicho cubo. De las 256 posibles maneras en que un cubo puede intersectar con una superficie, son sólo 16 los casos distintos, si eliminamos los casos imposibles, las simetrías y las rotaciones (puede consultarse la disposición de los 16 casos posibles en [3]). La unión de los distintos triángulos obtenidos forma la malla completa el objeto.

El algoritmo resultante es sencillo de implementar y tiene, relativamente, un bajo coste computacional. Además, da lugar a una malla igualmente sencilla. También tiene, no obstante algunos inconvenientes: no se puede mejorar la precisión localmente, puede provocar huecos en la malla y, en ocasiones, no preserva la conectividad de los objetos. Varios autores han presentado mejoras sobre el método inicial para solucionar estos problemas [4][5][6][7][8].

2.2 El algoritmo PRM

El algoritmo de Postproceso de Rectificación de Metapolígonos (PRM) se utilizó en primer lugar para adaptar una malla poligonal inicial de densidad fija a una isosuperficie de tipo blob [4]. La triangulación inicial se realizaba mediante el algoritmo Marching Cubes [3] o sobre variantes más avanzadas del mismo [7].

Posteriormente el método se ha adaptado para superficies libres [5]. En este caso el algoritmo cuenta con su propia poligonalización inicial, por lo que se trata más de un algoritmo completo que de un postproceso. Las características de las superficies libres permiten adaptar los nuevos vértices de una forma más rápida y precisa que la utilizada en el algoritmo original con blobs. En la adaptación a

superficies libres se introducen nuevos casos de partición triangular para optimizar la malla y se trata su aplicación sobre espacios discretos, generalizando el proceso.

El PRM trata de adaptar una malla poligonal inicial a la superficie (blob o libre) mediante la división recursiva de cada polígono de la malla. Para decidir si dividimos o no un triángulo (base de recursión) definimos un criterio de partición del triángulo: para cada una de sus aristas se calcula la longitud y el ángulo que forman entre sí los vectores gradiente de la superficie medidos en sus extremos. Si el ángulo asociado a alguna arista supera un umbral U o su longitud supera un umbral L, el triángulo debe dividirse por el punto medio de la arista. Dependiendo de dicho criterio, dividiremos el triángulo según las aristas que fallen (figura 1).

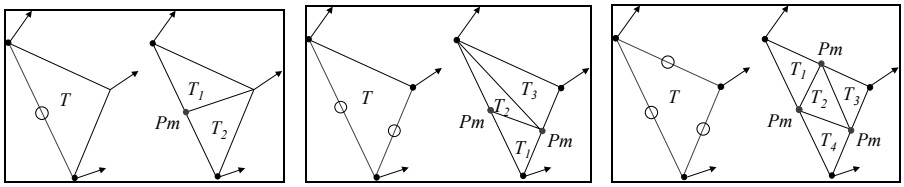


Figura 1: Casos de partición de triángulo en dos, tres y en cuatro triángulos.

A los triángulos resultantes les aplicaremos el mismo método recursivamente hasta una altura máxima H del árbol de recursión. Los nuevos vértices obtenidos de cada división han de adaptarse a la superficie mediante un método de búsqueda de un punto sobre la frontera. La malla poligonal final constará de todos los triángulos que incumplan el criterio de partición de un triángulo o que hayan llegado a la división recursiva máxima. El resultado de este algoritmo es una malla triangular adaptativa de densidad variable ajustada a partir de la malla inicial.

3 Simulación Incremental Adaptativa

El método de Simulación Incremental Adaptativa es un algoritmo de obtención del modelo secundario poligonal para visualización de objetos definidos a partir de trayectorias de mecanizado, obtenido como una variación del algoritmo PRM. Como éste, su funcionamiento parte de una poligonalización inicial del objeto según un esquema como el del algoritmo Marching Cubes, calculando la interioridad/exterioridad de los vértices de cada cubo y poligonalizándolo. Posteriormente esta malla se adapta localmente según el criterio de división de triángulos del PRM.

Nos planteamos dos aportaciones fundamentales sobre el algoritmo básico: en primer lugar el modelo poligonal se irá obteniendo de forma incremental, es decir, se obtiene un modelo completo para cada posición discreta de la herramienta y este modelo se toma como base para la siguiente posición de la herramienta. En segundo lugar, se incorpora el mecanismo de adaptación local que permite hacer más densa la malla en aquellas zonas donde es necesario, adaptando el método PRM ya referido. Estas características reducen considerablemente los tiempos de cómputo y permiten utilizarlo para simular el proceso de mecanizado en tiempo real.

3.1 Definición del problema

En primer lugar presentamos una definición del problema: Sea un objeto generado a partir de un conjunto de trayectorias de mecanizado T que desbastan un paralelepípedo de material M. Definimos este objeto como un par $O=(M,T)$.

Por simplificar y sin pérdida de generalidad, el material vendrá definido como un par de puntos en el espacio $M=(m_{min},m_{max})$, correspondientes a los puntos de coordenadas máxima y mínima que determinan el paralelepípedo.

El conjunto de trayectorias vendrá dado por un conjunto de pares $T=\{(c_1,h_1),(c_2,h_2),...,(c_n,h_n)\}$ donde cada c_i es una curva y cada h_i la herramienta que pasa por esa curva. La definición de la curva dependerá del modelo utilizado para diseñarla (una polilínea, una curva BSpline, una curva Bézier, ...), mientras que la herramienta se define como una tupla de parámetros que dependerán del tipo de herramienta. Por ejemplo, una herramienta esférica vendrá dada por la tupla $h_i=(r_i,a_i,\mathbf{v}_i)$, donde r_i es el radio de la herramienta, a_i el punto de aplicación (para una herramienta esférica coincide con el centro de la esfera) y $\mathbf{v}_i$ su vector de incidencia. En cualquier caso, supondremos que un punto de la trayectoria marca la posición del punto de aplicación de la herramienta correspondiente.

Un objeto de este tipo puede describirse utilizando el modelo CSG en el que la operación a realizar es:

$$O = M \sim \bigcup_{i=1}^{n} (c_i, h_i)$$

donde M es el paralelepípedo de material y cada (c_i,h_i) lo podemos ver como el objeto formado al desplazar la herramienta h_i a través de la curva c_i.

El problema que tratamos de resolver, dadas estas definiciones, es el de construir un modelo poligonal formado por una malla de triángulos que aproxime el objeto resultado de mecanizar el paralelepípedo de material haciendo pasar por las trayectorias las herramientas correspondientes, o lo que es lo mismo, que

aproxime el objeto CSG de la ecuación anterior. El algoritmo que construya este objeto tendrá como entrada el propio objeto y como salida una malla triangular.

3.2 Incrementalidad

La incrementalidad en el método se consigue reutilizando en cada nuevo paso de poligonalización los resultados del paso anterior, de manera que cada iteración incrementa la solución anterior. Este incrementalidad se da tanto en la configuración de los cubos como en la poligonalización.

El primer paso del algoritmo será dividir el paralelepípedo M en un conjunto disjunto de cubos o vóxeles de igual tamaño $V=\{v_j\}$, de manera que

$$\left(M = \bigcup v_j\right) \wedge \left(\left(\bigcap v_j = \phi\right)\right)$$

Como en el algoritmo original de Marching Cubes, cada vóxel se define mediante un par $v_j=(A_j,V_j)$, siendo A_j un conjunto de 12 aristas y V_j un conjunto de 8 vértices. Cada vértice es un valor binario, de manera que 0 indica que el vértice es exterior al objeto y 1 que es interior. Por su lado, cada arista almacena un valor en el intervalo [0,100], que indica el porcentaje de interioridad de la arista, es decir, aquellas aristas cuyos vértices sean interiores almacenarán el valor 100%; aquellas cuyos vértices sean exteriores tendrán un valor de 0% y aquellas cuyos vértices sean uno interior y el otro exterior, guardarán un valor intermedio que indique qué tanto por ciento de esa arista es interior al objeto.

Los vóxeles se guardan en una matriz tridimensional que se inicializa con los valores adecuados para los vértices y aristas: así, todos lo vértices interiores al paralelepípedo se etiquetan con 1, mientras que los exteriores se etiquetan con 0. También asigna valor a las aristas: 100% las interiores y 0% las exteriores.

A continuación debemos comenzar el proceso con cada una de las trayectorias, restando al paralelepípedo inicial la herramienta en la posición marcada por la curva. Es en estos pasos en los que se adapta el algoritmo para que el proceso sea incremental, es decir, la ecuación del objeto se aplica restando a cada paso al paralelepípedo la herramienta en la posición dada.

La curva c_i por donde pasa la herramienta es una curva continua, por lo que es necesaria una discretización que permita realizar el proceso en pasos discretos. Sea p_{max}, el paso máximo de trayectoria. Discretizaremos c_i en un conjunto de puntos p_k, tal que se cumpla que $\forall k, |p_k - p_{k+1}| < p_{max}$. Si p_{max} está bien escogido, evitaremos que la herramienta quede totalmente interior o exterior a uno de los cubos.

Llamemos c'_i a la discretización de la curva c_i. Para cada punto $p_k \in c'_i$, colocamos el centro de la herramienta en la posición p_k y procedemos a restar de forma booleana el volumen de la herramienta al del paralelepípedo. Para ello

comprobamos qué vértices de cada vóxel de la matriz son interiores o exteriores al volumen de la herramienta y los etiquetamos convenientemente. Evidentemente, conociendo el radio de la herramienta, podemos determinar un entorno de vecindad centrado en p_k y de radio el radio de la herramienta, de manera que sólo será necesario comprobar los vértices de aquellos vóxeles situados en ese entorno.

Una vez calculados y etiquetados los vértices interiores y exteriores, debemos asignar a cada arista el valor de corte adecuado. Se pueden dar los siguientes casos, dependiendo de los cambios que sufran los vértices:

- Si ambos vértices ya eran exteriores, no debe realizarse ninguna acción, puesto que ningún vértice puede pasar de ser exterior a ser interior.
- Si ambos vértices eran interiores pueden darse tres casos:
  - Si siguen siendo interiores, no realizamos ninguna acción adicional.
  - Si ambos pasan a ser a exteriores, tampoco debe realizarse acción alguna.
  - Si uno pasa a ser exterior, debemos intersectar la arista con el volumen de la herramienta y asignarle el porcentaje de interioridad correspondiente.
- Si un vértice era interior y otro exterior, pueden darse dos casos:
  - Si después del procedimiento ambos vértices son exteriores, no debe realizarse ninguna acción adicional.
  - Si sigue habiendo uno interior y otro exterior, debemos calcular la nueva intersección de la arista con el volumen de la herramienta y asignar a la arista el mínimo entre el porcentaje de interioridad que ya tenía asignado y el recién calculado.

Este procedimiento debe repetirse para todos los puntos de todas las trayectorias que conforman el objeto, utilizando en cada paso la configuración de vóxeles proveniente del paso anterior.

Determinar la interioridad/exterioridad de los vértices así como el punto de corte de las aristas con el volumen de la herramienta, va a depender de la geometría de ésta. En el caso de las herramientas estándar, estos cálculos son muy sencillos. Sea, por ejemplo, una herramienta esférica. Un punto será interior a la herramienta si la distancia euclídea del punto al centro de la esfera es menor que su radio. En cuanto al cálculo del punto de corte entre la herramienta y una arista recta, basta con sustituir la ecuación de la recta en la ecuación de la esfera. El caso de otro tipo de herramientas (cilíndricas, cónicas y tóricas) es igualmente sencillo.

Por último debe realizarse la triangulación conforme a los 14 casos posibles de Marching Cubes. Sin embargo, es posible aprovechar aquí también la coherencia espacial para disminuir los costes y dotar también, a este proceso, de características incrementales. Para ello proponemos una estructura de datos en la que los triángulos se encuentran asociados a cada cubo. De esta manera, sólo los cubos

cuyos vértices sufren algún cambio de configuración deben recalcular su triangulación. En la tabla 1 se resumen los casos posibles y las acciones a realizar.

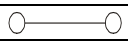
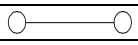
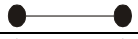
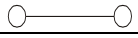
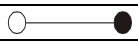
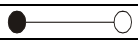
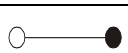
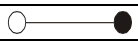
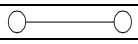
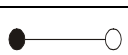
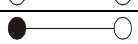
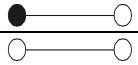
Situación inicial	Situación final	Acción
		Ninguna
		Ninguna
		Ninguna
		Intersectar arista-volumen y asignar porcentaje*
		Intersectar arista-volumen y asignar porcentaje*
		Intersectar arista-volumen y asignar porcentaje menor**
		Ninguna
		Intersectar arista-volumen y asignar porcentaje menor**
		Ninguna

*Tabla 1. Acciones a realizar dependiendo de la configuración de los vértices. En negro se muestran los vértices interiores, en blanco los exteriores. * Recalcular triangulación. ** Recalcular triangulación si porcentaje de interioridad cambia.*

3.3 Adaptatividad

Una vez realizada la triangulación básica de Marching Cubes, aplicamos los principios del PRM para conseguir una adaptación local a la precisión requerida, realizando los cálculos sólo para los triángulos recalculados en este último paso.

Para cada arista del triángulo debemos calcular su longitud y el ángulo que forman entre sí los vectores gradiente de la superficie medidos en sus extremos. Si el ángulo asociado a alguna arista supera un umbral U o su longitud un umbral L, el triángulo tendrá que ser partido por el punto medio de dicha arista. Dependiendo de dicho criterio, dividiremos el triángulo según el esquema de la figura 1.

A los triángulos resultantes les aplicaremos el mismo método recursivamente hasta un límite máximo H (altura máxima del árbol de recursión).

En el caso de la simulación del mecanizado no disponemos de la superficie que queremos triangular, por lo que no es posible calcular su gradiente (para isosuperficies o superficies libres esto es inmediato). Sin embargo, debemos fijarnos en que la superficie que estamos aproximando es la de la herramienta que realiza el desbaste. En definitiva, el vector gradiente es el de la superficie de la herramienta. Puesto que la geometría de las herramientas es habitualmente simple y estándar la normal a la superficie se puede calcular de forma sencilla y rápida.

Por ejemplo, en el caso de una herramienta esférica, la normal a un punto p de la superficie de la herramienta no es más que el vector que pasa por el centro de la

herramienta y el punto p. El caso de los demás tipos de herramienta también puede calcularse de forma inmediata, como puede comprobarse fácilmente.

Como ocurre con el cálculo del gradiente, la obtención del punto medio adaptado sobre la superficie también resulta mucho más sencilla que para isosuperficies (era necesario un proceso de divisiones sucesivas) e incluso que para superficies libres (debíamos calcular el punto en el espacio paramétrico y aplicar la función de la superficie para obtenerlo en el de representación).

Sea de nuevo una herramienta esférica de centro c y radio r y dos puntos p y q sobre la superficie. Podemos constatar que el punto medio p_m entre p y q situado sobre la superficie de la herramienta es un punto situado sobre el vector suma de los vectores **cp** y **cq** a una distancia r del centro c de la herramienta. El cálculo de los puntos medios en el caso de otros tipos de herramientas es igualmente sencillo.

Lo que en principio parecía ser un inconveniente (no contar con la superficie que queremos obtener) es, curiosamente, una ventaja a la hora de reducir costes.

4 Pruebas y resultados

Presentamos en primer lugar un estudio del coste del algoritmo y, a continuación, comprobamos empíricamente el método y sus resultados reales a través de una serie de pruebas con diferentes trayectorias. Los resultados de nuestro algoritmo se comparan con el de Marching Cubes original para comprobar si las mejoras realizadas se reflejan en los costes y en la calidad de los resultados.

4.1 Coste del algoritmo

Para el cálculo del coste teórico a priori del algoritmo hemos dividido el método en dos etapas: la primera de ellas estará compuesta por los cálculos necesarios para la primera posición de la herramienta. La segunda etapa estaría formada por estos mismos cálculos en las posiciones sucesivas.

En la primera etapa se realizan tres procesos: cálculo de la configuración de los vóxeles, poligonalización y adaptación local. Sólo en esta primera etapa se realizan los cálculos para todo el objeto. El coste de estos procesos es:

- El coste de la configuración de los vóxeles (equivalente al que realiza Marching Cubes) supone que para cada vértice del volumen total del paralelepípedo (el número de vértices es del orden del número de vóxeles) debemos realizar una prueba de interioridad/exterioridad y la intersección de la arista con la herramienta. Ambos procesos se componen de cálculos sencillos basados en

distancias euclídeas y suma de vectores, luego este coste se considera constante ($k_{prueba\text{-}inters}$) y el coste de configuración depende del número de vóxeles:

$$C_{config} = k_{prueba-inters}\left(V_x \cdot V_y \cdot V_z\right)$$

con V_x, V_y y V_z el número de vóxeles del paralelepípedo en sentido x, y ,z.

- En el cálculo del coste de la poligonalización, para cada vóxel debemos obtener la poligonalización según el algoritmo de Marching Cubes. Si de cada vóxel se pueden obtener un máximo de 4 triángulos, una cota del coste es $4 \cdot k_{triang} \cdot (V_x \cdot V_y \cdot V_z)$ donde k_{triang} indica el coste aproximadamente constante de las operaciones para construir los triángulos. Integrando el 4 en k_{triang} nos queda:

$$C_{polig} = k_{triang} \cdot \left(V_x \cdot V_y \cdot V_z\right)$$

- El coste de la adaptación local es el de de un algoritmo recursivo en el que la profundidad de recursión depende de la precisión requerida y de la herramienta que utilicemos. No obstante, puesto que fijamos una altura máxima H del árbol de recursiones, en el peor de los casos se producirán 4^H iteraciones para cada triángulo, puesto que como máximo se producen 4 particiones por triángulo (figura 1) y en el mejor de los casos ninguna. Si se produjeran todas las particiones posibles de todos los triángulos de cada vóxel, el número de iteraciones sería $4^H \cdot (V_x \cdot V_y \cdot V_z)$. Si integramos 4^H con los costes de los cálculos adicionales en una única constante, nos quedará:

$$C_{adapt} = k_{adapt} \cdot \left(V_x \cdot V_y \cdot V_z\right)$$

En la segunda etapa los procesos anteriores se repiten para cada punto de la trayectoria donde se coloca la herramienta. En cada una de estas posiciones se recalcula la configuración de los vóxeles, la poligonalización y la adaptación local, pero sólo dentro del entorno de vecindad de la herramienta, que viene marcado por su radio. Para una herramienta de radio r, el número de vóxeles que se encuentran dentro de su entorno de vecindad pueden considerarse como constante (en una misma trayectoria no se cambia de herramienta), por lo que los valores V_x, V_y y V_z pueden cambiarse por una constante $k_{entorno}$. Si la trayectoria tiene un total de n puntos, el coste de la segunda etapa vendrá dado por la expresión siguiente, donde en $k_{entorno}$ hemos integrado los costes constantes de los cálculos adicionales.

$$C_{etapa2} = k_{entorno} \cdot n$$

En definitiva, el coste total final del algoritmo vendrá dado por:

$$C_{SIA} = \left(k_{prueba-inters} \cdot \left(V_x \cdot V_y \cdot V_z\right) + k_{triang} \cdot \left(V_x \cdot V_y \cdot V_z\right) + k_{adapt} \cdot \left(V_x \cdot V_y \cdot V_z\right)\right) + k_{entorno} \cdot n$$

Suponiendo que el paralelepípedo y los vóxeles tienen tamaño constante (V_x, V_y y V_z son siempre iguales) el orden del algoritmo es lineal respecto a n, es decir, para un mismo bloque de material y una misma herramienta el coste depende linealmente de la longitud de la trayectoria.

En cuanto al algoritmo básico de Marching Cubes, los costes de configuración de los vóxeles y de poligonalización son los mismos, pero en este caso desaparece el coste de ajuste local. No obstante, al no ser un método incremental, para poder representar el objeto en cada etapa es necesario recalcular la configuración y la poligonalización para cada punto de la trayectoria. El coste total es:

$$C_{MC} = \left(k_{prueba-inters} \cdot \left(V_x \cdot V_y \cdot V_z\right) + k_{triang} \cdot \left(V_x \cdot V_y \cdot V_z\right)\right) \cdot k_{MC} \cdot n$$

En definitiva y suponiendo también un paralelepípedo y unos vóxeles de tamaño constante el orden del algoritmo es de nuevo lineal respecto a *n*. Sin embargo, tal y como podemos apreciar en la figura 3, este coste es considerablemente superior al del método propuesto. Esto se debe a dos razones fundamentales:

- En C_{SIA} los valores $k_{prueba\text{-}inters} \cdot (V_x \cdot V_y \cdot V_z) + k_{triang} \cdot (V_x \cdot V_y \cdot V_z)$ se suman a la expresión $k_{entorno} \cdot n$, mientras que en C_{CM} se multiplican.
- En Marching Cubes se consigue mayor precisión a cambio de disminuir el tamaño del cubo, lo que repercute sobre el coste de forma cúbica, mientras que en nuestro algoritmo el aumento de precisión se consigue incrementando la profundidad del árbol, de manera que la repercusión en el coste final es lineal.

4.2 Pruebas

En las figuras 3 y 4 se presentan de forma gráfica los resultados obtenidos por el método de Simulación Incremental Adaptativa y de Marching Cubes. Las pruebas se han realizado en un ordenador Pentium IV a 2.4 GHz y con 1 GB de RAM. Para poder comparar adecuadamente los resultados, los parámetros utilizados son:

- Para la Simulación Incremental Adaptativa el tamaño de vóxel es de 5 mientras que la adaptación local tiene como umbrales para la longitud de los lados de un triángulo un valor de 1 y para el ángulo de las normales 0.5 radianes.
- En el algoritmo Marching Cubes el tamaño del vóxel es de 1, para que la precisión obtenida en la malla sea equivalente a la anterior.
- En todos los casos la herramienta es esférica y tiene un radio de 5.
- Las pruebas se han realizado con trayectorias de longitudes entre 10 y 200 puntos, aplicando para cada trayectoria los dos algoritmos.

En la figura 3 se compara el tiempo de proceso para los dos algoritmos con respecto a la longitud de la trayectoria y al número de polígonos obtenidos. En ambos casos se observa la enorme diferencia existente entre los tiempos consumidos por los dos métodos. El algoritmo de Simulación Incremental Adaptativa obtiene los resultados en menos de 2 segundos en todos los casos, mientras que el caso de Marching Cubes las trayectorias de mayor longitud consumen hasta 160 segundos. Debemos tener en cuenta que en ambos procesos se

obtiene una poligonalización total en cada punto de la herramienta para mostrar el proceso de simulación completo. Para que podamos apreciar convenientemente todo el proceso de simulación necesitaremos con toda probabilidad más de 2 segundos, por lo que podemos decir que el algoritmo propuesto se adapta a las características de tiempo real que perseguíamos.

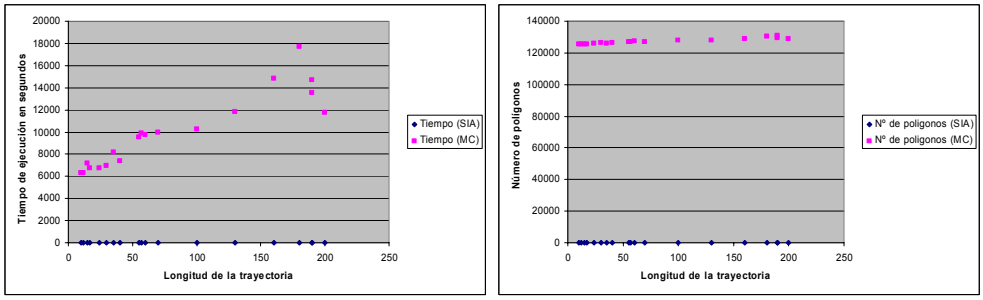


Figura 3: Comparación de los tiempos de proceso para los algoritmos de Simulación Incremental Adaptativa (SIA) y Marching Cubes (MC).

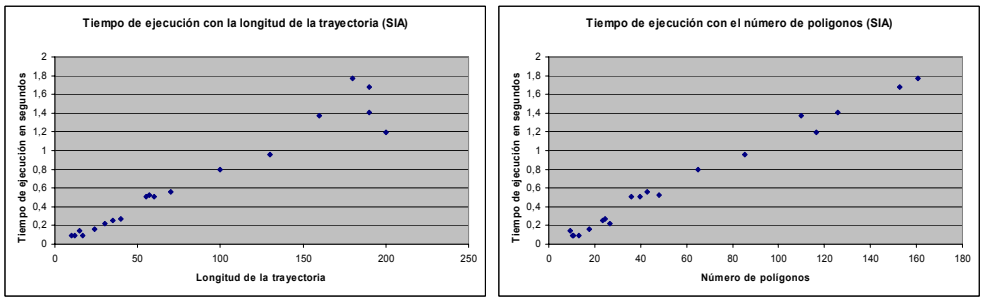


Figura 4: Tiempo de proceso con respecto a la longitud de la trayectoria y al número de polígonos del algoritmo de Simulación Incremental Adaptativa (SIA).

En estas figuras puede observarse también la linealidad del algoritmo Marching Cubes con respecto a la longitud de la trayectoria, mientras que el número de polígonos permanece prácticamente constante (esto es lógico puesto que por la forma en que se comporta este algoritmo el número de polígonos sufre pocos cambios). En cuanto a nuestro algoritmo, en ambos casos se comporta linealmente, aunque no puede apreciarse en la figura 3 debido a que los valores obtenidos son comparativamente mucho menores que los de Marching Cubes.

En la figura 4 sí que puede observarse, sin embargo, la linealidad de la Simulación Incremental Adaptativa.

En la figura 5 se muestra el resultado obtenido con los dos algoritmos para la misma trayectoria: el algoritmo propuesto consigue obtener precisiones similares con una malla mucho menos densa que MC donde no es necesario.

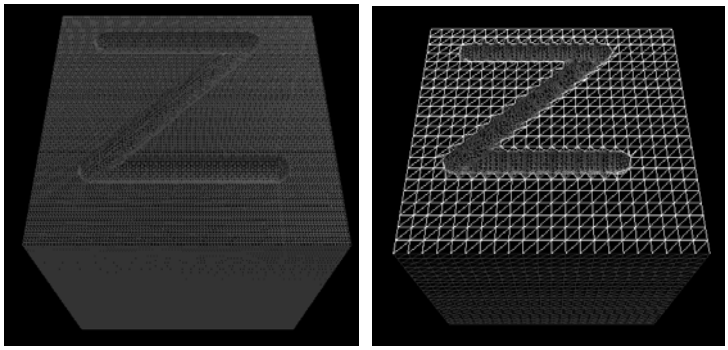


Figura 5: Comparación de la malla obtenida mediante los algoritmos de Marching Cubes (izquierda) y Simulación Incremental Adaptativa (derecha).

4 Conclusiones

Hemos presentado un método para obtener una malla poligonal que represente adecuadamente un objeto obtenido por desbaste de material utilizando una herramienta. Este método presenta las siguientes características:

- Aprovecha la coherencia espacial, es decir, utiliza las restricciones espaciales del problema para disminuir su coste. En este sentido procesa en cada paso sólo aquellos cubos que se encuentran en el área de influencia de la herramienta, área que vendrá marcada como un entorno de vecindad alrededor del punto de aplicación de radio el de la herramienta. Esta característica reduce drásticamente el número de vóxeles a tratar en cada paso.
- Es un procedimiento incremental, es decir, en cada paso del algoritmo los cálculos realizados se utilizan como entrada del siguiente: tanto la configuración de los vóxeles como su poligonalización se reutilizan para disminuir los costes.
- En cada paso discreto de herramienta disponemos de la configuración completa de todos los vóxeles del objeto y de la malla poligonal, con lo que es posible representar el objeto de forma inmediata.
- El proceso de poligonalización es adaptativo de manera que la precisión de la malla se adapta localmente al gradiente de la superficie de la herramienta. De esta manera se obtienen mallas que presentan una mayor densidad en aquellas zonas donde la curvatura de la superficie lo requiere.

Las características anteriores hacen que los tiempos de proceso se reduzcan drásticamente si los comparamos con otros métodos tradicionales. Esto permite realizar una visualización en tiempo real, especialmente adecuada para visualizar la simulación del mecanizado de piezas.

Referencias

[1] Jang, D., Kim, K., Jung, J. "Voxel-Based Virtual Multi-Axis Machining". Int. Journal of Advanced Manufacturing Technology, 16:709-713, 2000.

[2] Wyvill, G; McPheeters, C. y Wyvill, B. "Data Structure for Soft Object". Visual Computer, vol. 2, num. 4, págs. 227-34. 1986.

[3] Lorensen, W.E. and Cline, H.E. "Marching Cubes: A High Resolution 3D Surface Construction Algorithm". ACM Computer Graphics, vol 21, num. 24, págs. 163-169. 1987.

[4] Puchol García, Juan A.; Sáez Martínez, Juan M. y Molina Carmona, Rafael. "Postproceso de rectificación de metapolígonos". Actas del IX Congreso Español de Informática Gráfica, CEIG99, págs. 331-342. 1999.

[5] Sáez Martínez, Juan M.; Jimeno Morenilla, Antonio M.; Puchol García, Juan A. y Molina Carmona, Rafael. "Adaptación del PRM sobre superficies libres. Aplicación a entornos CAD/CAM". Actas del X Congreso Español de Informática Gráfica, CEIG00. 2000.

[6] Puchol García, Juan A.; Molina Carmona, Rafael y García Quintana, Candelaria. "Conversión paramétrica de Blobs a modelo de fronteras en tiempo real". IX Congreso Int. de Ingeniería Gráfica, págs. 465-474. 1997.

[7] Payne, Bradley A. y Toga, Arthur W. "Surface mapping brain function on 3D models". IEEE Computer Graphics and Applications, págs. 33-41. 1990.

[8] Kaneko, Toyohisa y Yamamoto, Yuuki. "Volume-Preserving Surface Reconstruction from Volume Data". Proceedings of the 1997 International Conference on Image Processing (ICIP'97). IEEE. 1997.

Agradecimientos

Agradecemos a Juan Manuel Sáez Martínez su colaboración en la implementación de las versiones preliminares de este método.