VNIVERSIDAD NACIONAL AVPNOMA DE MEXICO

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN
FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLÁN

"DESARROLLO DE SOFTWARE INTEGRADO DE DINÁMICA MOLECULAR Y SIMULADOR GRÁFICO UTILIZANDO OPENGL Y PROGRAMACIÓN EN PARALELO"

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN CIENCIAS (COMPUTACIÓN)

P R E S E N T A:

JOSÉ LUIS GARZA RIVERA

DIRECTOR DE TESIS: DR. VLADIMIR TCHIJOV

México, D.F. 2005.

Dedicada a Angélica y mis hijos.Gracias.

AGRADECIMIENTOS

Quisiera agradecer por todo el apoyo, tiempo y recursos a:

La Universidad Nacional Autónoma de México y en particular a la Facultad de Estudios Superiores Cuautitlán.

Especialmente a mi tutor el Dr. Vladimir Tchijov.

Al Consejo Nacional de Ciencia y Tecnologia (CONACYT) por el apoyo económico brindado.

A los profesores y todos aquellos que hacen posible el Posgrado.

A los miembros del Jurado.

A mis guías, compañeros y amigos de Biblioteca, Licenciatura en Informática y Centro de Cómputo de la FES Cuautitlán.

ÍNDICE

OBJETIVO

Elaborar un sistema de computo integrado que realice cálculos numéricos de dinámica molecular y despliegue su respectiva simulación gráfica en tiempo real. Valiéndose del uso de una interface de graficación de alto nivel (OpenGL) y el uso de hilos (***threads***) permitiendo encauzar los recursos de computo de manera óptima; utilizando para su desarrollo un lenguaje de programación eficiente, dentro de los sistemas operativos Microsoft Windows y Linux.

INTRODUCCIÓN

Los programas de dinámica molecular (DM) de estado sólido y líquido son herramientas indispensables en simulación computacional. Por otro lado, la visualización científica y graficación de estructuras moleculares son de gran utilidad en el análisis de los resultados de cálculos de DM. Aunque existen programas de modelación molecular, incluso de uso libre, el usuario por lo general, no puede modificar el programa ni extraer todos los datos necesarios. La graficación de los resultados se hace al finalizar los cálculos o bien usando numerosos archivos temporales de salida y algún visualizador independiente, lo que de alguna manera dificulta los estudios. Sin embargo, existen herramientas computacionales tales como programación con hilos (*threads*) y OpenGL que permiten el desarrollo de sistemas de cómputo integrales de DM y graficación facilitando notablemente la interacción entre el usuario y el programa.

Se desarrolla un sistema de cómputo original compuesto de un programa de cálculos numéricos de dinámica molecular y un simulador gráfico. El programa de dinámica molecular se basa en el esquema de cuaterniones , utiliza tecnología orientada a objetos y lenguaje de programación C++. El simulador gráfico está escrito en C++ y utiliza el paquete de visualización computacional OpenGL.

El sistema propuesto permite generar los cálculos basándose en los datos de entrada. El usuario podrá ver el estado del modelo, e interactuar con la simulación gráfica en tiempo real durante el tiempo de ejecución, o bien, con el modelo final al término de la ejecución de los cálculos.

Mientras que el usuario no interactué con la representación gráfica, el subsistema de graficación entrará en un estado de espera y los cálculos podrán continuar de manera desahogada optimizando el proceso de cálculo, el cual, en el caso de los modelos estudiados en la Facultad de Estudios Superiores Cuautitlán lleva en promedio varios días de ejecución.

Al utilizar OpenGL el proceso de graficación será también altamente eficiente y transportable. OpenGL se emplea incluso para la construcción de la interface gráfica del usuario, la cual facilita el manejo de todo el sistema.

La contribución principal del proyecto de esta tesis es su integración a los actuales trabajos de investigación de la Facultad de Estudios Superiores Cuautitlán, específicamente con relación al proyecto PAPIIT No. 105401 "Modelación matemática de sistemas multifásicos y multicomponentes en

problemas del medio ambiente" (investigador responsable: Dr. Vladimir Tchijov), por lo tanto, los programas obtenidos se aplicarán de manera directa e inmediata a la investigación realizada en esta Facultad.

Otra de las contribuciones es que los programas se desarrollan tanto para la plataforma Microsoft Windows por ser más utilizada, así como para la plataforma LINUX por ser de uso libre, de tal manera que tanto el código fuente como el programa pueden ser distribuidos y usados en ambas plataformas.

En el CAPÍTULO 1 se verán conceptos básicos sobre simulación y modelación. También se introducen los conceptos generales y métodos de modelación molecular y visualización.

El CAPÍTULO 2 serán explicados los algoritmos empleados, los métodos de solución de sistemas de ecuaciones, la complejidad del sistema, así como algunas técnicas de optimización .

Las interfaces de graficación, generación de ventanas y eventos serán tratadas en el CAPÍTULO 3.

En el CAPÍTULO 4 se abordarán cuestiones referentes a concurrencia y paralelización. En este mismo capítulo se explicarán los diferentes modelos de concurrencia, las ventajas y consecuencias del manejo de este tipo de programas.

En el CAPÍTULO final, se abordará la integración en un sistema computacional de los elementos tratados en los capítulos anteriores. La segmentación del código y los algoritmos empleados. También se mostrarán los resultados obtenidos con un modelo de prueba. Así como los tiempos de ejecución generados tanto en Microsoft Windows como en Linux.

CAPÍTULO 1. Simulación y Modelación

1.1 Simulación Computacional

La simulación computacional es una herramienta de la investigación científica actual. Permite predecir el comportamiento de sistemas naturales, o bien, realizar la simulación de los mismos en condiciones que no pueden obtenerse en laboratorio, debido a su costo, complejidad o tiempo del experimento.

En el caso de la simulación a escala molecular, la utilidad de los sistemas de simulación es todavía mayor dadas las dimensiones de los sistemas naturales estudiados.

De manera general se puede definir a la simulación como la explotación de un modelo para predecir las consecuencias de situaciones hipotéticas. [SEVERANCE 2001].

El objetivo de la simulación computacional es resolver los modelos teóricos en su total complejidad, mediante la resolución numérica de las ecuaciones involucradas, haciendo uso intensivo de computadoras. [GUTIERREZ 2001].

Uno de los objetivos principales de la simulación es la predicción de los sistemas naturales [HERRERA 2003], para tal fin, es necesario la generación de modelos que constituyen una representación del sistema a simular.

La generación de modelos comprende 3 etapas:

1. Modelación Matemática:
En esta fase, el sistema se reduce y representa por medio de un modelo matemático, conformado por un conjunto de ecuaciones que requieren solución.

De manera general, los sistemas pueden ser clasificados en **continuos** y **discretos**. Los primeros son aquellos cuyas variables cambian continuamente con respecto al tiempo. Por otro lado, en los sistemas discretos, las variables cambian únicamente en instantes específicos (finitos) del tiempo.

Los sistemas pueden también clasificarse en lineales y no-lineales, los sistemas lineales son aquellos que satisfacen las siguientes condiciones:

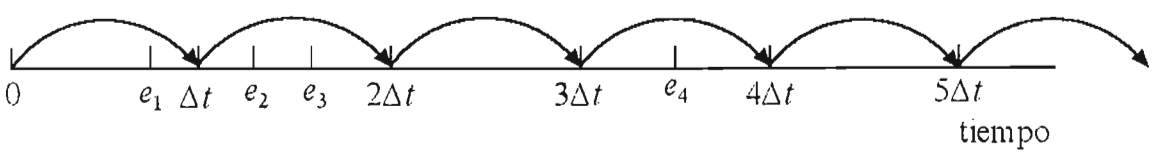
a) Al multiplicar las entradas del sistema por una constante, trae como resultado la multiplicación de las salidas por dicha constante.
b) El resultado obtenido al aplicar un número de entradas simultáneamente, es igual a la suma de los resultados obtenidos al aplicar cada una de las entradas de manera individual.

Los sistemas no lineales son los que no cumplen con las condiciones mencionadas.

2. Selección del método de solución

De acuerdo al modelo matemático resultante, es posible obtener resultados exactos. Sin embargo, en muchos modelos únicamente se llegar a resultados aproximados. ;como consecuencia de la complejidad del sistema y el número de cálculos implicados que significan un tiempo de ejecución muy largo.

Adicionalmente, al tratar sistemas continuos mediante el uso de equipos de cómputo electrónicos, se obtienen los valores de las variables continuas en puntos específicos del tiempo. La simulación de un sistema continuo en una computadora digital es de hecho, un sistema discreto con respecto al tiempo, cuyas variables solamente cambian en distintos (finitos) instantes en el tiempo. Ver figura 1-1. [KHEIR 1988].



$e_1 \ldots e_n$ - Eventos

$\Delta t \ldots n\Delta t$- *Puntos de discretización en el tiempo.*

Figura 1-1 Discretización de un sistema con respecto al tiempo.

La selección del método numérico da como resultado un conjunto de ecuaciones algebraicas.

3.- Las ecuaciones algebraicas pueden ser implementadas en programas computacionales mediante lenguajes de programación, generándose un **Modelo Computacional** [VERA 2003], conformado por el conjunto de programas que permiten implementar la simulación.

1.2 Modelación Molecular

Modelación Molecular es la generación, visualización, manipulación y predicción de estructuras moleculares realísticas y sus propiedades físico-químicas asociadas. [RIUS 2000].

Dentro de la modelación molecular existen 2 técnicas principales:

1.2.1 Métodos de Monte Carlo

Estos métodos fueron diseñados originalmente por Von Neumann, Ulam y Metropolis al final de la segunda guerra mundial. El nombre Monte Carlo fue adoptado debido al uso de números generados de manera aleatoria. Los métodos de Monte Carlo son probabilísticos y constituyen una exploración al azar de un espacio tridimensional disponible para un sistema dado, mediante la utilización de algoritmos de generación de números aleatorios para cambiar los sistemas de coordenadas [RIUS 2000].En la presente tesis no son empleados los métodos de Monte Carlo.

1.2.2 Dinámica Molecular

La dinámica molecular (DM) emplea métodos deterministas, permite tomar en cuenta el efecto de las interacciones entre las partículas, y obtener las propiedades dinámicas del sistema de partículas [ALLEN 1987] ,la simulación se realiza aplicando las leyes del movimiento de Newton.

Una parte central de todo programa de DM lo constituye el algoritmo de integración de las ecuaciones de movimiento. Dichas ecuaciones deben ser resueltas numéricamente. Dadas las posiciones y velocidades iniciales a un tiempo inicial t_0, la tarea del algoritmo es entregar las posiciones y velocidades al tiempo $t + \Delta t$.

La información que genera la ejecución de un programa utilizando Dinámica Molecular es principalmente la posición y la velocidad de cada partícula del sistema en cada instante de tiempo.

1.2.3 Visualización

Un elemento importante de la Modelación Molecular es la visualización, ya que permite analizar de manera rápida el modelo molecular, generalmente conformado por un conjunto muy extenso de datos de entrada y salida. También es posible observar el desarrollo de la simulación durante la ejecución del modelo computacional. La visualización utiliza un conjunto de aplicaciones y herramientas de programación que emplean los datos resultantes de la simulación numérica.

Uno de los recursos empleados en la simulación computacional es la paralelización, con ella es posible resolver problemas que implican calculo intensivo. Entre sus variantes se encuentran la paralelización por software, utilización de equipos con varios procesadores o el uso de sistemas de computo distribuido.

La técnica empleada en esta investigación es la de Dinámica Molecular, con el método numérico de integración de ecuaciones de traslación y rotación de moléculas, mediante el algoritmo de Verlet. La herramienta de visualización es OpenGL. La paralelización y concurrencia se realiza por medio de APIs (*Aplication Program Interfaces*) del Sistema Operativo.

CAPÍTULO 2. Dinámica Molecular

2.1 Algoritmos de Dinámica Molecular

Los algoritmos empleados en este trabajo se aplican a modelos moleculares rígidos, es decir, aquellos en los que las longitudes de los enlaces entre átomos y los ángulos que estos forman se consideran fijos.

Son empleadas las ecuaciones de movimiento clásicas Newton-Euler y los principios de la mecánica de cuerpos rígidos.

Aunque los sistemas moleculares en la vida real no son rígidos debido a las fuerzas intramoleculares e intermoleculares, los movimientos en los enlaces ocurren comparativamente mucho más rápido que los movimientos al nivel de moléculas. Debido a esto, es necesario un tamaño de paso de tiempo mucho menor para resolver las ecuaciones de movimiento[ALLEN 1987] ; dando como resultado un incremento en el número pasos de toda la simulación y el número de cálculos por molécula, implicando un costo computacional muy alto.

Una forma de resolver este inconveniente es considerar las longitudes y ángulos de los enlaces entre átomos como si fueran fijos.

2.1.1 Ecuaciones de Movimiento

La fuerza ejercida por el átomo α de la molécula i en el átomo β de la molécula j se define como $f_{i\alpha j\beta}$. La fuerza total que actúa sobre la molécula i esta dada por:

$$Fi = \sum_{j} \sum_{\beta} \sum_{\alpha} f_{i\alpha j\beta} \tag{2.1}$$

y el torque esta definido por:

$$N_i = \sum_{\alpha} (r_{i\alpha} - R_i) \times f_{i\alpha} \tag{2.2}$$

Donde:

$$R_i = \frac{1}{M_i \sum_{\alpha} m_{i\alpha} r_{i\alpha}} \tag{2.3}$$

es el la posición del centro de masa i.

En mecánica clásica, el movimiento molecular se divide en traslación del centro de masa y rotación sobre dicho centro [GOLDSTEIN 1980]. De tal forma que se tiene:

Traslación:

$$M_i \ddot{R}_i = F_i \tag{2.4}$$

Rotación:

$$I_i \cdot \dot{\omega}_i - \omega_i \times I_i \cdot \omega_i = N_i \tag{2.5}$$

Donde:

ω_i es la velocidad angular de la molécula i.

$$I_i = \sum_{\alpha} m_{i\alpha} \left(p_{i\alpha}^2 1 - p_{i\alpha} p_{i\alpha} \right) \tag{2.6}$$

es el tensor de inercia

$$p_{i\alpha} = r_{i\alpha} - R_i \tag{2.7}$$

es el sistema de coordenadas relativo al centro de masa molecular.

R_i como se había especificado antes es la posición del centro de masa de la molécula i.

El cálculo de las trayectorias de los centros de masa y rotación de las moléculas implica la solución de sistemas de ecuaciones diferenciales obtenidas a partir de las ecuaciones (2.4) y (2.5).

La forma de resolver estos sistemas es mediante métodos numéricos.

2.1.2 Integración

Dadas las posiciones y velocidades moleculares a un tiempo *t*, se intenta obtener las posiciones y velocidades a un tiempo $t+\Delta t$ con un grado de precisión suficiente.

El sistema es resuelto por pasos mediante el método de diferencias finitas [ALLEN 1987].El tamaño de paso de tiempo Δt deberá ser significativamente menor que el tiempo que le toma típicamente a una molécula recorrer su propia longitud.

Si la trayectoria de la molécula es continua, es posible obtener una estimación de las posiciones y velocidad al tiempo $t+\Delta t$ mediante una serie de Taylor de la siguiente forma.

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''}{2!}(x - x_0)^2 + \ldots + \frac{f^{(n)}}{n!}(x - x_0)^n \qquad (2.8)$$

Donde:

x	$t+\Delta t$
x_0	t
r	Posición
a	Aceleración
v	velocidad
b	tercera derivada de r

Se puede expresar como:

$$r(t+\Delta t) = r(t) + r'(t)\Delta t + \tfrac{1}{2} r''(t)\Delta t^2 + \tfrac{1}{6} b(t)\Delta t^3 \ldots . \qquad (2.9)$$

Haciendo lo mismo para la aceleración y la velocidad

$$v(t+\Delta t) = v(t) + v'(t)\Delta t + \tfrac{1}{2} v''(t)\Delta t^2 + \ldots \qquad (2.10)$$

$$a(t+\Delta t) = a(t) + a'(t)\Delta t + \ldots \qquad (2.11)$$

$$b(t+\Delta t) = b(t) + \ldots \qquad (2.12)$$

dado que:

$$r'(t) = v(t) \qquad (2.13)$$

$$r''(t) = v'(t) = a(t) \qquad (2.14)$$

$$r'''(t) = v''(t) = a'(t) = b(t) \qquad (2.15)$$

Las ecuaciones se pueden describir como:

$$r(t+t\Delta) = r(t) + v(t)\Delta t + \tfrac{1}{2}a(t)\Delta t^2 + \tfrac{1}{6}b(t)\Delta t^3 + \ldots \quad (2.16)$$

$$v(t+t\Delta) = v(t) + a(t)\Delta t + \tfrac{1}{2}b(t)\Delta t^2 + \ldots \quad (2.17)$$

$$a(t+t\Delta) = a(t) + b(t)\Delta t + \ldots \quad (2.18)$$

$$b(t+\Delta t) = b(t) + \ldots \quad (2.19)$$

Para efectos computacionales se pueden aproximar como:

$$r(t+t\Delta) = r(t) + v(t)\Delta t + \tfrac{1}{2}a(t)\Delta t^2 + \tfrac{1}{6}b(t)\Delta t^3 \quad (2.20)$$

$$v(t+t\Delta) = v(t) + a(t)\Delta t + \tfrac{1}{2}b(t)\Delta t^2 \quad (2.21)$$

$$a(t+t\Delta) = a(t) + b(t)\Delta t \quad (2.22)$$

$$b(t+\Delta t) = b(t) \quad (2.23)$$

Lo siguiente será seleccionar un algoritmo para la solución de este sistema de ecuaciones.

Para elegir de entre los algoritmos disponibles el que se utilizará, se deben considerar características como facilidad de programación, tamaño máximo de paso Δt, complejidad y el menor grado de error con respecto a la trayectoria clásica.

Uno de los algoritmos más empleados en Dinámica Molecular es el algoritmo de Verlet [ALLEN 1987], este parte de la ecuación (2.20) para $r(t+\Delta)$ y $r(t-\Delta)$ escrita como:

$$r(t+\Delta t) = r(t) + v(t)\Delta t + \tfrac{1}{2}a(t)\Delta t^2 + \tfrac{2}{3!}b(t)\Delta t^3 \quad (2.24)$$

$$r(t-\delta t) = r(t) - v(t)\Delta t + \tfrac{1}{2}a(t)\Delta t^2 - \tfrac{2}{3!}b(t)\Delta t^3 \quad (2.25)$$

Si se emplea solamente la ecuación (2.20) se obtiene un error de Δt^4 en la posición de la partícula.

Si se restan se obtiene:

$$r(t+\Delta t) - r(t-\Delta t) = 2v(t)t + \tfrac{2}{6}b(t)\Delta t^3 \quad (2.26)$$

despejando

$$v(t) = \frac{r(t+\Delta t) - r(t-\Delta t)}{2\Delta t} - \frac{1}{3} b(t)\Delta t^3 \tag{2.27}$$

se realiza la aproximación siguiente:

$$v(t) = \frac{r(t+\Delta t) - r(t-\Delta t)}{2\Delta t} \tag{2.28}$$

obteniendo un error del orden de Δt^2 en la velocidad de la partícula.

Retomando la ecuación (2.22) se suma a(t) en ambos lados y se multiplica por ½ Δt

$$a(t+\Delta t) = a(t) + b(t)\Delta t \tag{2.29}$$
$$\tfrac{1}{2}(a(t+\Delta t) + a(t))\Delta t = \tfrac{1}{2}(2a(t) + b(t)\Delta t)\Delta t \tag{2.30}$$
$$\tfrac{1}{2}(a(t+\Delta t) + a(t))\Delta t = a(t)\Delta t + \tfrac{1}{2} b(t)\Delta t^2 \tag{2.31}$$

Se substituye la ecuación (2.30) en (2.21)

$$v(t+\Delta t) = v(t) + \tfrac{1}{2}(a(t+\Delta t) + a(t))\Delta t \tag{2.32}$$

Para calcular la velocidad de medio paso se utiliza:

$$v(t+\tfrac{1}{2}\Delta t) = v(t) + \tfrac{1}{2} a(t)\Delta t \tag{2.33}$$

despejando

$$v(t) = v(t+\tfrac{1}{2}\Delta t) - \tfrac{1}{2} a(t)\Delta t \tag{2.34}$$

al sustituir (2.34) en (2.32)

$$v(t+\Delta t) = v(t+\tfrac{1}{2}\Delta t) + \tfrac{1}{2} a(t+\Delta t)\Delta t \tag{2.35}$$

Al rescribir las ecuaciones (2.20), (2.33) y (2.35) remplazando $a = \frac{F}{m}$

$$r(t+\Delta t) = r(t) + v(t)\Delta t + \frac{1}{2m} F(t)\Delta t^2 \tag{2.36}$$

$$v(t+\tfrac{1}{2}\Delta t) = v(t) + \frac{1}{2m}F(t)\Delta t \tag{2.37}$$

$$v(t+\Delta t) = v(t+\tfrac{1}{2}\Delta t) + \frac{1}{2m}F(t+\Delta t)\Delta t \tag{2.38}$$

2.1.3 Descripción del Algoritmo de Traslación

Empleando las ecuaciones obtenidas en el punto anterior, se puede resumir el algoritmo de traslación de la siguiente manera:

1. Al tiempo t se conocen las posiciones r(t) y velocidades v(t) de todas las partículas
2. Se calcula la fuerza que actúa en el tiempo t en cada partícula donde F(t)

$$F_{ij} = -\frac{1}{r_{ij}}\frac{dv(r_{ij})}{dr_{ij}}\vec{r}_{ij} \tag{2.39}$$

3. Se calcula la posición r(t+Δt) usando la ecuación (2.36)
4. Se calculan las velocidades de medio paso v(t+ ½ Δt) utilizando la ecuación (2.37)
5. Al tenerse ya las posiciones r(t+Δt) se calculan F(t+Δt)
6. Se calculan las velocidades v(t+Δt) con la ecuación (2.38).
7. Ir a 1 hasta completar el número de pasos.

El número de pasos, se refiere a la cantidad de pasos de tiempo empleada para la simulación.

2.1.4 Orientación Molecular

La orientación de cada molécula esta definida mediante la relación de dos sistemas de coordenadas. Un sistema de ejes fijo con respecto al espacio de la simulación, y otro con respecto a la molécula o cuerpo rígido.

Si se define ***efs*** como *eje fijo de la simulación* y ***efm*** como *eje fijo de la molécula* es posible cambiar de una base a otra mediante:

$$efs = \tilde{A} \cdot efm \tag{2.40}$$

$$efm = A \cdot efs \tag{2.41}$$

donde la matriz de rotación A es[ALLEN 1987]:

$$A = \begin{pmatrix} \cos\phi\cos\psi - \operatorname{sen}\phi\cos\theta\operatorname{sen}\psi & \operatorname{sen}\phi\cos\psi + \cos\phi\cos\theta\operatorname{sen}\psi & \operatorname{sen}\theta\operatorname{sen}\psi \\ -\cos\phi\operatorname{sen}\psi - \operatorname{sen}\phi\cos\theta\cos\psi & -\operatorname{sen}\phi\operatorname{sen}\psi + \cos\phi\cos\theta\cos\psi & \operatorname{sen}\theta\cos\psi \\ \operatorname{sen}\phi\operatorname{sen}\theta & -\cos\phi\operatorname{sen}\theta & \cos\theta \end{pmatrix} \quad (2.42)$$

los valores φ,θ,ψ se conocen como ángulos de Euler. Estos ángulos definen una rotación mediante rotaciones de tres ejes, es decir se transforma un sistema de ejes cartesianos en otro.

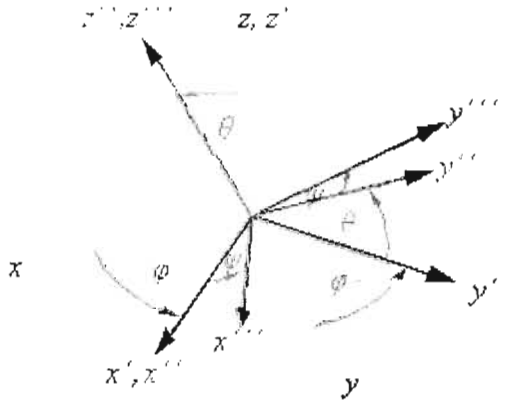


Figura 2-1 Rotaciones de Euler

En la **figura 2-2**, se puede ver como primero se rotaría el sobre el eje z (φ), luego sobre el nuevo eje x' (θ) y finalmente sobre el nuevo eje z" (ψ).

Las rotaciones de Euler reciben los nombres de *nutación, precesión* y *rotación propia* .

La posición de una molécula estará dada por las coordenadas cartesianas x,y,z de su centro de masa y su orientación en base a sus ángulos de Euler.

Para efectos computacionales, en lugar de los ángulos de Euler son utilizados los cuaterniones para indicar la orientación de la molécula por las siguientes razones:

1.- En el transcurso de la simulación eventualmente se tendrán que resolver ecuaciones de movimiento para la rotación donde pudiera aparecer el término $\frac{1}{sen\theta}$, entonces cuando $\theta = 0$, ocurriría una indeterminación que produciría un error en tiempo de ejecución que finalizaría el programa.

2.- Estabilidad numérica de la simulación.
3.- Es posible expresar las operaciones en álgebra de cuaterniones.

Un cuaternion es un conjunto de cuatro valores escalares

$$Q = (q_0, q_1, q_2, q_3) \tag{2.43}$$

y se obtienen a partir de los ángulos de Euler [REFSON 2001] mediante:

$$\begin{aligned} q_0 &= \cos\frac{\phi+\psi}{2}\cos\frac{\theta}{2} \\ q_1 &= \text{sen}\frac{\phi-\psi}{2}\text{sen}\frac{\theta}{2} \\ q_2 &= \cos\frac{\phi-\psi}{2}\text{sen}\frac{\theta}{2} \\ q_3 &= \text{sen}\frac{\phi+\psi}{2}\cos\frac{\theta}{2} \end{aligned} \tag{2.44}$$

Al construir un cuaternion este debe cumplir con la restricción:

$$q_0^2 + q_1^2 + q_2^2 + q_3^2 = 1 \tag{2.45}$$

Finalmente se obtiene la siguiente matriz de rotación al utilizar cuaterniones [REFSON 2001]:

$$A = \begin{pmatrix} q_0^2+q_1^2-q_2^2-q_3^2 & 2(q_1q_2+q_0q_3) & 2(q_1q_3-q_0q_2) \\ 2(q_1q_2-q_0q_3) & q_0^2-q_1^2+q_2^2-q_3^2 & 2(q_2q_3+q_0q_1) \\ 2(q_1q_3+q_0q_2) & 2(q_2q_3-q_0q_1) & q_0^2-q_1^2-q_2^2+q_3^2 \end{pmatrix} \tag{2.46}$$

2.1.5 Descripción del Algoritmo de Rotación

Para calcular las orientaciones de las moléculas a lo largo de la simulación, es necesario la obtención de un conjunto de valores que a su vez permitirán la obtención de otros más. Los valores obtenidos al final de procedimiento, darán la posición de los centros de masa ,los quaterniones, así como velocidades angulares (rotación) y de centros de masa (traslación) en t+Δt, completándose así un paso de la simulación.

1. Al tiempo t se calcula la torca τ', en cada molécula utilizando el sistema de ejes fijo de la molécula, mediante:
   $$\tau'(t) = \sum_i r_i x F_i \qquad (2.47)$$
   donde F_i es la fuerza que actúa en el sitio i de la molécula, r_i es el vector que va del centro e masa de la molécula al sitio i.

2. Se calcula la torca en el sistema de ejes fijo de la simulación mediante la matriz transpuesta
   $$\tau(t) = \tilde{A}\tau'(t) \qquad (2.48)$$

3. En $t - \frac{1}{2}\Delta t$ se conocen las velocidades angulares $w'(t - \frac{1}{2}\Delta t)$ permitiendo el cálculo del momento angular L' mediante:
   $$L'(t - \tfrac{1}{2}\Delta t) = \sum_i m_i \left(r_i x \left(w' x r_i\right)\right) \qquad (2.49)$$
   para calcular el momento angular en el sistema de ejes fijo del sistema se emplea:
   $$L(t - \tfrac{1}{2}\Delta t) = \tilde{A}L'(t - \tfrac{1}{2}\Delta t) \qquad (2.50)$$

4. Ahora se calcula el momento angular L para el tiempo t=0 mediante:
   $$L(t) = L(t - \tfrac{1}{2}\Delta t) + \tfrac{1}{2}\tau(t)\Delta t \qquad (2.51)$$

5. Para obtener el mismo valor en el sistema de ejes fijos de la molécula se aplica:
   $$L'(t) = A\tau L(t) \qquad (2.52)$$
   este último valor permite conocer las componentes de las velocidades angulares w'_x, w'_y, w'_z
6. Los valores w'_x, w'_y, w'_z, hacen posible la obtención de las derivadas temporales de los cuaterniones al tiempo t.

$$\dot{Q}(t)=\begin{pmatrix}\dot{q}_0\\ \dot{q}_1\\ \dot{q}_2\\ \dot{q}_3\end{pmatrix}=\frac{1}{2}\begin{pmatrix}q_0 & -q_1 & -q_2 & -q_3\\ q_1 & q_0 & -q_3 & q_2\\ q_2 & q_3 & q_0 & -q_1\\ q_3 & -q_2 & q_1 & q_0\end{pmatrix}\begin{pmatrix}0\\ w'_x\\ w'_y\\ w'_z\end{pmatrix} \tag{2.53}$$

7. Para obtener los cuaterniones al tiempo $t+\Delta t$ se utiliza la formula:
$$Q(t+\tfrac{1}{2}\Delta t)=Q(t)+\dot{Q}(t+\tfrac{1}{2}\Delta t)dt \tag{2.54}$$
8. Ahora se requiere de $t+\Delta t$ mediante
$$L(t+\tfrac{1}{2}\Delta t)=L(t-\tfrac{1}{2}\Delta t)+\tau(t)\Delta t \tag{2.55}$$
este mismo valore es obtenido ahora para el sistema de ejes fijo de la molécula
$$L'(t+\tfrac{1}{2}\Delta t)=AL(t+\tfrac{1}{2}\Delta t) \tag{2.56}$$
al igual que en el paso 6. podemos obtener w'_x, w'_y, w'_z para $t+\frac{1}{2}\Delta t$, para poder calcular las derivadas temporales de los cuaterniones en $Q(t+\tfrac{1}{2}\Delta t)$

$$\dot{Q}(t+\tfrac{1}{2}\Delta t)=\begin{pmatrix}\dot{q}_0\\ \dot{q}_1\\ \dot{q}_2\\ \dot{q}_3\end{pmatrix}=\frac{1}{2}\begin{pmatrix}q_0 & -q_1 & -q_2 & -q_3\\ q_1 & q_0 & -q_3 & q_2\\ q_2 & q_3 & q_0 & -q_1\\ q_3 & -q_2 & q_1 & q_0\end{pmatrix}\begin{pmatrix}0\\ w'_x\\ w'_y\\ w'_z\end{pmatrix} \tag{2.57}$$

a diferencia del paso 6, aquí se utilizan w'_x, w'_y, w'_z y los quaterniones en el tiempo $t+\frac{1}{2}\Delta t$.

9. Se obtienen los cuaterniones en $t+\Delta t$
$$Q(t+\Delta t)=Q(t)+\dot{Q}(t+\tfrac{1}{2}\Delta t)\Delta t \tag{2.58}$$

10. Se calculan los cuaterniones al tiempo $t+\frac{1}{2}\Delta t$
$$Q(t+\tfrac{1}{2}\Delta t)=Q(t)+\dot{Q}(t+\tfrac{1}{2}\Delta t)dt \tag{2.59}$$

2.1.6 Condiciones periódicas de Frontera

El problema de efecto de superficie puede ser resuelto implementando condiciones periódicas de frontera [ALLEN 1987]. Una caja tridimensional es replicada a través del espacio de manera que al moverse una molécula en su cubo original, su imagen periódica en cada uno de las cajas vecinas se mueve

exactamente igual. De este modo, cuando una molécula deja la caja de la simulación, una de sus imágenes deberá entrar por la cara opuesta de la caja.

Utilizando esta técnica, bastará con simular lo que puede denominarse como la caja central. En esta caja se llevará el control de la posición de las moléculas que la conforman, al rebasar cualquiera de estas partículas los limites de la caja, su imagen periódica reingresara por la cara contraria. Véase **figura 2-2**.

La validez de la simulación al representar solamente un sistema limitado que se supone simula un sistema infinitamente mayor, así como el volumen de la caja y el número de partículas a simular dependerá del rango de potencial intermolecular y el fenómeno de la investigación. [ALLEN 1987].

La principal ventaja al realizar la simulación de este modo radica en el número finito de cálculos necesarios y por tanto, el tiempo de ejecución y recursos empleados por el algoritmo.

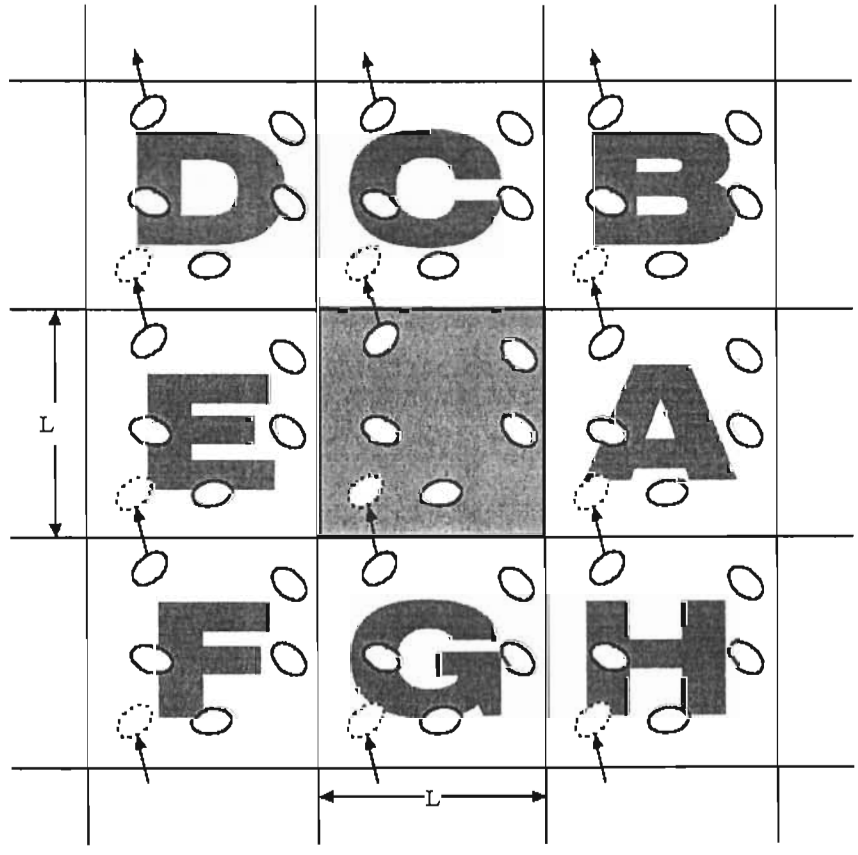


Figura 2-2. Condiciones periódicas de frontera.

El uso de cubos para las cajas de simulación es común, aunque otras formas también son empleadas, entre ellas se encuentran el dodecaedro rómbico y el octaedro truncado[ALLEN 1987].

Al calcular las fuerzas que interactúan entre todas las partículas que integran una caja, es necesario también calcular las fuerzas con respecto a las supuestas moléculas en las cajas vecinas, las cuales son infinitas. En principio, lo que se hace, es tomar en cuenta solamente las cajas vecinas inmediatas. Aun de esta manera, el número de interacciones llega a ser muy grande y su calculo muy costoso; por tal motivo, se implementa el uso de distancias de corte.

Las distancias de corte, delimitan un radio que forma una esfera. Véase **figura 2-3**. Se establecen tantas esferas como moléculas existan en el sistema. Se considera que no existe una interacción directa entre la molécula al centro de la esfera y las moléculas que se encuentran fuera de ella.

La distancia de corte no deberá ser mayor a ½ L .Donde L es el largo de las caras de la caja de simulación. La consecuencia de la utilización de distancias de corte y la consiguientes zonas esféricas que estas implican es la precisión del modelo.

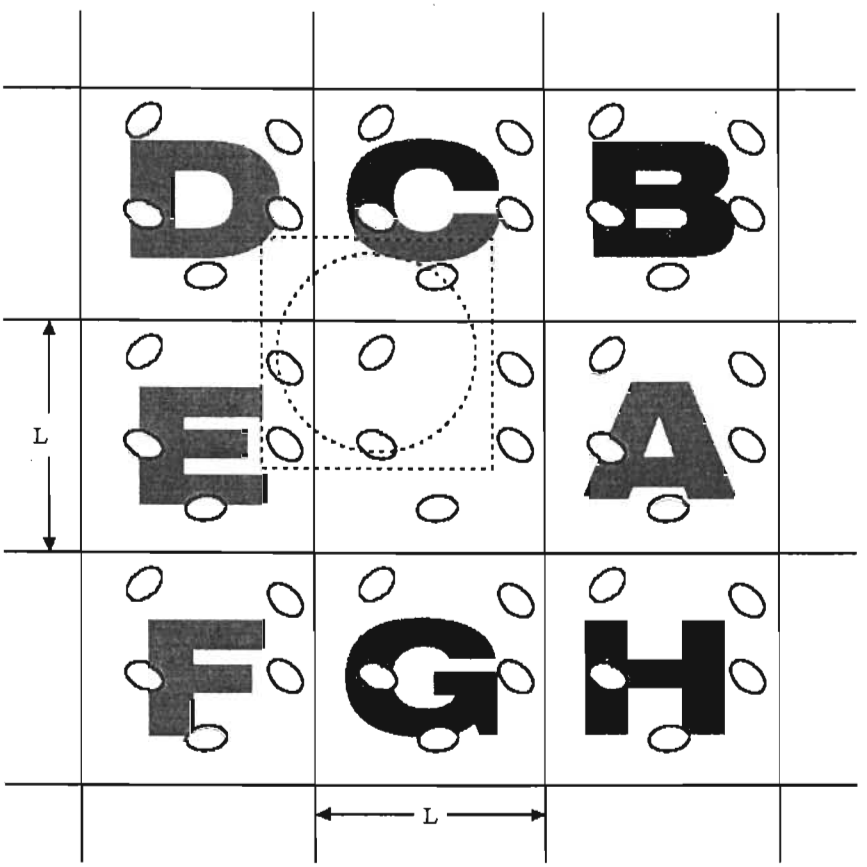


Figura 2-3. Distancias de corte.

2.1.7 Propiedades Macroscópicas

Los procedimientos enunciados en los puntos anteriores, permiten la obtención de valores sobre las moléculas de manera individual. Esta información se encuentra dentro de las **propiedades microscópicas** o individuales. A partir de las propiedades microscópicas, es posible la obtención de propiedades o valores referentes a la simulación completa, dicha información se conoce como **propiedades macroscópicas**.

Entre las propiedades macroscópicas que se obtienen en el sistema, se encuentran valores termodinámicos como la presión y la temperatura. Estas propiedades son graficadas durante el tiempo de ejecución de la simulación, y son obtenidas en cada paso $t + \Delta t$ de la simulación.

Las propiedades macroscópicas termodinámicas, requieren el cálculo de energía del sistema. La energía total del sistema E se calcula como la suma de la energía

cinética $K(p^N)$ y la energía potencial $U(r^N)$ [BALTAZAR 2003], siendo **p** el momento de la partícula N, y **r** su posición.

$$E = K(p^N) + U(r^N) \tag{2.60}$$

En el caso de los sistemas moleculares rígidos como los empleados en este trabajo, la energía cinética $K(p^N)$ se calcula como una suma de la energía cinética de traslación y energía cinética de rotación.

$$K = K_{TRASL} + K_{ROT} \tag{2.61}$$

La temperatura del sistema es obtenida basándose en el promedio de la energía cinética del sistema $\langle K \rangle$ mediante[GUTIERREZ 2001]:

$$\langle K \rangle = \frac{3}{2} N k_B T \tag{2.62}$$

donde:
N = Número de Moléculas
k_B = Constante de Boltzman que equivale a 1.38062×10^{-23} J/K
T = Temperatura

La energía potencial es obtenida mediante:

$$U(r^N) = \sum_{i=1}^{N} \sum_{j<i}^{N} U(r_{ij}) \tag{2.63}$$

donde $U(r_{ij})$ constituye la energía potencial entre cada uno de los pares de moléculas, tal energía, se obtiene en función a la fuerza de interacción entre la molécula i y la molécula j, la distancia entre ellas r_{ij} . El valor de energía potencial se obtiene:

$$U(r_{ij}) = F_{ij} \bullet r_{ij} \tag{2.64}$$

El cálculo de la presión emplea la siguiente formula[BALTAZAR 2003]:

$$P = \frac{2N}{3V} \langle K \rangle + \frac{1}{3V} \left\langle \sum_{i<j} F_{ij} \bullet r_{ij} \right\rangle \tag{2.65}$$

Donde F_{ij} es la fuerza que ejerce la molécula j sobre la molécula i, como si esta fuerza actuara sobre el centro de masa de i. r_{ij} se refiere a la distancia entre la molécula i y j.

2.2 Complejidad

El calculo de las fuerzas es la parte más costosa en términos de tiempo de computo en una simulación. De hecho, el tiempo que se ocupa en integrar las ecuaciones es casi despreciable al lado de éste. Para un sistema de N partículas, evaluar en forma directa la interacción de dos cuerpos requiere $O(N^2)$ de operaciones. Al emplear mecanismos como las distancias de corte no se calculan fuerzas para todo el sistema de N-1 partículas con respecto a cada molécula, sino para un número menor.

Otra mejora se consigue al utilizar otros mecanismos como las **listas de vecinos,** que permiten hacer más eficiente la técnica de distancia de corte. Si bien, al reducir el número de partículas se obtiene una reducción drástica de tiempo de computo, es necesario calcular cuales de las N-1 partículas del modelo se encuentran dentro o fuera del área de corte r_c. Esto trae como resultado una complejidad de $O(N)^2$ [ALLEN 1987] en este proceso.

El método de lista de vecinos consiste en determinar las partículas dentro de las distancias de corte r_c para cada partícula al iniciar la simulación, y las partículas en una distancia de corte mayor r_s y almacenarla en una lista Li. Véase **figura 2-4**.

Dependiendo de la naturaleza del sistema, se puede suponer que esta lista no variará considerablemente de un paso a otro de la simulación, por lo cual, la lista de vecinos se empleará para varios pasos antes de ser renovada. En cada paso de la simulación, en lugar de evaluar cuantas de las N-1 partículas se encuentran dentro de r_c, se realiza el calculo solamente para las partículas dentro del área de corte r_s almacenadas en Li, esto puede implicar que ciertas partículas de Li entren o salgan del área r_c en varios pasos antes de que Li sea reelaborada.

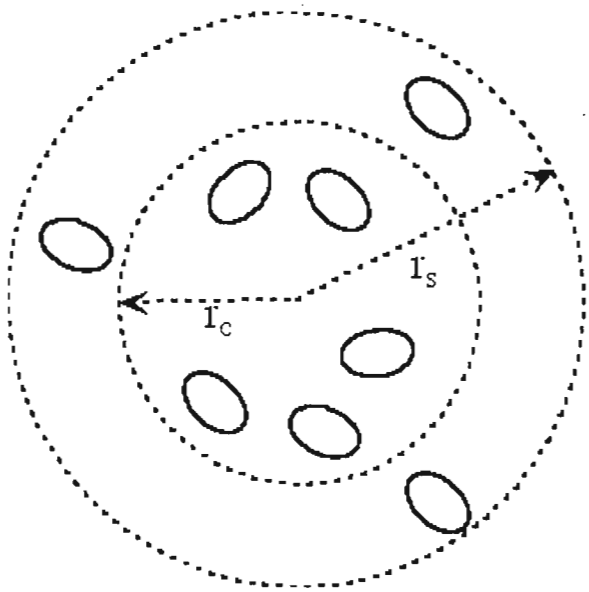


Figura 2-4. Radios de corte y radios para generar listas de vecinos.

Esta reducción de tiempo es muy importante, ya que se estima que el 95% del tiempo de computo de una simulación de Dinámica Molecular típica es empleado en calcular distancias para el conjunto de los pares de partículas, determinar cuales se encuentran dentro de r_c, y calcular las fuerzas para este subconjunto[ALLEN 1987].

El inconveniente principal de este método, es el aumento de la complejidad de programación y la necesidad de espacio de almacenamiento, que pudiese complicar el modelo computacional si el número de partículas es muy grande. Si el número de partículas es reducido, el costo de los cálculos de las listas y su almacenamiento y recuperación disminuye la ganancia en tiempo computacional.

CAPÍTULO 3. OpenGL

OpenGL es una interface de software de alto rendimiento para el manejo del hardware de gráficos independiente del sistema de ventanas empleado. OpenGL está constituida por una biblioteca de modelado y graficación de 3D. Esta biblioteca se caracteriza por su nivel de optimización debido a sus algoritmos.

Estas características hacen que OpenGL sea adecuada para su empleo en la presente investigación. La función de OpenGL en este trabajo, es permitir la visualización y manipulación de los modelos tridimensionales generados por los cálculos de la Dinámica Molecular. Debido al número de partículas de la simulación, es necesaria una interface gráfica que permita su representación , haciendo uso directo de los recursos de hardware gráfico. Permitiendo la liberación de recursos de procesamiento central para aprovecharse en los cálculos de Dinámica Molecular.

OpenGL también es empleada para la graficación de propiedades macroscópicas que permiten determinar el estado de la simulación. De gran importancia debido al número de pasos de una simulación.

3.1 Implementaciones

OpenGL tiene sus orígenes en IRIS GL desarrollado por Silicon Graphics. IRIS GL es una API diseñada y optimizada para el despliegue y manipulación de imágenes 3D en las estaciones de trabajo IRIS, equipos diseñados para el trabajo con gráficos por tener hardware especializado, que entre otras características, permitía la realización de transformaciones matriciales de manera rápida y eficiente.

OpenGL es el resultado del trabajo de Silicon Graphics para exportar IRIS GL a otras plataformas de hardware e impulsar así el mercado de la industria del computo en aplicaciones gráficas. Con la idea de convertir a OpenGL en un estándar abierto, se creo la *OpenGL Architecture Review Board* (ARB), contando entre sus miembros fundadores con empresas como Digital Equipment, IBM, Intel, Microsoft y Silicon Graphics, además de permitir la participación de otras empresas en sus reuniones. En Julio de 1992 la ARB lanza la primera especificación de OpenGL. ARB controla el desarrollo de OpenGL, aunque Silicon Graphics sigue manejando la concesión de licencias de OpenGL a los diferentes fabricantes de software y hardware. Bajo el esquema actual de

licencias de Silicon Graphics "los desarrolladores de software no necesitan licencia, solamente los fabricantes de hardware"[1]. Entre los principales miembros de la ARB actual figuran: 3Dlabs, Apple, ATI, Dell Computer, Evans & Sutherland, Hewlett-Packard, IBM, Intel, Matrox, NVIDIA, Microsoft, Silicon Graphics y Sun Microsystems.

Como resultado del trabajo de la ARB surgieron varias implementaciones de OpenGL, agrupándose en dos tipos, las enfocadas al software(Bibliotecas de programación o APIs) y las de hardware (véase **figura 3-1**) en forma de controladores de dispositivos de hardware generalmente tarjetas de video.

Existen diversas implementaciones de OpenGL sobre varias plataformas de hardware y diferentes sistemas operativos.

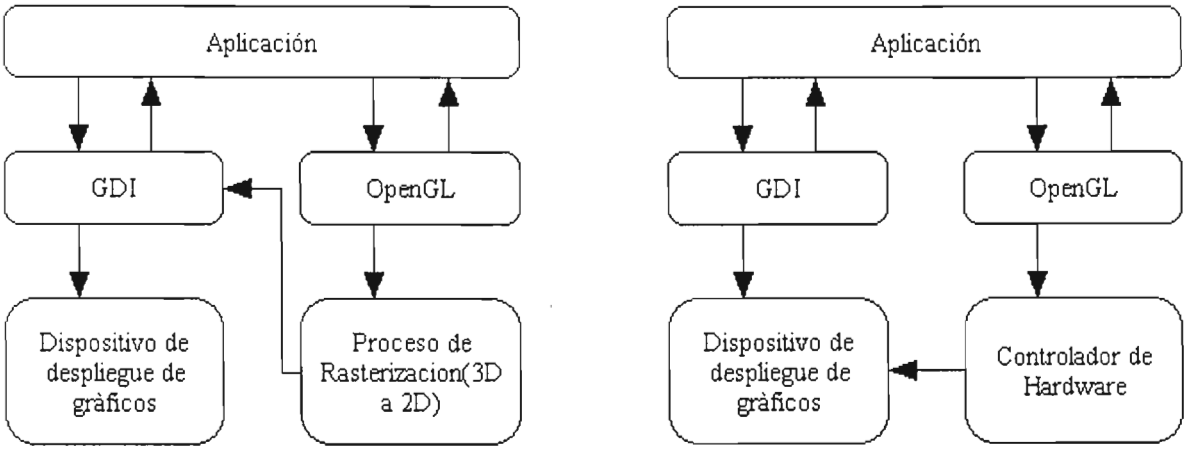


Figura 3-1. Implementaciónes OpenGL.

Actualmente OpenGL con sus más de 250 comandos[WOO 1999], constituye uno de los estándares de graficación 3D de la industria, aunque cabe mencionar que existen otros estándares como Farenheit (Silicon Graphics y Microsoft 1997) y Direct 3D (Microsoft), los cuales, están atados a alguna plataforma de software especifica o su desarrollo es controlado por una sola empresa o un conjunto reducido de las mismas, en algunos casos mediante estándares no abiertos.

En el caso del Sistema Operativo Windows de Microsoft, OpenGL 1.1 se incluye en sus versiones desde Windows NT 4.0 y Windows 95 OSR2. Las

[1] http://www.sgi.com/software/opengl

versiones 1.2 y 1.3 se obtienen junto con los controladores de la tarjeta de video proporcionados por los fabricantes de la misma.

Linux por ser un sistema operativo no solamente de código abierto, sino además de uso libre y conformado por la comunidad internacional, no cuenta con una licencia de OpenGL como tal, aunque existe MESA3D una API desarrollada por Brian Paul[WRIGHT 1999] .

MESA 3D es una API que trabaja de manera similar a OpenGL y de alguna manera, pudiera ser considerada como una implementacion "no oficial" de OpenGL, MESA fue lanzada con consentimiento de OpenGL, aunque no utiliza nombres registrados de OpenGL; sin embargo, cumple con las pruebas de conformidad exigidas por OpenGL para las implementaciones oficiales. Es de esperarse, que un programa que utiliza una implementacion oficial de OpenGL pueda trabajar sin problemas con MESA, sin tener que modificar el código fuente. MESA así como su código fuente pueden ser obtenidos y distribuidos de manera libre, incluso se incluyen dentro de las principales distribuciones actuales de Linux bajo el acuerdo de licencia GPL ("General Public Licence").

3.2 Caracterisiticas de OpenGL

Entre las principales elementos de OGL se tienen las siguientes:

Primitivas Geométricas: Permiten construir descripciones matemáticas de objetos, a partir de elementos simples. Las primitivas son puntos, líneas y polígonos.

Primitivas de *Raster* en forma de *bitmaps* y rectángulos de píxeles.

Color en modo RGBA. Los *buffers* de color permiten almacenar valores distintos para el rojo, verde, azul y Alfa, este último es un valor de transparencia.

Color en modo Indexado. Permite trabajar con paletas de colores.

Mapeado de texturas, proceso que consiste en aplicar una imagen a una primitiva grafica. Esta técnica es utilizada para dar realismo a las imágenes.

Transformación, capacidad de rotar, cambiar el tamaño y la perspectiva de un objeto en el espacio tridimensional.

Z-buffer, este *buffer* mantiene registros de la coordenada Z de un objeto 3D, permitiendo representar de manera correcta la proximidad de un objeto al observador y eliminar las superficies ocultas.

Listas de despliegue que permiten almacenar comandos de dibujo en una lista para un trazado posterior, incrementando el rendimiento.

Evaluadores polinómicos que dan soporte a B-splines racionales no uniformes (NURBS).

Simulación de **efectos atmosféricos** como niebla, bruma, humo o polución.

Antialisado. Un método de renderización utilizado para suavizar líneas y curvas. Esta técnica promedia el color de los pixeles adyacentes a la línea, reduciendo el efecto de bordes escalonados.

Doble buffer, mediante este buffer se puede trabajar la construcción de imágenes en segundo plano y ser mostradas al estar terminadas.

Sombreado de Gouraud consiste en la interpolación progresiva de colores a través de un polígono o segmento de línea. Los colores a interpolar se asignan a los vértices y son interpolados linealmente a lo largo de la primitiva para producir una variación relativamente suave de color.

3.3 Funcionamiento de OpenGL

OpenGL es una máquina de estado. Al momento de su inicialización, son creadas en memoria una serie de variables con valores por omisión que determinarán la forma de trabajo de OpenGL. Por ejemplo, encender o apagar la forma en que se mostrarán los objetos de tres dimensiones, si aparecerán como sólidos o mediante estructuras de alambre. Las variables de estado pueden ser modificadas durante la ejecución de la aplicación.

De forma resumida la generación de gráficos con OpenGL tiene los siguientes pasos:

1. Al momento de desarrollar la aplicación, el programador deberá incluir la invocación a las bibliotecas de OpenGL, y su inicialización.
2. En tiempo de ejecución, las llamadas a OpenGL son almacenadas en el buffer de comandos. Estas instrucciones serán procesadas de acuerdo a su contenido, de ser necesario se aplicará un proceso de transformación e iluminación.
3. Los valores obtenidos son utilizados en el proceso de rasterización que consiste básicamente en la generación de imágenes 2D, que pueden ser desplegadas en el dispositivo gráfico.
4. Las imágenes obtenidas son almacenadas en la memoria del hardware de manejo de gráficos "*frame buffer*" listas para su despliegue en el dispositivo de visualización de gráficos del sistema.

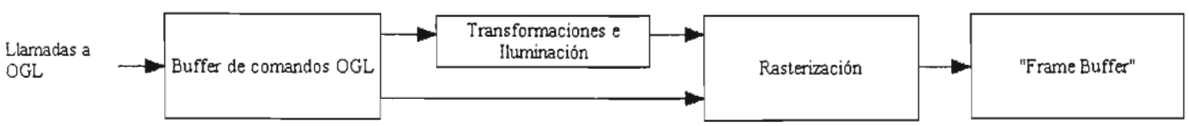


Figura 3-2 Diagrama de bloques resumido de operación de OpenGL

3.3.1 Sistemas de coordenadas y área de visualización

Antes de comenzar a graficar con OpenGL es necesario determinar el sistema de coordenadas a utilizar y como traducir coordenadas especificas en unidades de dibujo en pantalla.

Para tal propósito se dispone de tres zonas (véase figura 3-3).

- ✓ Volumen de recorte.
- ✓ Ventana cliente.
- ✓ Área de visualización o "*viewport*".

El **volumen de recorte** ("*clipping* volumen") está definido por los valores mínimos y máximos de X, Y y Z del espacio cartesiano que se desee visualizar en el modelo, si algún elemento rebasa esta área no será mostrado en pantalla.

La **ventana del cliente**, es la ventana gráfica destinada por la aplicación para el despliegue del gráfico OpenGL, la cual, deberá ser creada con un gestor de manejo de ventanas adicional.

El **área de visualización** o *viewport* es la región dentro de la ventana del cliente que se emplea para mostrar el volumen de recorte.

Tanto el *viewport* como el área de la ventana del cliente, son bidimensionales y por lo general dadas en píxeles y valores positivos. Al momento de hacer el ajuste entre el *viewport* y el volumen de recorte, son considerados solamente los valores mínimos y máximos de X e Y del volumen de recorte. El área resultante se le denomina "área de recorte".

La ventana del cliente y el área de visualización pueden tener tamaños distintos, en tal caso, OpenGL y el gestor adicional de manejo de ventanas ajustaran la imagen ampliando o reduciendo los gráficos trazados o mostrando solamente una parte de ellos.

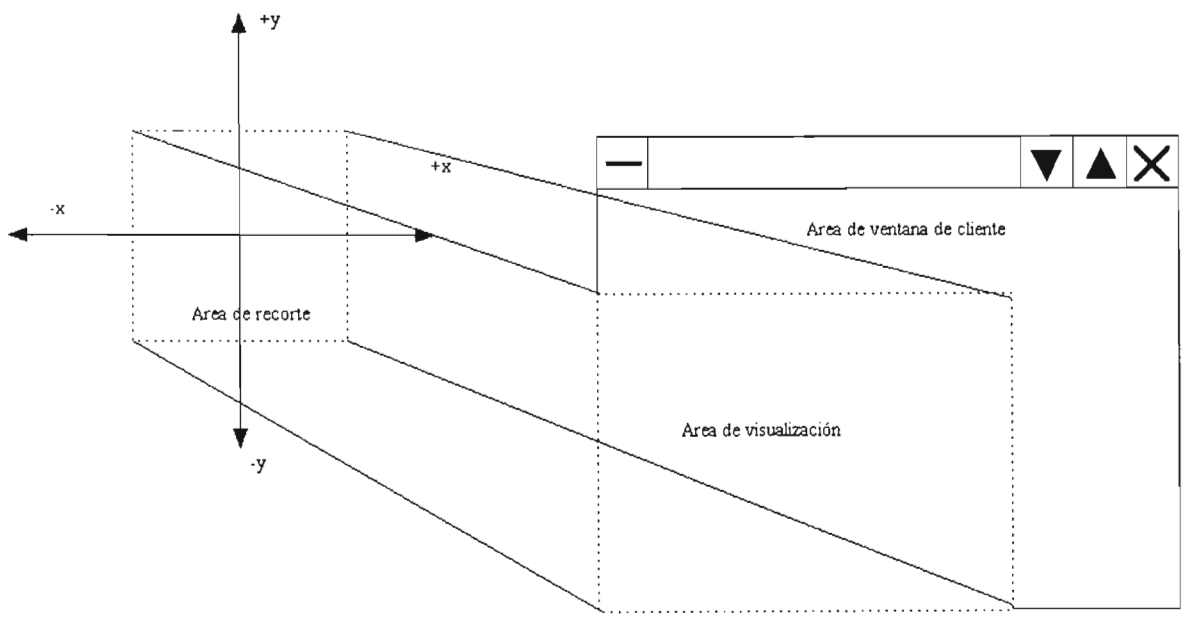


Figura 3-3 Áreas de visualización.

3.3.2 Transformaciones

Una vez creadas las primitivas que conformarán elementos de dibujo más complejos, será necesario plasmarlos sobre el espacio bidimensional de la pantalla y en muchas ocasiones realizar algunos cambios sobre el modelo

obtenido como rotaciones, cambio de tamaño, cambios de posición con respecto al espectador (movimiento de cámara), etc.

Para generar los nuevos gráficos, OpenGL se vale principalmente del álgebra lineal manipulando las coordenadas originales del modelo como matrices, al aplicar a dichas matrices transformaciones obtiene sus equivalentes en otro sistema de coordenadas, que al ser graficadas producen el efecto deseado..

La manera en que cada uno de los vértices del modelo original es procesado es la siguiente(figura 3-4):

1.- Cada uno de los vértices de las primitivas del modelo a graficar es definido por 4 valores, los cuales, constituirán matrices de 4x1. Donde x, y, y z son coordenadas cartesianas y w es un factor de escala que generalmente tiene un valor de 1 (Sistema homogéneo de coordenadas).

2.- La matriz de modelado o "modelview" se multiplica por cada una de las matrices correspondientes a los vértices obtenidos en el paso uno, las matrices resultantes constituyen las coordenadas de visualización "eye coordinates"

3.- La matriz de proyección es multiplicada por las matrices obtenidas en el paso 2 dando como resultado coordenadas de recorte "clipping", esta operación elimina los datos fuera del área de recorte.

4.- Las coordenadas de recorte son divididas por el valor w, obteniéndose coordenadas de dispositivo normalizadas.

5.- Finalmente las coordenadas normalizadas son procesadas para obtener las coordenadas para en el dispositivo de despliegue "coordenadas de ventana" que dependen del área de visualización previamente definida.

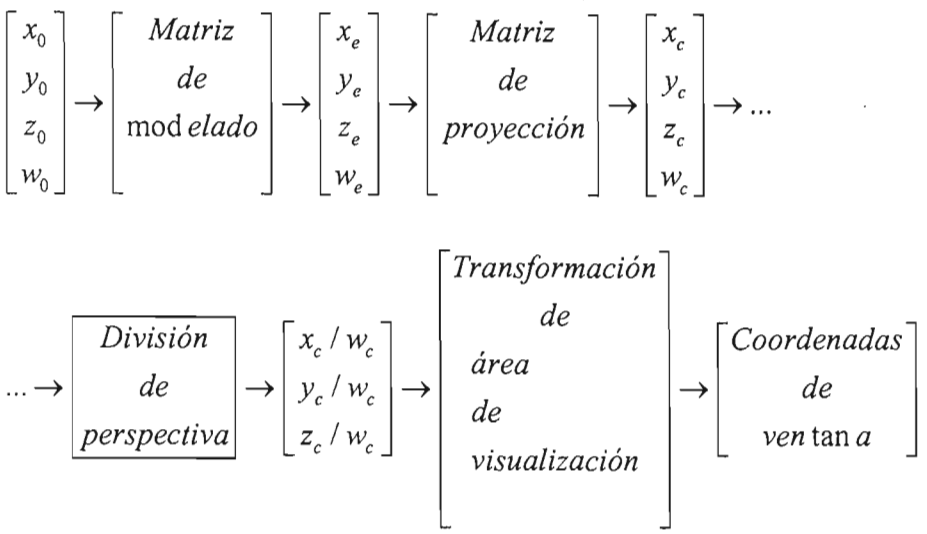


Figura 3-4 Línea de proceso de transformaciones

3.3.3 Transformación del área de visualización

En esta fase, el modelo en tres dimensiones es convertido a uno bidimensional para ser representado en el dispositivo final de visualización. Las coordenadas obtenidas son expresadas como x_n y y_n, para cada uno de los vértices, de acuerdo al área de recorte (cara frontal del volumen de recorte con respecto al punto de visualización). Finalmente estas coordenadas son convertidas en coordenadas de ventana para su despliegue en el área de visualización, x_f y y_f, de acuerdo a la siguientes fórmulas.

$$x_f = (x_n + 1)\frac{Ancho}{2} + x \tag{3.1}$$

$$y_f = (y_n + 1)\frac{Alto}{2} + y \tag{3.2}$$

donde:

x, y. especifican la esquina inferior izquierda del área de visualización en pantalla. (pixeles).
Alto, Ancho. Especifican la altura y ancho del área de visualización. (pixeles)
x_n, y_n. Coordenadas de dispositivo normalizadas.
x_f, y_f. Coordinadas de ventana o finales(pixeles).

3.3.4 Rasterización

En esta parte del proceso, la información es convertida en imágenes representadas por píxeles, combinándose la información obtenida a partir de primitivas geométricas, transformaciones, parámetros y fuentes de iluminación, valores de color, imágenes y textos de mapa de bits, texturas aplicadas a los objetos, efectos atmosféricos, etc. Algunos definidos al momento de programar y otros tomados como valores por defecto de la máquina de estado de OpenGL.

La imagen obtenida es dividida en fragmentos que representan cada uno de los píxeles del "*frame buffer*", constituido por la memoria del adaptador de despliegue de gráficos del equipo. La imagen final es colocada en el *frame buffer* antes de ser desplegada en el dispositivo de visualización. En el caso de utilizarse el modo de buffer simple, la imagen será desplegada a la brevedad posible al llegar al "frame buffer". Si se emplea el modo de *buffer* doble, los datos se almacenaran en memoria, hasta que se emplee el comando de volcado de memoria a pantalla.

3.4 Sistemas de Ventanas y controles

Con la finalidad de que OpenGL fuese independiente de plataforma, se creo únicamente como una interface para la creación de primitivas de dibujo, la cual incluye manejo de profundidad, buffers, iluminación, áreas de visualización, sistemas de color RGBA e indexado, etc. Dada esta independencia, no incluye un sistema de generación de ventanas, ni de manejo de eventos, por lo que es necesaria la utilización de un subsistema gráfico y de generación de ventanas, así como de captura de eventos para la interacción con OpenGL.

Entre los principales podemos mencionar:
GLX: Sistemas X Windows
AGL: Sistemas Apple Machintosh
PGL: Sistemas IBM OS/2 Wrap
WGL: Sistemas Microsoft Windows
GLUT: Multiplataforma.

3.4.1 GLUT y GLUI

Los sistemas mencionados en el punto anterior, tienen el inconveniente de estar sujetos a un sistema operativo en particular. Una de las opciones disponibles es GLUT(OpenGL Utility Toolkit) desarrollado por Mark J. Kilgard de Silicon Graphics Inc. GLUT esta compuesto de una API que permite al manejo de ventanas, captura de eventos, manejo de dispositivos de entrada y realización de operaciones de dibujo complejas de manera resumida.

Permite la creación mediante una sola llamada a función de los siguientes Objetos tridimensionales:

- ✓ Esferas.
- ✓ Cubos
- ✓ Conos
- ✓ Toroides
- ✓ Dodecaedros
- ✓ Octaedros
- ✓ Tetraedros
- ✓ Icosaedros
- ✓ Teteras

Todos estos objetos pueden ser graficados como sólidos o mediante estructuras de alambre.

GLUT esta construido sobre OpenGL bajo la modalidad GPL(General Public License) , su distribución incluye el código fuente de uso libre por lo que puede ser implementado en múltiples plataformas. Puede ser utilizado en programas ANSI C y FORTRAN de manera independiente al sistema de ventanas del Sistema Operativo empleado.

Entre sus principales características destacan:

- Manejo de múltiples ventanas para el rendereado de gráficos OpenGL.
- Captura y procesamiento de eventos.
- Manejo de dispositivos de entrada.
- Rutinas "idle" que son convocadas cuando el usuario no interactúa con el sistema en ejecución y eventos asociados a tiempo.
- Creación de menús.
- Funciones para generación de cuerpos sólidos y de estructuras de alambre.
- Soporte para manejo de mapas de bits.

GLUI es una librería C++ de interface de usuario basada en GLUT, proporciona controles como botones, cajas de selección (*Checkboxes*), botones de radio, listas, etc., a aplicaciones OpenGL. Es independiente al sistema de ventanas, delegando a GLUT las funciones referentes a la creación y manejo de ventanas, así como de dispositivos de entrada.

Entre sus características se destaca:

- Integración completa con GLUT
- Creación y manejo de controles de interface de usuario.
- Los controles pueden generar llamadas a funciones cuando sus valores cambian, permitiendo la simulación de generación de eventos desde código.
- Se puede ligar variables a controles de manera que estos son actualizados automáticamente cuando los valores de las variables asociadas cambien.
- Los controles pueden lanzar directamente llamadas a funciones de refresco GLUT.
- El ajuste de tamaño de los controles se realiza de manera automática al redimensionar las ventanas que los contienen.
- Se puede ir de un control a otro y accionarlos mediante el teclado.

Al igual que GLUT, GLUI se distribuye bajo el acuerdo de licencia GPL, permitiendo su integración en multiplataforma, en programas en C++.

Como se menciono anteriormente GLUI esta construido sobre GLUT, y GLUT sobre OpenGL(véase **figura 3-5)** , por lo que es necesario hacer llamadas a las API de GLUT y OpenGL para poder utilizar GLUT, y a las API GLUI, GLUT y OpenGL para poder utilizar GLUI. Una vez invocadas las 3 APIs, se pueden hacer llamadas directas a cada una de ellas.

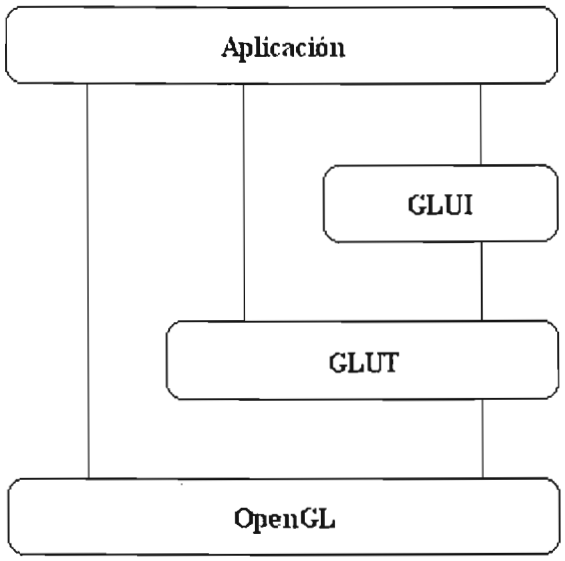


Figura 3-5. Integración OpenGL,GLUT y GLUI.

Las siguientes características gráficas fueron empleadas en la elaboración del sistema:

Característica o modo	Valor
Proyección	Ortogonal 3D.
Modo de Color	RGBA
Transformaciones	Escalado, rotación y traslación.
Operaciones de Píxel	Manejo de caracteres en 2D.
Buffer de Frame	Doble.
Iluminación	Ambiental, especular y difusa.
Estructuras:	Sólidas y alambre
Cuerpos:	GLUT
Sistema de ventanas e interactividad	GLUT, GLUI

CAPÍTULO 4. Paralelismo y concurrencia.

Un programa concurrente se compone de varios procesos que se ejecutan de manera independiente, pero interactúan entre si. En caso de que el control no solamente pase de un proceso a otro, sino que adicionalmente, dos o más procesos se ejecuten al mismo tiempo se produce paralelismo.

El manejo de concurrencia es importante en este sistema, ya que si se emplease el modelo de programación no concurrente (secuencial) surgirían las siguientes situaciones:

1. La parte de los cálculos de dinámica molecular implica más del 95% del tiempo de ejecución del programa(Este valor se obtuvo al emplear el programa de perfilamiento *gprofile*).Si los cálculos son realizados antes de la invocación del sistema gráfico, solamente se podrían visualizar al final de la simulación.

2. El sistema gráfico responde a eventos generados por el usuario. Si las funciones de cálculos son llamadas después de invocar al sistema gráfico, esta parte de código no es alcanzada hasta que se da por terminada la simulación gráfica. Esto debido a que el sistema de eventos genera un bucle en espera de interacción por parte del usuario, por tal motivo, las instrucciones colocadas después de la activación del sistema de eventos no son alcanzadas hasta que se da por terminada la captura de los eventos del usuario.

3. Si las funciones de cálculo son intercaladas con las funciones de graficación, tomando al sistema gráfico como programa principal, se produce un tiempo de respuesta no aceptable del sistema gráfico, adicionalmente, eleva considerablemente el tiempo de ejecución del sistema en general.

4. En caso de que las funciones gráficas fueran intercaladas dentro de las funciones de calculo, esto produciría una regeneración del sistema gráfico en cada paso. Si se establecieran en intervalos o número fijo de pasos, esto le restaría tiempo de respuesta a la interacción con el usuario; adicionalmente, se consumirían una cantidad considerable de recursos aumentando significativamente el tiempo de ejecución del sistema.

4.1 Ejecución secuencial, concurrente y paralela.

Un programa secuencial consta de un conjunto de instrucciones que son ejecutadas en un orden predefinido una a la vez. Dentro de un programa secuencial se cuenta con solamente un hilo o *thread* de control. De tal forma que, todas las llamadas a rutinas del programa son manejadas por él.

Un proceso es definido como la instancia de ejecución de un programa y también como la unidad básica de ejecución del sistema operativo[WALL2000]. Los procesos están conformados por los siguientes elementos(véase **figura 4-1**):

- Código
- Datos
- Contexto

El código y los datos están compuestos por las instrucciones del programa, los datos que maneja, el montículo(heap), librerías compartidas y el espacio de la pila que ocupa. Mientras que el contexto a su vez se divide en:

4.1.1 Contexto de programa

Reside en el procesador e incluye el contenido de los registros de propósito general, registros de códigos de condición, el puntero a la pila (SP) y el contador de programa (PC).

4.1.2 Contexto de kernel.

Reside en el kernel, y está conformado por el identificador del proceso (PID), las estructuras que caracterizan la organización de la memoria virtual, información sobre los archivos abiertos, manejadores de señal, y el apuntador de final (break pointer).

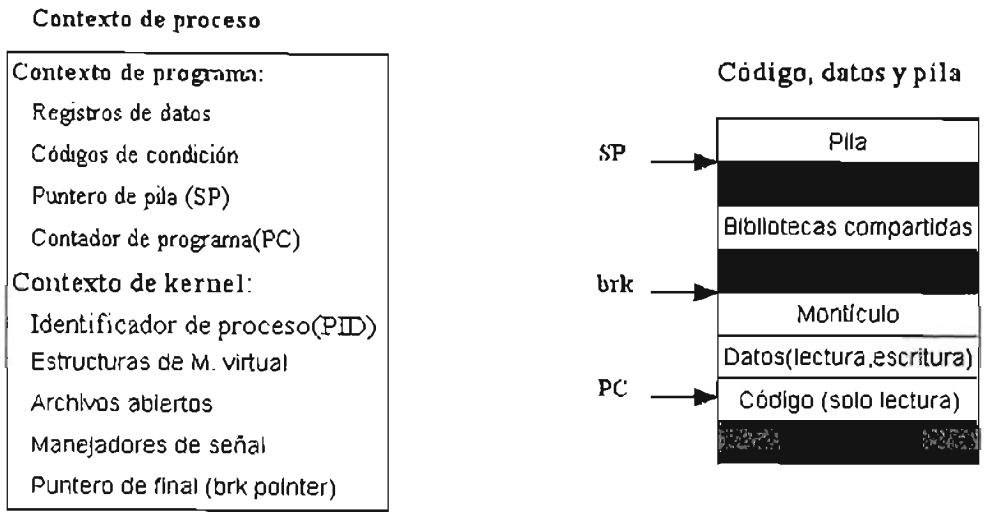


Figura 4-1. Componentes de un proceso

4.2 Ejecución concurrente

La ejecución concurrente se produce cuando un programa se integra por un conjunto de procesos que se ejecutan de manera independiente pero que interactúan entre sí para lograr un fin común.

Esto da lugar a tres escenarios principales:

1.- Los procesos del programa concurrente se ejecutan en un único procesador.

2.- Los procesos se ejecutan en más de un procesador dentro de un mismo equipo. Por lo general comparten un espacio de memoria "Multiproceso".

3.-Los procesos se llevan acabo en equipos independientes interconectados por redes de comunicaciones, a lo que se denomina procesamiento distribuido.

En los casos 2 y 3 se puede llegar a presentar paralelismo, cuando 2 o más procesos se ejecutan de manera simultanea en 2 o más procesadores.

En el caso 1 no se produce un paralelismo, a menos que el procesador empleado sea superescalar, es decir que contenga más de una Unidad Aritmético Lógica "ALU" [GARCIA 2003]. Sin embargo, aunque no se presente paralelismo se puede obtener una mayor eficiencia que al utilizar programación secuencial, debido a la disminución de tiempos ociosos del procesador.

El problema principal de eficiencia con los procesadores superescalares, consiste en que las instrucciones que se reciben pueden llegar a tener una fuerte dependencia entre ellas. Existen varias técnicas que permiten disminuir este efecto, algunas implementadas desde el diseño mismo del procesador y otras en el compilador o compiladores utilizados.

Tratándose de optimización desde el procesador, este lee las instrucciones de manera secuencial, decide qué instrucciones pueden ser ejecutadas en un mismo ciclo de reloj y las envía a las unidades funcionales correspondientes para ejecutarse[CATALAN 2003].

La optimación desde el compilador dependerá del nivel de optimación que se le solicite, por lo regular los niveles de optimación altos generan código objeto o ejecutable de mayor tamaño[WALL2000].

En el caso de sistemas basados en procesos(más adelante se definirán sistemas que permiten el manejo de hilos), la mayoría de los sistemas operativos emplean el modelo jerárquico de creación de procesos. En este modelo, dentro de cada proceso pueden crearse procesos hijos, los cuales, al momento de ser creados son una copia del proceso padre, con un número distinto de PID(*Process Identificator*) , y PPID(*Parent Process Identificator*).EL PPID constituye un elemento más del contexto del proceso y está conformado por el PID del proceso padre. Un proceso solamente puede existir si su proceso padre existe. Si un proceso padre termina, todos sus procesos hijos terminarán.

4.2.1 Estados de un proceso

Un proceso puede estar en alguno de los siguientes estados [NORTON 1996] durante su ciclo de vida:

1. **Creación**: En este estado, el kernel crea el contexto para el nuevo proceso hijo. Durante la creación del proceso, el proceso padre debe estar en estado Activo.
2. **Ejecutable**: El proceso esta listo para ser ejecutado, es decir pasar a estado Activo. Generalmente se coloca en una cola de ejecución.
3. **Activo**. El proceso esta siendo ejecutado.
4. **Dormido**. El proceso se encuentra en pausa o bloqueado, ya sea por alguna instrucción, o en espera de la liberación de algún recurso para utilizar.
5. **Detenido**. En este estado, se encuentra en pausa, en espera de una señal de continuar, sin mas razón que la espera de dicha señal.
6. **Terminado**. Se produce cuando se hace una llamada a la función de salida de proceso. Por ejemplo la llamada a la función denominada en algunos sistemas exit().

Una de las transiciones más relevantes en la ejecución concurrente es el cambio entre un proceso y otro, conocida como *process context switch*,(véase **figura 4-2**) consiste básicamente en cambiar el estado de un proceso activo y activar algún otro.

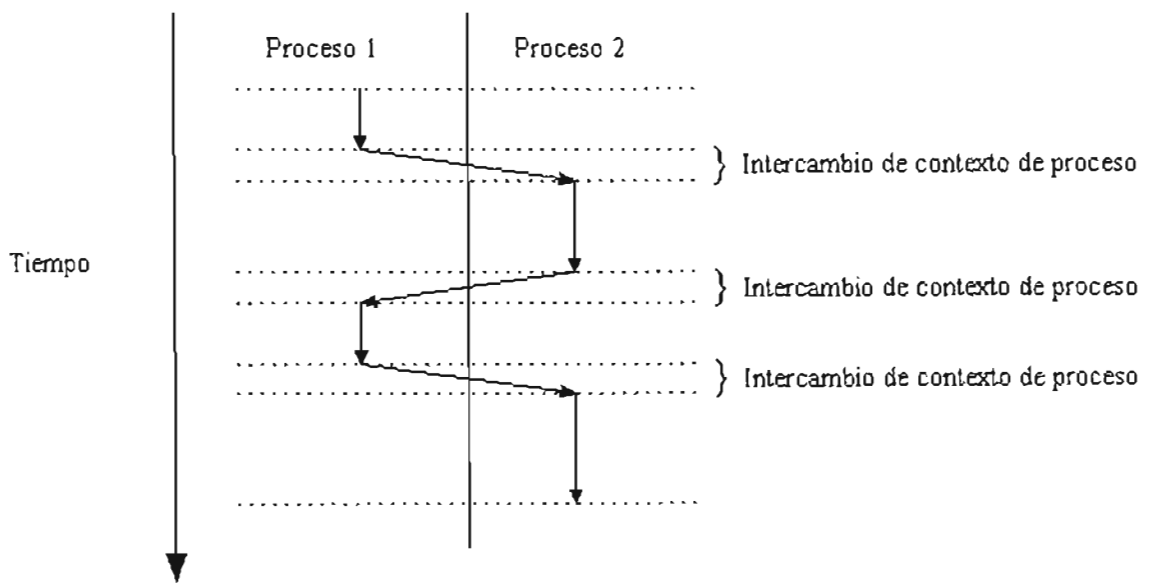


Figura 4-2 Process context switch

4.2.2 Características de la ejecución concurrente

No-determinismo:
A diferencia de la ejecución secuencial, en la ejecución concurrente el orden de las instrucciones no es totalmente predeterminado. Aunque el orden de las instrucciones dentro de cada proceso este dado, el orden de la ejecución final es no-determinista.

Calendarización(*Scheduling*)
La ejecución de las instrucciones que componen un proceso, de entre los diferentes procesos que se encuentran en un estado de listos para ser ejecutados, y la manera en que estas son intercaladas es llevada a cabo por el sistema operativo. Esta caractiristica está muy relacionada con la anterior.

4.2.3 Técnicas de Prueba

El número de posibles combinaciones o rutas de ejecución se incrementa drásticamente a mayor número de procesos y procesadores involucrados Algunas combinaciones pudieran conducir a trayectorias o rutas de ejecución denominadas como **inseguras,** las cuales producen resultados incorrectos.

Por tal motivo, las reglas y métodos de prueba tradicionales utilizadas en la programación secuencial no son eficientes en modelos en los que exista una dependencia de datos, acceso de escritura entre procesos concurrentes o regiones inseguras amplias.

Una grafica de progreso, modela la ejecución de **n** procesos concurrentes como una trayectoria a través de un espacio cartesiano n-dimensional [BRYANT 2001].

La **figura 4-3** es una gráfica de progreso que representa el siguiente ejemplo:

Se tienen dos procesos concurrentes. Cada uno de lo procesos está compuesto de cinco instrucciones, y se denominan como A,B,C,D,E los estados alcanzados después de ejecutar cada una de las instrucciones. Las flechas indican las transiciones de un estado a otro ,la línea vertical hacia arriba indica ejecución del proceso 1, la línea horizontal a la derecha ejecución del proceso 2 y la línea diagonal ascendente a la derecha, paralelismo y ejecución de ambos procesos. Estas tres líneas de transición son las únicas validas.

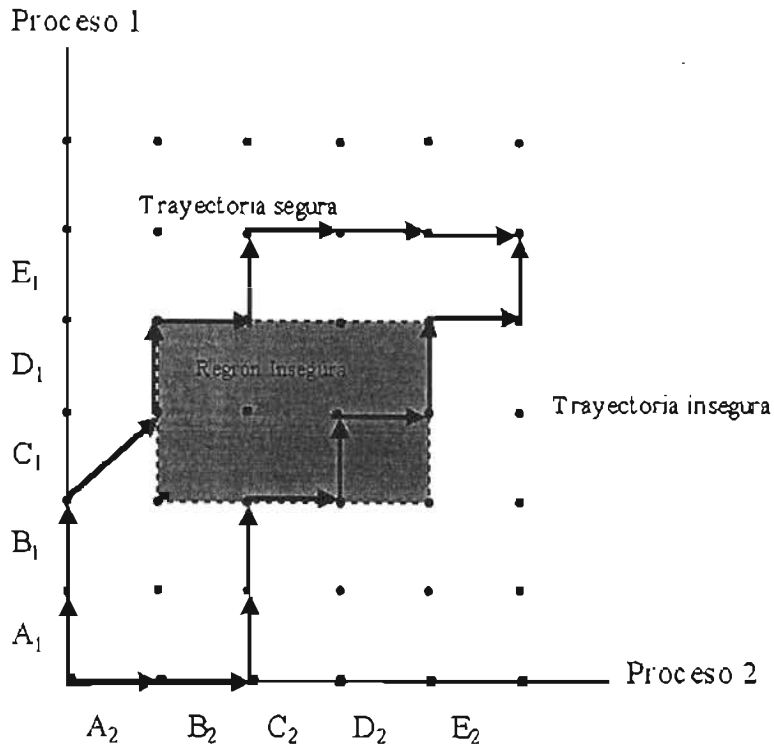


Figura 4-3 Gráfica de progreso.

Se denomina trayectoria al historial de ejecución del programa. La **trayectoria segura** es aquella que al representarse en la grafica de progreso no pasa por regiones inseguras. Una región insegura se produce al darse un orden no apropiado en la ejecución de las instrucciones de dos o más procesos. Las áreas que forman regiones inseguras estarán determinadas por el problema a resolver.

4.2.3.1 Condiciones de carrera, *deadlocks* y mecanismos de sincronización

Las condiciones de carrera ocurren cuando la correcta ejecución de un programa es afectada por el orden de ejecución de los procesos que lo conforman, dando lugar a regiones inseguras. Una de las maneras de evitar condiciones de carrera es mediante el uso de mecanismos de sincronización que controlen el acceso a los recursos comunes y el orden de ejecución de los procesos.

El empleo correcto de los mecanismos de sincronización, permitirá también disminuir la probabilidad de aparición de *deadlocks*. Un *deadlock* se produce cuando un proceso está en espera de que otro proceso ejecute cierta acción para poder continuar, y este último necesita también que el primero ejecute alguna

acción para poder continuar, de tal forma que, ambos se bloquean mutuamente. Se produce un bloqueo permanente de un conjunto de procesos que compiten por recursos del sistema o comunicarse entre ellos[STALLING 1995]. Ejemplo:

El proceso "A" se encuentra bloqueado en espera del recurso "R2", el cual, está bajo el control del proceso "B", y el proceso "B" se encuentra bloqueado en espera del recurso "R1" que se encuentra bajo control del proceso "A".

Entre los mecanismos de sincronización más comunes se encuentran:

4.2.3.1.1 Secciones Críticas:

Se conforman de secciones de código que deben ejecutarse de manera atómica, es decir sin interrupción, si más de un proceso intenta ejecutar la misma sección crítica, deberán esperar a que el primero en alcanzar el código termine la sección, antes que otro proceso la ejecute. Una vez que otro proceso comienza la ejecución de la sección crítica, los demás deberán esperar. Este procedimiento se repetirá hasta que todos los procesos que intentan ejecutar la misma sección critica de manera simultanea hubiesen terminado.

La creación de secciones criticas debe hacerse lo más corta posible, ya que esta puede afectar significativamente la concurrencia y paralelismo del programa completo.

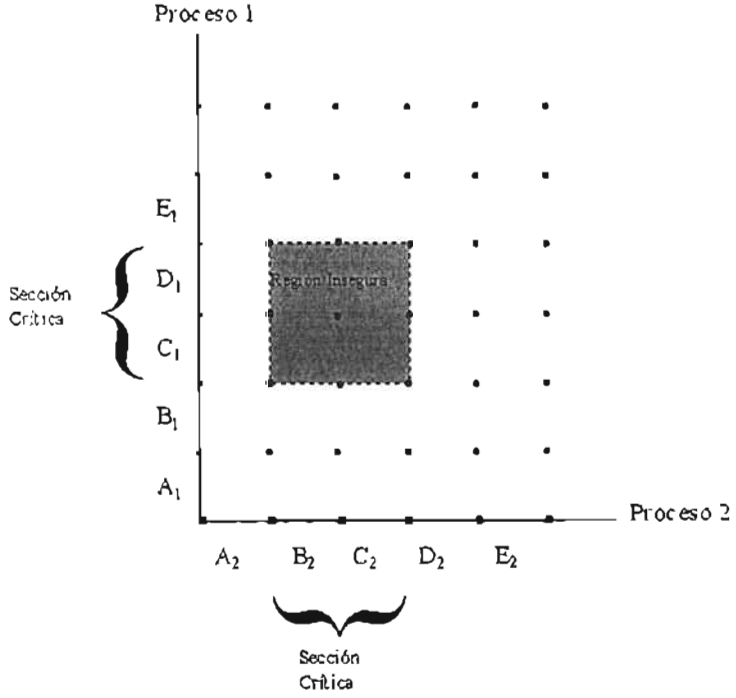


Figura 4-4 Gráfica de progreso con secciones críticas.

4.2.3.1.2 Bloqueo de variables.

Las secciones de memoria a las que accede más de un proceso, son bloqueadas durante las operaciones de escritura y en algunos casos lectura, mediante mecanismos como:

Mutex (*Mutual exclution*). Este mecanismo asigna el control absoluto de las variables que se encuentren dentro de la sección definida como de exclusión mutua. El primer proceso que tiene acceso a la sección de código de exclusión mutua, tendrá el control exclusivo de las variables contenidas en la sección. En caso de que otros procesos traten de tener acceso a las variables contenidas en el bloque de mutex, dichos procesos entrarán en alguno de los estados de inactividad hasta que el primer proceso termine la ejecución de la parte de mutex.

Una variante de estos mecanismos son los **bloqueos lectura/escritura**, los cuales, permiten el acceso concurrente a una variable, siempre y cuando se trate de accesos de lectura. Si alguno de los procesos solicita realizar una operación de escritura sobre la variable, se producirá un bloqueo generando una suspensión en todos los procesos que la traten de acceder, únicamente el proceso que solicitó la escritura podrá hacer uso de la variable. Terminada la operación de escritura se reestablece el acceso concurrente.

4.2.3.1.3 Variables de condición

Son empleadas cuando se requiere que el acceso al recuso compartido sea calendarizado, es decir, en los casos en los que no es suficiente con proteger un recurso de los demás procesos, sino que además, se requiere establecer un orden en el que dicho recurso será utilizado.

Semáforos

Un semáforo es una variable global entera positiva que se inicializa con un valor mayor que cero, generalmente uno; puede ser incrementada arbitrariamente, pero solamente puede ser decrementada hasta llegar a cero. Mediante el manejo de esta variable, se puede administrar el envío de señales de suspensión y reanudación a los procesos, controlando así el orden de su ejecución.

Dentro de este mecanismo se definen 2 operaciones para el manejo de la variable del semáforo (s) , a los cuales se les denominará P(s) y V(s) respectivamente, la operación P(s) decrementará la variable del semáforo siempre y cuando esta sea mayor a cero, la operación V(s) incrementara la variable en uno.

El mecanismo de función de los semáforos consiste en colocar una operación P(s) antes de ingresar a una sección critica, llevando la variable del semáforo a cero, al termino de la sección critica se realiza una operación V(s), la cual hace que "s" sea diferente de cero.

Mientras que "s" tenga un valor de cero, no se permitirá el cambio de proceso activo, ni el paralelismo, ordenando a otros procesos entrar en estados inactivos; controlando así, la trayectoria del programa al crear regiones prohibidas o inalcanzables en el diagrama de procesos. Véase la **figura 4-5**.

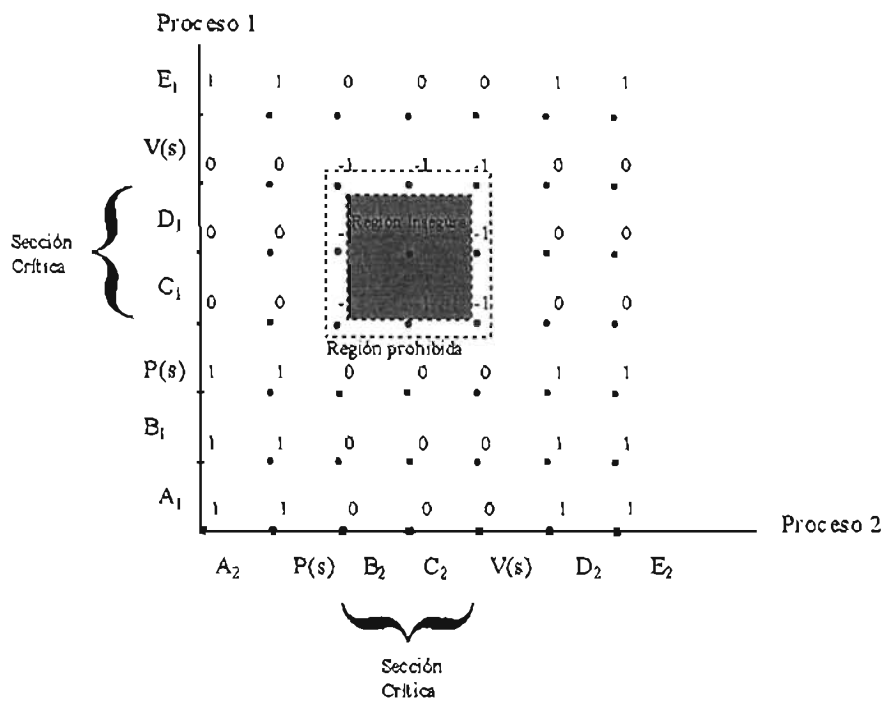


Figura 4-5 Gráfica de progreso con secciones críticas utilizando semáforos.

Uno de los inconvenientes del uso de los semáforos es que cuando en un mismo programa se utilizan varios semáforos definiendo regiones prohibidas que se traslapan, se producen ciertos estados dentro de una zona denominada región de *deadlock*. Si una trayectoria toca un estado de la región de *deadlock* (punto muerto), el *deadlock* es inevitable y la trayectoria nunca saldrá de la región de *deadlock*. En el siguiente ejemplo representado en la figura 4-6 se utilizan dos semáforos denominados "s" y "t", para los cuales, existen regiones prohibidas que se traslapan.

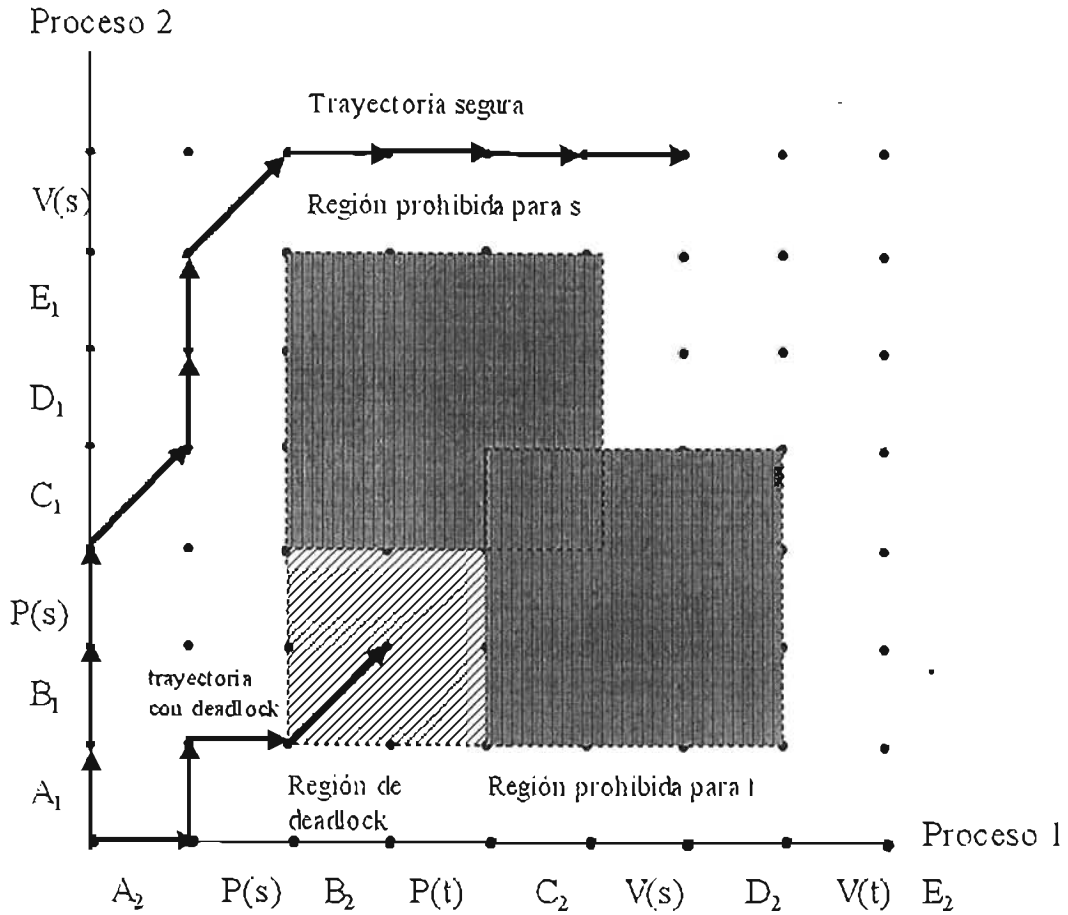


Figura 4-6 Gráfica de progreso de 2 proceso con secciones críticas que se traslapan.

Barreras

Una barrera permite a un conjunto de procesos sincronizarse en un punto de su código. Consiste en declarar una variable como barrera e inicializarla con un valor igual al número de procesos concurrentes, al alcanzar cada uno de ellos la parte del código de la barrera, esta se decrementa en uno y el proceso se detiene, cuando el último proceso alcanza el código de la barrera, el valor de la barrera llega a cero, y los procesos detenidos por la barrera reciben la señal de continuar.

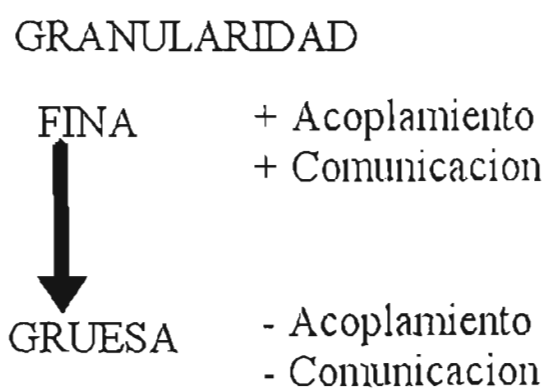
4.2.4 Granularidad

Se refiere a la unidad mínima de código que puede ser dispuesta para ser ejecutada de manera paralela, y casi siempre tienen una relación directa con el nivel de acoplamiento del sistema de paralelización. Se clasifica en fina, media y gruesa.

a)**Fina**. Se presenta en sistemas de ejecución paralela fuertemente acoplados a nivel de ciclos o sentencias individuales de programación, por lo general a nivel de ensamblador. Casi siempre es llevada a cabo por el hardware, como en el caso de la optimización de los procesadores superescalares mencionada anteriormente.

b) **Media**. La unidades de granularidad se constituyen por módulos, rutinas o funciones, hilos y procesos. Se realizan a nivel de programación por sistemas de paralelización automáticos, tal es el caso de OpenMosix. Se basa en procesos dentro de un mismo programa. También se produce en programas que utilizan bibliotecas de programación como PVM y MPI .

c) **Gruesa**. La granularidad se presenta a nivel de programas, tareas o módulos completos, generalmente depende de la forma en que el usuario final acomoda o lanza las unidades hacia un gestor cargado sobre el sistema operativo. Este gestor, mediante un sistema de colas administra la ejecución final, la comunicación entre unidades paralelas es escasa o no esta presente.



4.2.5 Modelos de concurrencia

Dependiendo del problema a resolver, en algunos casos el nivel de no-determinismo que tenga un programa concurrente es un factor decisivo "determinismo observable".

La manera en que los recursos compartidos por los procesos concurrentes son empleados, principalmente la memoria, dan como resultado varios modelos de concurrencia. Algunos de los más utilizados son la concurrencia declarativa y el paso de mensajes.

4.2.5.1 Concurrencia declarativa

Los programas escritos en este modelo son de utilidad en problemas con determinismo no observable. Los procesos generalmente se comunican entre si mediante acceso directo a áreas de memoria compartida. Se utilizan ampliamente en equipos con arquitectura de memoria compartida, o bien, en equipos de memoria distribuida que están formados por nodos de varios procesadores con memoria compartida dentro del nodo; en dicho caso el programa se pude ejecutar en un solo nodo.

Como ejemplo se tienen los programas que implementan las librerías y compiladores diseñados para equipos multiprocesador y el manejo de hilos (threads).

4.2.5.1.1 Hilos

Los hilos al igual que los procesos son unidades de ejecución pero con algunas diferencias.

Para efectos de esta tesis se definirá hilo como:

Unidad de ejecución asociada a un proceso pero con su propia pila, puntero de pila, contador de programa, códigos de condición , registros de propósito general e identificador de hilo [BRYANT 2001].

Los hilos al crearse comparten código, datos, montículo (heap), librerías, manejadores de señales, identificador de proceso(PID), estructuras de memoria virtual, apuntador de final y archivos abiertos declarados para el proceso en el cual se lanza el hilo. véase **figura 4-7**.

Al existir varios hilos, cada uno de ellos tendrá su propia pila y contexto de hilo (*Thread context*) integrado por registros de propósito general, códigos de condición, puntero de pila(SP) y contador de programa (PC), así como un identificador de hilo (TID).véase **figura 4-8**.

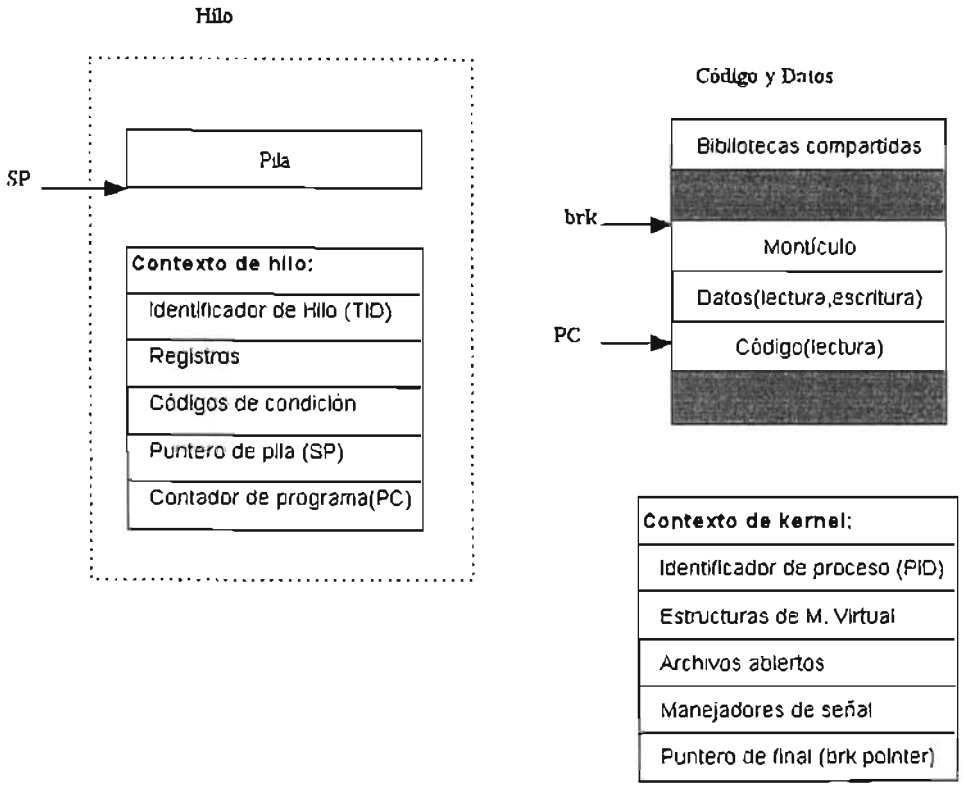


Figura 4-7 Componentes de un hilo.

De esta forma, al crear varios hilos dentro de un proceso se tienen varias unidades de ejecución concurrente relativamente independientes con algunos recursos propios y otros definidos en el proceso. La existencia de varios hilos, crea una estructura de memoria compartida a través de los recursos del proceso, que funge como un mecanismo trivial para compartir datos.

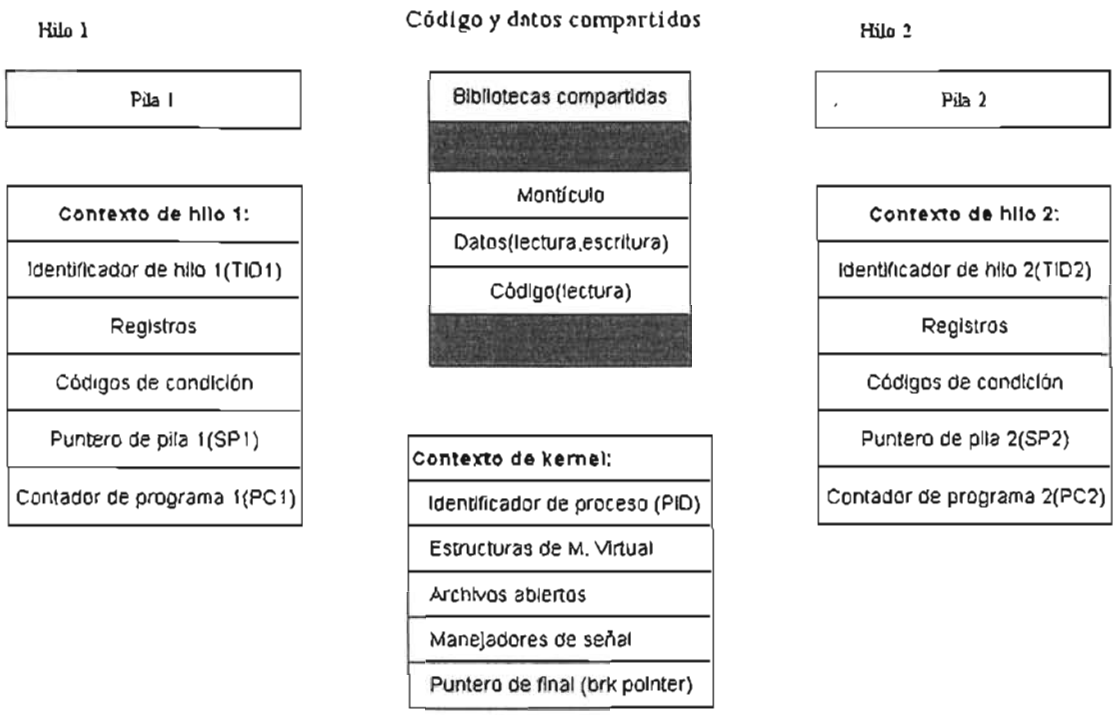


Figura 4-8 Recursos propios y compartidos al definir 2 hilos.

Al utilizar un proceso con varios hilos, se emplean menos recursos que al crear varios procesos concurrentes bajo el esquema clásico de los procesos.

Debido a que el contexto de un hilo es más pequeño que el de un proceso, el cambio entre un hilo y otro *(thread context switch)* es más rápido que si se llevara a cabo entre procesos. **figura 4-9**.

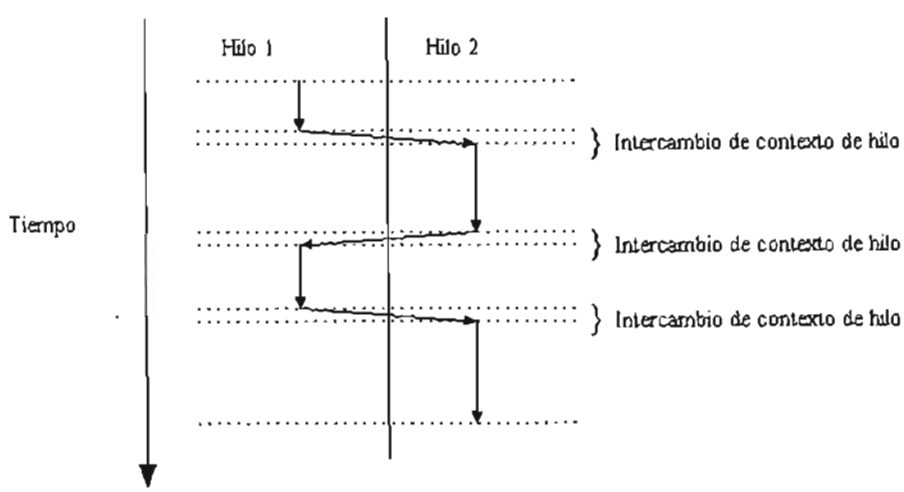


Figura 4-9 Thread Context Switch

Los hilos a diferencia de los procesos, no se organizan en la estricta jerarquía padre-hijo. Todos los hilos tienen acceso a los recursos comunes, pueden terminar la ejecución de otros hilos y sincronizarse con ellos, independientemente del orden de creación y que hilo los creara dentro del proceso.

Bajo este esquema, dentro de todo proceso existe un hilo principal, cuya única diferencia con los demás es haber sido el primero en ser creado y ejecutado por el flujo normal del programa, y ser también el primero en crear y lanzar otros hilos.

Los sistemas operativos modernos, permiten la concurrencia por medio de procesos tradicionales o bien mediante hilos. En ambos casos son necesarios los mecanismos de sincronización mencionados en la sección de procesos, dadas las características de la ejecución concurrente.

Inicialmente cada sistema definía sus propias bibliotecas "APIs" para la creación de hilos, así como la forma de uso de las mismas, dificultando la creación de sistemas transportables a múltiples plataformas. Para tratar de resolver esta situación la IEEE(*Institute of Electrical and Electronics Engineers*) ha creado varios estándares que dan lugar a POSIX, (*Portable Operating System Interface*)que entre otras cosas, trata cuestiones sobre lenguajes, comandos, métodos de prueba e interfaces de sistema operativo.

POSIX 1003.4a también conocido como ISO/IEC 9945-1, es la porción de POSIX que se refiere a los hilos. Incluye las funciones y las interfaces de programación de aplicaciones APIs que soportan múltiples flujos de control dentro de un procesador[NORTON 1996].

El resultado de POSIX en cuanto a hilos, es la definición de las bibliotecas de lenguaje C denominadas Pthreads.

Sin embargo, los distintos compiladores y sistemas operativos llegan a integrar sus propias bibliotecas para manejo de hilos.

Hilos de usuario e hilos de kernel

Los hilos de usuario, son creados, administrados y terminados por una aplicación que utiliza las funciones de API de manejo de hilos. Existen en el espacio del usuario y ejecutan código del usuario. Para tener acceso a los recursos exclusivos del kernel, deben estar asociados a un hilo del kernel.

Los hilos de kernel, son creados por funciones de las bibliotecas de hilos, son visibles para el kernel del sistema operativo. Un hilo de kernel puede soportar uno o más hilos de usuario. Su función es ejecutar código de kernel o llamadas de sistema en nombre de hilos de usuario. Existen en el espacio del kernel.

El espacio de usuario se refiere a la parte de la memoria donde se coloca el código de las aplicaciones que ejecuta el sistema operativo. El espacio de kernel es una zona de memoria donde es colocado el kernel del sistema, las instrucciones colocadas en este espacio tienen los privilegios de ejecución, así como el acceso privilegiado a los recursos del sistema.

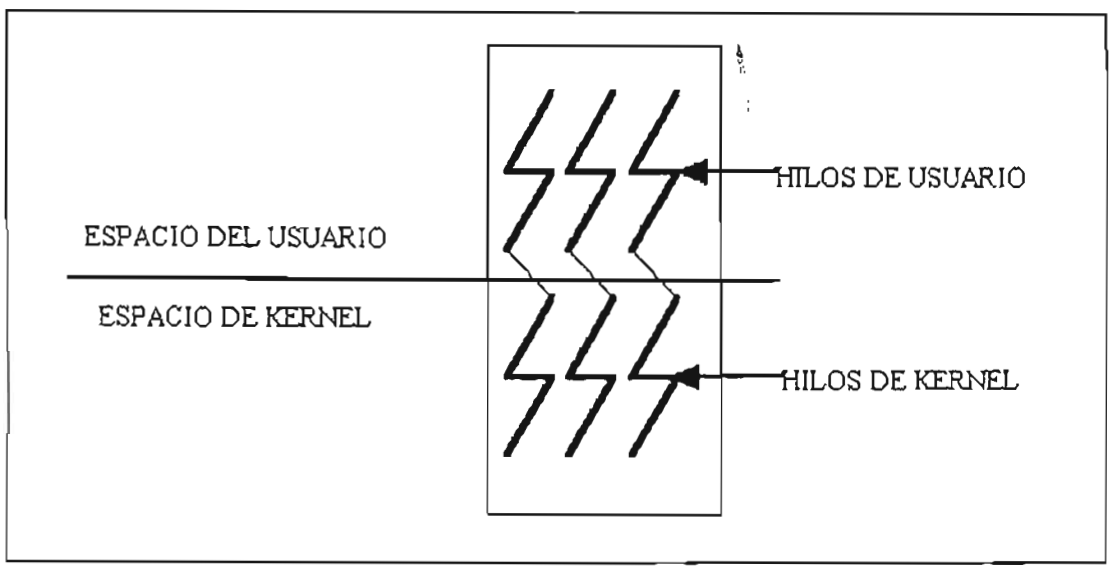


Figura 4-10 Tipos de hilos y espacios.

Operaciones con hilos

Al crearse hilos dentro de un programa, estos pueden escribir o leer datos de áreas memoria comunes, tener dependencia de datos o recursos entre ellos, y en algunos casos correr solamente en ciertos momentos y permanecer inactivos o en espera en otros. Para prevenir errores en tiempo de ejecución o lógicos, además de la creación y terminación de hilos existen otras operaciones sobre los hilos (véase **figura 4.11**) que dan como resultado los siguientes estados:

Operaciones:
Detener, continuar, despertar ,dormir ,activar, despachar, terminar.

Estados:
Ejecutable, detenido ,activo ,dormido, terminado[LEWIS 1996].

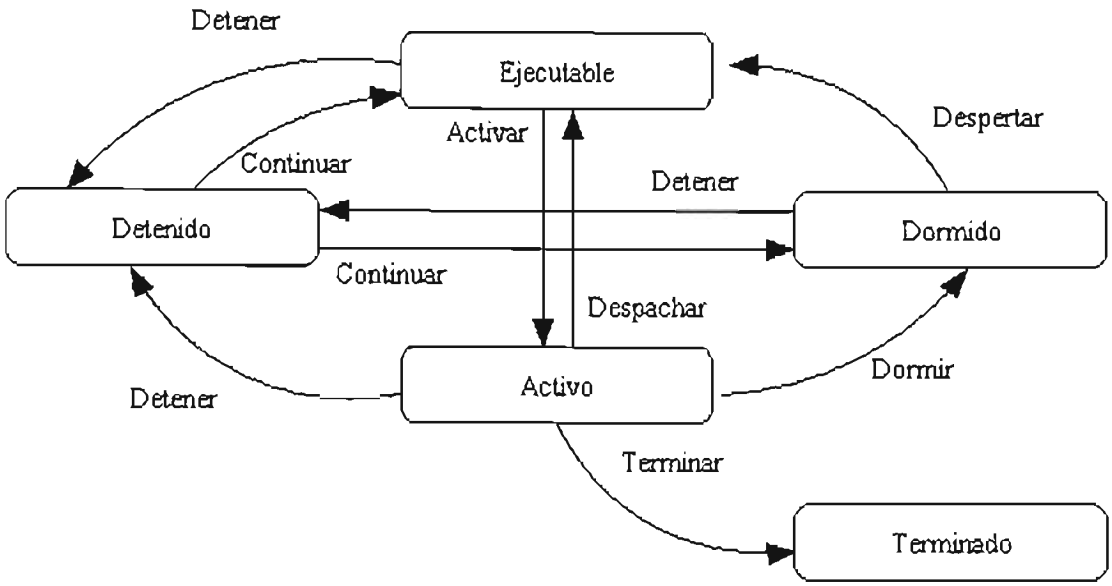


Figura 4-11 Estados y transiciones de los hilos.

4.2.5.2Paso de Mensajes

Este modelo implementa el concepto de puerto. Los procesos se comunican entre si mediante mensajes, que consisten en accesos de lectura y escritura a los puertos.

Un puerto es un tipo de dato abstracto, definido en la implementación de paso de mensajes. Se realizan básicamente tres operaciones con respecto a los puertos:

Creación del puerto, envío de datos al puerto, recepción de datos desde el puerto.

Creación

Durante la creación del puerto "P", este es asociado con:

- ✓ Una unidad de ejecución que para efectos de generalización se denominará "proceso", mediante un identificador "I" con el cual se creo dicho proceso , ya sea, el nombre de manejador, PID, TID o algún otro dependiendo de la implementación en particular.
- ✓ Un espacio de almacenamiento "S" donde se guardaran de manera secuencial con mecanismo FIFO (First Input First Output) los mensajes que se envían hacia dicho puerto "P".

PuertoNuevo(P,I,S)

Envío

Esta operación consiste en enviar un mensaje almacenado en la variable o estructura "M", la cual ,deberá tener un tipo compatible con los registros del espacio de almacenamiento "S" del puerto "P" , que previamente se asoció con una unidad de ejecución "I". Este mecanismo de comunicación es asíncrono, debido a que una vez que el proceso envía el mensaje al puerto, no necesita confirmación del proceso receptor.

Envia(P,M)

Recepción

En este caso, el proceso que tiene asociado el puerto P, solicita la recuperación de los mensajes o de un mensaje de manera individual almacenado en el puerto P, y lo coloca en su variable o estructura local M.

Recibe(P,M)

En caso de que la implementación solamente permita la asociación de un puerto por proceso no será necesaria la indicación del identificador del mismo.

Recibe(M)

Debido a que los puertos son secuénciales, se disminuye el no-determinismo. El orden de llegadas de los mensajes enviados desde dos procesos A y B hacia un tercer proceso C, no esta definido, pero, dos mensajes enviados desde un proceso A, hacia C llegarán en el orden en que fueron enviados. **Figura 4-12**.

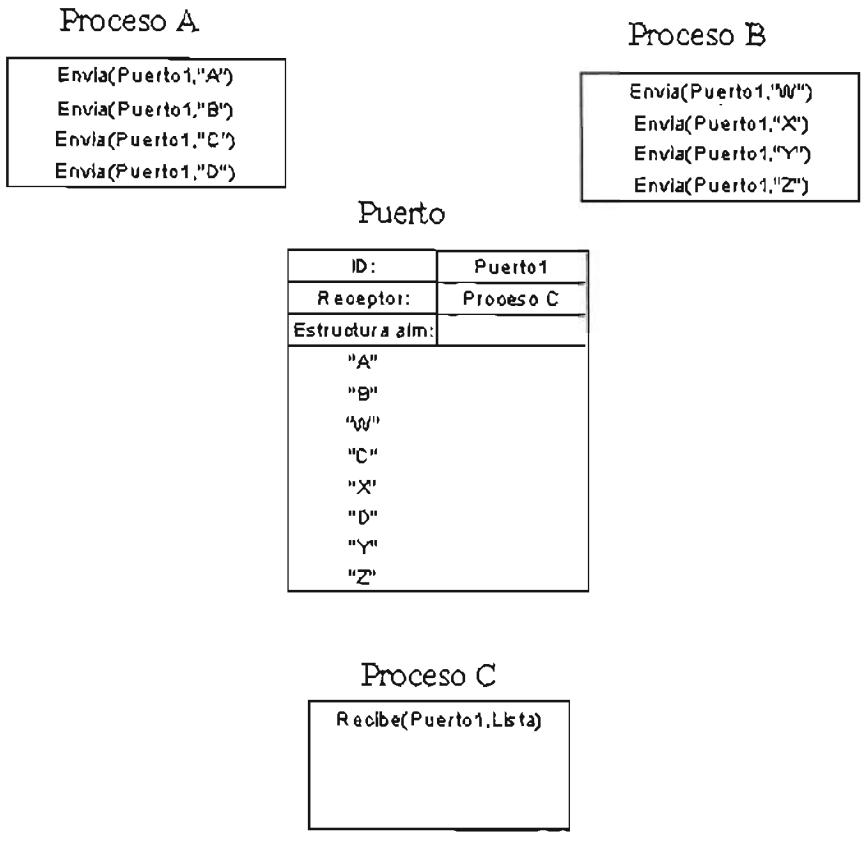


Figura 4-12.Envió y recepción de mensajes.

Algunas implementaciones permiten reducir aun más el no-determinismo al introducir el concepto de canal. Un canal "C" al igual que un puerto tiene asociado un espacio de almacenamiento "S", destinado solamente a un proceso receptor y un proceso emisor, lo que permite definir en que orden se recuperarán los mensajes de acuerdo al proceso emisor. Las operaciones son las mismas que en el modo normal de manejo de puertos, con la distinción que al momento del crear el puerto se establecen los procesos emisor "E" y receptor "R".

CanalNuevo(C1,E1,R,S1)
CanalNuevo(C2,E2,R,S2)
Envia(C1,M)
Envia(C2,M)
Recibe(C1,M)
Recibe(C2,M)

La **figura 4-13** muestra un ejemplo de empleo de canales:

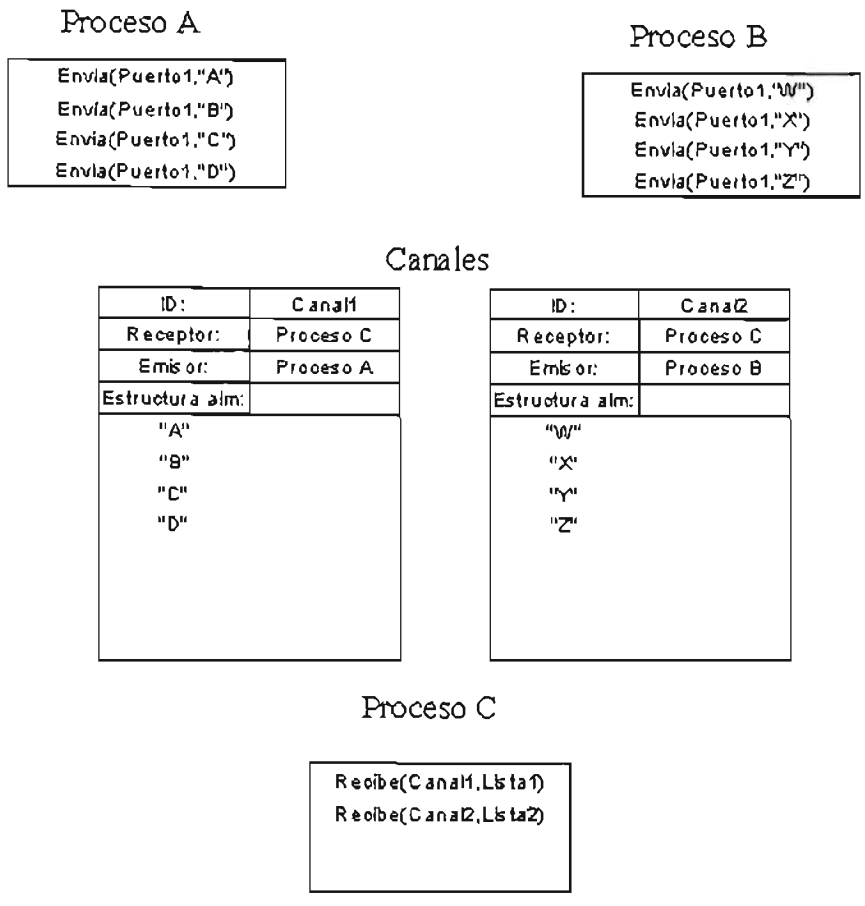


Figura 4-13 Canales

Todas las llamadas a función mencionadas anteriormente son generalizaciones, su número y formato dependerá de la implementación de paso de mensajes en particular que se emplee.

El modelo de paso de mensajes se utiliza ampliamente en equipos de memoria distribuida como es el caso de los Clusters. En el ámbito de software se puede tomar como ejemplo los programas implementados con las librerías PVM ("Paralell Virtual Machine") y MPI ("Message Passing Interface"). Por lo regular su programación conlleva una mayor complejidad que la de concurrencia declarativa.

4.2.6 Aplicación

Para el desarrollo del sistema de esta tesis, se emplea un modelo de concurrencia declarativa mediante hilos. Se selecciono este modelo, ya que permite de manera simple el acceso a los datos compartidos.

La solución implementada consiste en generar dos hilos, uno para los cálculos de DM y otro para el sistema gráfico. Al funcionar el sistema gráfico en un ambiente de eventos, tiene un consumo mínimo de recursos mientras el usuario no interactué con él, desahogando el equipo para emplear la mayor cantidad de recursos en el hilo de DM.

Al utilizar OpenGL, en el momento de que el usuario genera un evento que pone a funcionar el hilo de manejo de gráficos, muchos de los recursos empleados son del adaptador gráfico y no del procesador o procesadores del Sistema.

No son necesarios mecanismos de bloqueo de variables, ya que el hilo encargado de la graficación únicamente realiza operaciones de lectura sobre las variables manejadas en los cálculos de DM. Los mecanismos de sincronización y bloqueo de variables, serían empleados en un desarrollo futuro, en el que se paralelizarán las funciones de cálculos de Dinámica Molecular.

En la implementación en el Sistema Operativo Microsoft Windows se empleó el compilador Borland C++ Builder 5. Para el manejo de hilos en Builder se dispone de 2 opciones:

- ✓ Uso de APIs del sistema operativo.
- ✓ Manejo de objetos TThread predefinidos en Builder para el manejo de hilos. Tt
- ✓ TThread encapsula las funciones de hilos y simplifica la programación de funciones avanzadas para el trabajo de hilos[HOLLINGWORTH 2001].

La opción seleccionada fue el uso de WinAPI 32. WinAPI 32 se conforma de una serie de bibliotecas incluidas en el sistema operativo para el desarrollo de aplicaciones. Las funciones de manejo de hilos se encuentran en la biblioteca kernel32.dll. Esta biblioteca permite el manejo a bajo nivel de archivos, memoria y recursos del sistema y multihilos. Su implementación a bajo nivel, permite obtener una mayor eficiencia.

TThread no fue seleccionada debido a que :

1. Al utilizarse junto con GLUI producía algunos errores por identificadores duplicados, la corrección de este error implicaba la modificación del código fuente de GLUI .
2. No se emplean mecanismos de bloqueo o sincronización por lo que las funciones de manejo de hilos contenidas en WINAPI son suficientes.

En el caso del Sistema Operativo Linux se empleó la biblioteca PThreads del estándar POSIX, implementada desde la instalación de la distribución empleada.

CAPÍTULO 5. Integración.

El sistema desarrollado en esta tesis, consiste en una aplicación de Dinámica Molecular que permite la visualización del modelo molecular simulado, así como la generación de gráficos estadísticos de varias propiedades macroscópicas. La visualización gráfica y los cálculos de Dinámicas Molecular se realizan de manera concurrente, dando la posibilidad al investigador de observar el desarrollo de la simulación sin tener que detenerla. También le permite observar si en algún momento se producen condiciones no deseadas o no esperadas. En tal caso, se puede tomar la decisión de detener todo el proceso y corregir los datos iniciales. Esta característica es de gran utilidad considerando que algunas ejecuciones emplean del orden de varios cientos de miles de pasos, implicando varios días de ejecución.

Para lograr la concurrencia se emplean hilos. En el caso de los gráficos se utiliza OpenGL por su nivel de eficiencia. Para la solución del sistema de ecuaciones de Dinámica Molecular es empleado el Algoritmo de Verlet y los procedimientos descritos en el capítulo 2.

5.1 Elementos utilizados

El sistema se desarrolló para los sistemas operativos Linux y Windows. El lenguaje de programación empleado fue C++.

En el caso de Linux se empleo el siguiente software de código abierto.Véase **figura 5-1**:

Compilador:
Gcc (GNU C Compiler).- 3.2

APIs:

Mesa3d
POSIX Threads(Pthreads)
GLUT 3.7 de Mark J. Kilgard
GLUI de Paul Rademacher versión 2.0 adaptada para funcionar con gcc 3.

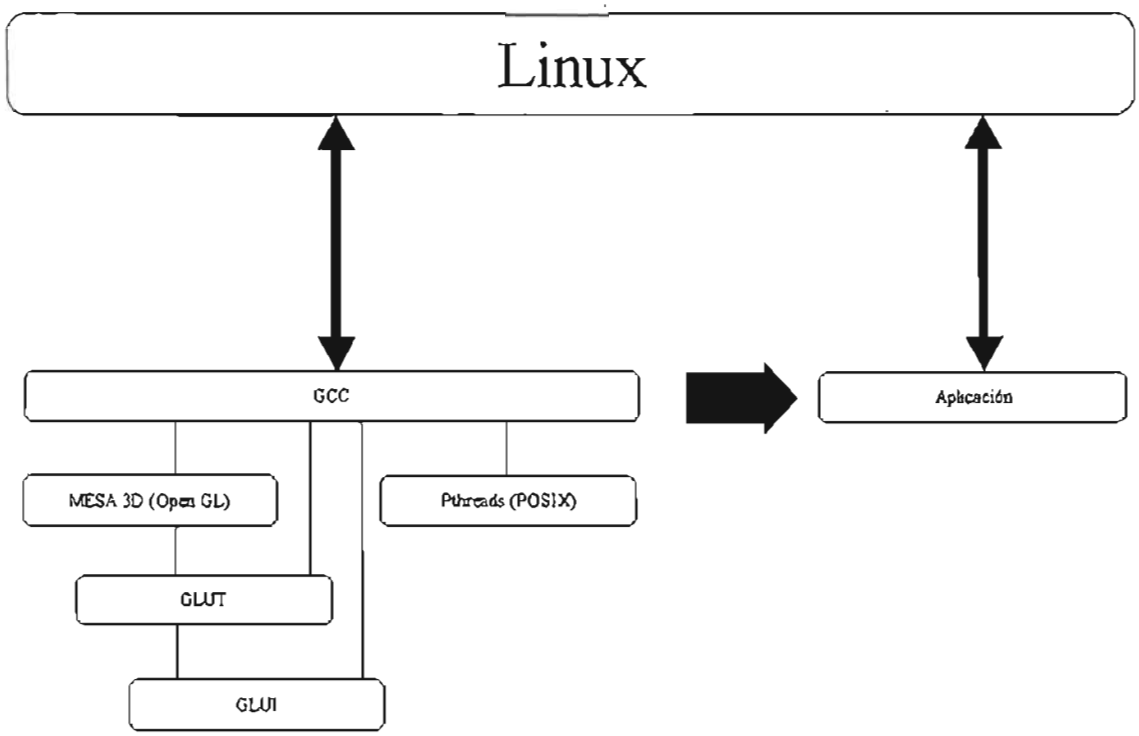


Figura 5-1. Recursos empleados en Linux.

Para el desarrollo en Microsoft Windows fue empleado el compilador Boralnd C++ Builder versión 5 y las siguientes APIs.véase figura 5-2:

OpenGL incluida con Borland C++ Builder
WinAPI incluidas con el sistema operativo.
Una versión de GLUT para Borland C++ Builder 5 basada en la versión 3.7 del código fuente.
Una versión de GLUI para Borland C++ Builder 5 fue complilada como biblioteca a partir de código fuente de GLUI de Paul Rademacher.

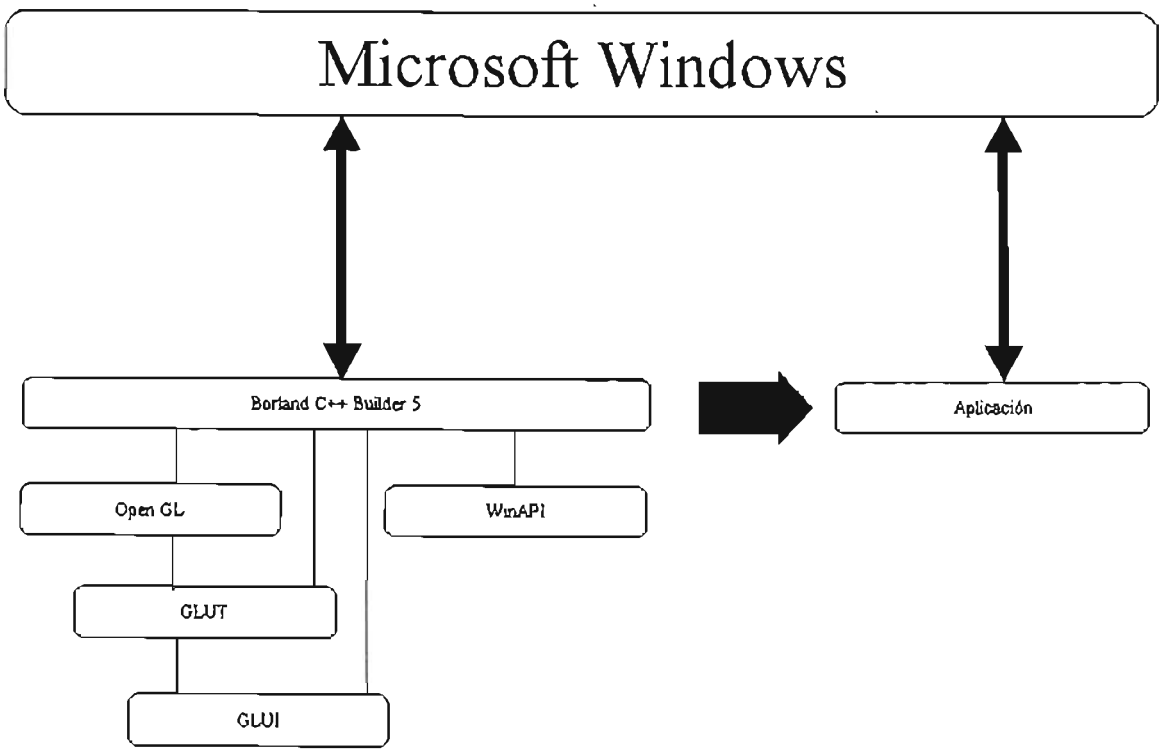


Figura 5-2 Recursos empleados en Microsoft Windows.

5.2 DESCRIPCION GENERAL DEL SISTEMA

La ejecución del sistema consta de 2 Hilos :

El hilo principal se encarga de realizar la inicialización de toda la simulación, así como la invocación del sistema gráfico y del segundo hilo.

El segundo hilo, contiene las funciones encargadas de realizar los cálculos matemáticos referentes a la simulación numérica.

Por las características de GLUT, al invocar al sistema gráfico de OpenGL, se crean y dibujan todas las ventanas creadas, así como todas las directivas de dibujo asociadas con la función principal de dibujo, después de esto, el sistema entra en un modo de espera u orientado a eventos . Las diversas funciones que regeneran el sistema gráfico son asociadas a eventos que son producidos por el usuario en cualquier momento de la ejecución. Debido a esto, los datos de la simulación gráfica, o la forma en que estos son presentados se actualizan únicamente mediante intervención del usuario, ahorrando recursos de computo que son empleados por el hilo de los cálculos. La utilización de OpenGL, además permite un mejor uso de recursos del microprocesador, ya que estas bibliotecas se encuentran optimizadas para emplear preferentemente los recursos de la tarjeta de video antes que los de procesamiento central.

El sistema se resume en las siguientes tareas:
1.Cálculo de valores iniciales de la simulación.
2. Inicialización del sistema gráfico.
3. Creación y encendido de hilo de cálculos.
4. Encendido del sistema grafico y captura de eventos del usuario.

A continuación las tareas se describen con mayor detalle:
1. Calculo de valores iniciales de la simulación:
   1.1 Tamaño de la caja.
   1.2 Cargas.
   1.3 Posiciones de los átomos dentro de la molécula.
   1.4 Posiciones iniciales de las moléculas.
   1.5 Orientaciones iniciales de las moléculas.
   1.6 Carga de los archivos que contienen los centros de masa y quaterniones, y de las velocidades de rotación y traslación iniciales del modelo.
   1.7 Asignación de las velocidades iniciales y angulares.
   1.8 Fuerzas y torques que actúan sobre todas las moléculas.
   1.9 Valores iniciales de energía potencial y cinética.

2. Inicialización del sistema grafico
   2.1 Inicialización de OpenGL.
   2.2 Modo de despliegue(*buffers*, colores, profundidad).
   2.3 Colores de borrado, sistemas de coordenadas y áreas de visualización.
   2.4 Parámetros de iluminación.
   2.5 Dimensiones y posición de las ventanas principales y secundarias.
   2.6 Relación entre funciones y eventos.

3. Creación y encendido del Hilo de Cálculos.
   En el caso de Microsoft Windows, se utiliza la API del sistema creando el hilo de modo suspendido al inicio de la función principal y se activa justo antes de encender el modo gráfico. En Linux se empleo la librería *pthreads* incluida en la distribución. Pthreads se ajusta al estándar POSIX. Tanto la creación como el encendido del Hilo se hacen antes de activar el modo grafico.

   Al activar el hilo asociado con la función de cálculos inicia el ciclo de la simulación numérica. Los cálculos comprenden los siguientes procesos:

   a) Apertura de los archivos de salida.

b) Ejecución desde 1 hasta NumPasos de:

Cálculo de Paso del sistema de Ecuaciones Diferenciales:
Rutina principal del sistema, en ella reside la mayor parte de la simulación, y se resuelve el sistema de ecuaciones, comprende los siguientes subprocesos:

3.1. Recálculo de orientaciones
Cálculo desde 1 hasta NumPartículas
Momento angular.
Matriz de rotación en el paso t.
Velocidad Angular a partir de los momentos angulares antes obtenidos.
Derivadas de cuaterniones en en paso t.
Cuaterniones en el paso $t + \frac{1}{2}\Delta t$.
Matriz de rotación en el paso $t + \Delta t$.
Momento angular en el paso $t - \frac{1}{2}\Delta t$ a $t + \frac{1}{2}\Delta t$.
Velocidades angulares a partir de los momentos angulares obtenidos en el paso $t + \frac{1}{2}\Delta t$.
Energía cinética rotacional en el paso $t + \frac{1}{2}\Delta t$.
Derivadas de cuaterniones en el paso $t + \Delta t$.
Normalización de cuaterniones..

3.2.- Recálculo de las nuevas posiciones y velocidades
Cálculo de 1 hasta NumPart de:
Posiciones de los centros de masa en el paso $t + \Delta t$ y velocidades en $t + \frac{1}{2}\Delta t$.
Posiciones de los átomos tomando en cuenta las condiciones periódicas de frontera así como las cargas.

3.3- Cálculo desde 1 hasta NumPart de:
Fuerzas y Torques que actúan sobre las moléculas en el paso $t + \Delta t$.

3.4.- Calculo desde 1 hasta NumPart de:
Velocidades desde $t + \frac{1}{2}\Delta t$ a $t + \Delta t$

3.5. Obtención de los siguientes valores
Energía potencial en el paso $t + \Delta t$
Energía potencial en el paso $t + \frac{1}{2}\Delta t$
Energía cinética de traslación en $t + \Delta t$

Energía cinética de traslación en $t + \frac{1}{2}\Delta t$

Energía cinética total(traslación + rotación) en el paso $t + \frac{1}{2}\Delta t$

Energía total en el paso $t + \frac{1}{2}\Delta t$

Temperatura en el paso $t + \frac{1}{2}\Delta t$

Presión en el paso $t + \frac{1}{2}\Delta t$

3.6. Colocación de los datos correspondientes al paso en el vector que contiene los datos para la generación de las ventanas de presión, temperatura y densidad.

3.7. Escritura en los archivos de salida cada determinado número de pasos, según definición del usuario

4. Encendido del sistema gráfico.
   Aquí se activa el sistema gráfico mediante GLUT, que toma el control y captura los eventos generados por el usuario. Al producirse eventos asociados con funciones previamente definidas en el paso 2, se ejecutaran de manera concurrente con el hilo de cálculos del paso 3. Aun después de concluida la ejecución del hilo de cálculos, el sistema GLUT seguirá activo permitiendo la interacción y regeneración visual del modelo obtenido en el último paso. Al seleccionar el usuario la opción de finalizar el programa en cualquier momento de la corrida, la simulación completa terminará. Entre las funciones que puede seleccionar el usuario se encuentran las de:

   - Rotar o trasladar el modelo.
   - Realizar acercamientos y alejamientos.
   - Encender o apagar el sistema de iluminación.
   - Utilizar la representación de los átomos por medio de esferas sólidas o mediante estructuras de alambre.

Las últimas dos opciones ahorran recursos y aumentan la velocidad con que se regenera en pantalla el sistema en equipos con tarjetas de vídeo con aceleración de gráficos de bajas prestaciones. De igual forma, al realizar una rotación o traslación solamente se muestran los enlaces entre los átomos de las moléculas para incrementar la velocidad de regeneración y hacer más fluido el manejo del programa.

Adicionalmente, al interactuar con alguna de las ventanas de la simulación, únicamente esta ventana será actualizada. Las ventanas que se presentan son:

- Principal del modelo.
- Gráfico estadístico de temperatura.
- Gráfico estadístico de presión.
- Grafico estadístico de densidad.

Los eventos que implican una actualización de los datos contenidos en una ventana son:

- Hacer clic sobre la ventana.
- Mover la ventana.
- Redimensionar la ventana, ajustando automáticamente la escala de su contenido para llenar la nueva área de visualización.
- En el caso de la ventana principal, las operaciones de rotación, traslación y acercamiento o alejamiento implicaran una regeneración grafica del modelo.

Al generar un evento que implique la actualización de una ventana de grafico estadístico, se limpiara dicha ventana y se realizara una línea uniendo los puntos almacenados en el vector llenado en el hilo de cálculos para dicho propósito.

Al generar un evento que de cómo resultado la actualización de la ventana principal de la simulación, se borrara el contenido de la ventana destinado a la visualización del modelo, se calculara la posición de cada uno de los átomos del modelo, y se dibujara de acuerdo a los valores de acercamiento, iluminación y estructura de alambres seleccionados, se dibujaran los enlaces y se presentará el número de paso correspondiente. En el caso de la rotación y la traslación solamente se dibujaran los enlaces hasta que el usuario deje de oprimir el botón del ratón.

5.3 SEGMENTACION DEL CODIGO

Para su manejo el código se divide en varios archivos con sus respectivas cabeceras:

WATTER.CPP
Programa principal.

DATA.CPP
Definición de constantes, variables de uso general y vectores.

GRAFIC.CPP
Funciones relacionadas con el modo gráfico OpenGL

GEOM.CPP
Funciones relacionadas con el hilo de cálculos. Fuerzas y torques, energía potencial, cálculos del Sistema Ecuaciones Diferenciales.

INIT.CPP
Calculo de valores iniciales del modelo.

VECTOR.CPP
Funciones para el calculo de operaciones con vectores.

UTIL.CPP
Contiene funciones para manejo de errores, manejo de asignación dinámica de memoria para los vectores y las matrices.

RANDOM.CPP
Funciones para generación de números aleatorios basándose en los algoritmos RAN3 y RAN1.

5.4 OPERACIÓN DEL SISTEMA

Preparativos para la ejecución:
Deberá disponerse de un archivo de texto separado por espacios que contenga los datos de las posiciones de los centros de masa, así como los quaterniones de cada una de las moléculas. Al tratarse de modelos rígidos, las posiciones de los átomos se calculan a partir de los valores de los centros de masa que determinaran la traslación de la molécula, y los quaterniones su rotación. El formato del archivo de entrada se muestra en el siguiente fragmento de archivo de entrada:

```
7.37524e-11  2.09234e-10  2.31322e-10  -0.666673  -0.574805  -0.247403  -0.404892
2.50057e-10  2.55117e-10  7.36439e-12  0.81802  0.422656  0.0959409  -0.378155
2.52376e-10  5.07404e-10  6.62936e-10  -0.868156  -0.229764  0.265011  0.351115
4.46519e-11  5.4948e-10  1.98318e-10  -0.252591  0.840979  0.309621  -0.364811
2.85795e-10  2.05501e-10  6.97218e-10  -0.164678  0.261186  -0.607184  0.732113
3.626e-10  3.78572e-10  3.61241e-10  0.176515  0.456697  -0.850013  -0.194289
```

. . .

Los tres primeros valores indican las coordenadas x,y,z del centro de masa y los cuatro restantes los valores del cuaternion(x,y,z,u). Cada valor es separado por un espacio y cada renglón comprenderá una molécula.

También será necesario disponer de un archivo en formato de texto con las velocidades iniciales traslación y angulares (rotación) del modelo, como el mostrado en el siguiente fragmento de archivo de entrada:

```
-842.129 152.644 84.6192 5.10276e+13 2.36341e+12 2.37421e+13
240.803 175.557 410.222 7.49511e+12 2.95495e+13 -1.08076e+12
-26.9258 61.62 -374.495 6.81646e+12 3.46077e+13 7.70712e+11
-79.1375 672.868 50.949 3.40367e+12 1.99278e+13 -9.35688e+12
108.662 -199.053 187.222 -2.35394e+13 1.05396e+13 -4.40714e+12
270.774 -80.8163 -46.8265 1.46553e+13 1.28609e+13 -3.0519e+13
-534.092 -65.2629 125.548 6.71945e+12 1.26473e+13 1.10706e+13
32.9327 -236.055 277.101 7.35895e+12 2.34347e+12 -5.04851e+12
-302.566 -34.6902 -25.2757 -1.85415e+13 -5.63184e+13 -9.18842e+12
-37.6305 -110.105 -1212.74 -8.34147e+11 2.70541e+13 -7.83561e+12
. . .
```

Los tres primeros valores de cada registro se refieren a las velocidades de traslación en los ejes x,y,z respectivamente. Los tres valores restantes corresponden a las velocidades angulares en x,y,z. Al igual que en el archivo de entrada anterior, cada valor es separado del siguiente por un espacio simple, y cada renglón corresponde al registro de una molécula.

En cuanto a los parámetros de la simulación como condiciones iniciales de presión y temperatura, número de pasos, número de moléculas, tamaño de paso, distancias y ángulos entre los átomos de la molécula; fuerzas y otros valores referentes a la molécula a simular, así como constantes físicas utilizadas. Son definidos en los archivos de programa DATA.CPP y DATA.H.

Al iniciar la ejecución del sistema, se despliegan cinco ventanas, una de texto, que muestran los valores de: número de paso, presión, temperatura y densidad cada cierto número de pasos. Otras tres contendrán gráficos estadísticos de propiedades macroscópicas. La ventana restante mostrará la simulación grafica del modelo. Véase **figuras 5-3 y 5-4** Al comienzo de la simulación mostrara el modelo inicial, en caso de que el usuario interactué con dicha ventana, esta se redibujará mostrando los valores del modelo en el paso actual de la simulación. Si la ejecución de los N pasos de la simulación termina, se redibujará mostrando el modelo final.

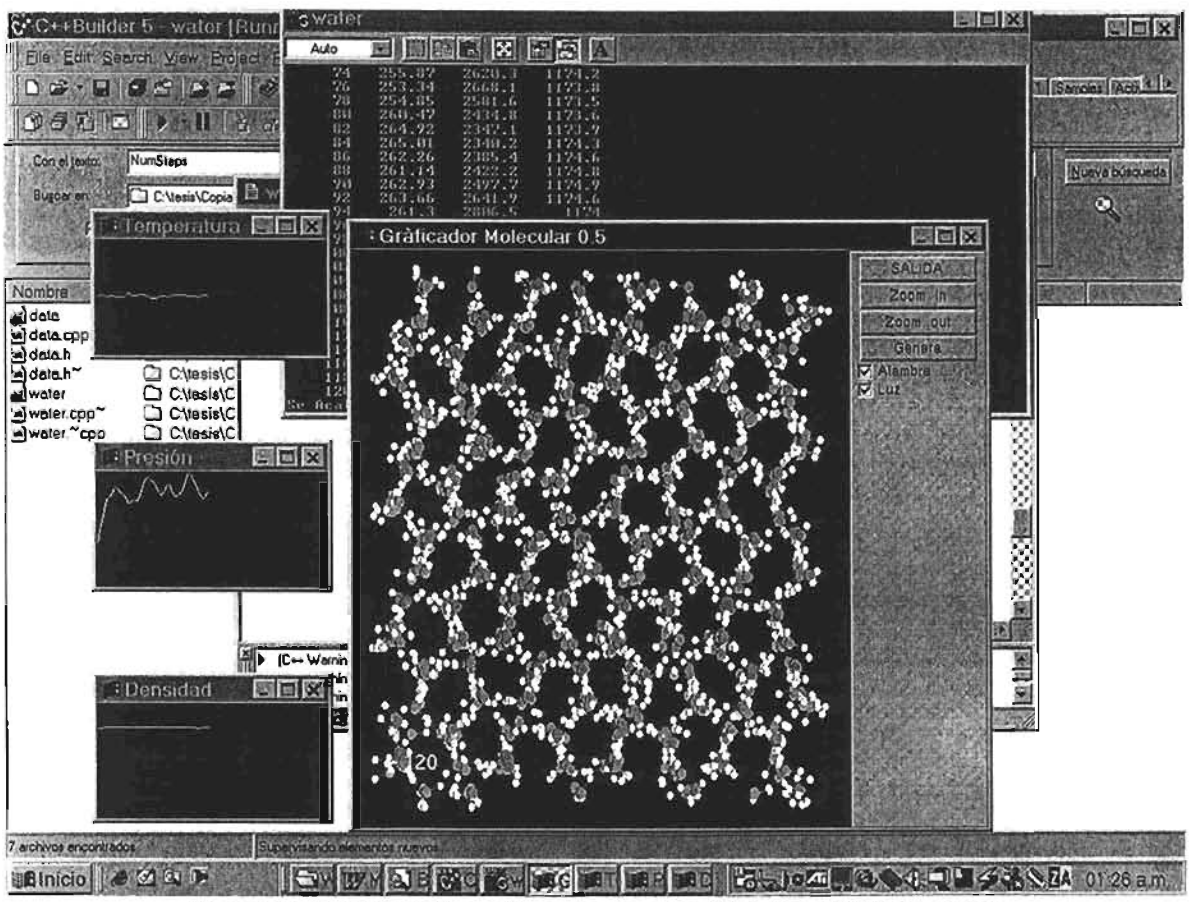


Figura 5-3. Ejecución del programa en Microsoft Windows 98.

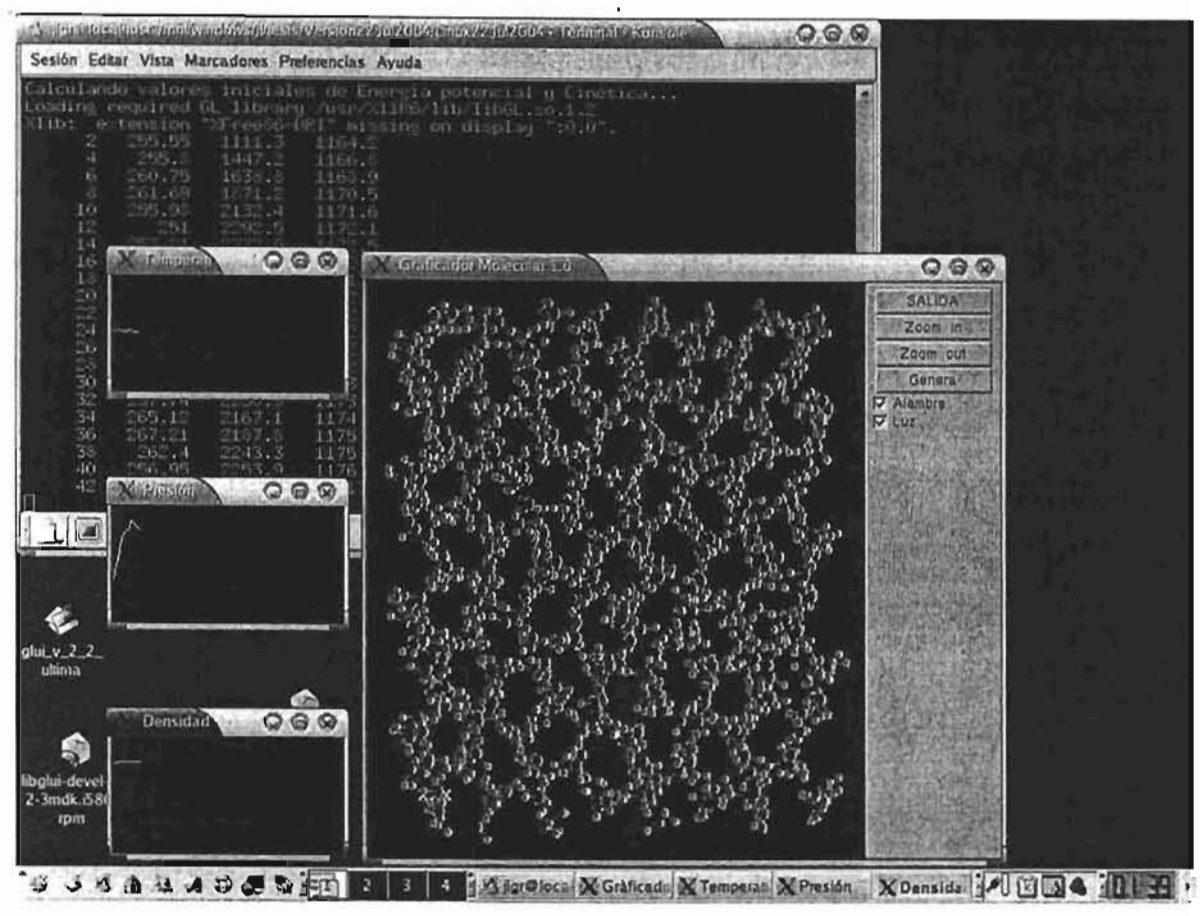


Figura 5-4 Ejecución del programa en Linux.

Archivos de Salida:

Durante la ejecución se generan cuatro archivos de salida en los siguientes formatos:

El primero contendrá el número de paso, temperatura, presión, densidad, energía potencial, energía cinética de rotación, energía cinética de traslación y energía total. El archivo se produce en formato de texto separado por espacios. Los datos son almacenados cada cierto número de pasos que se definen en el código del programa principal. A continuación se muestra un fragmento de este archivo de salida:

```
52  261.62  2795.6  1177.1 -6.18e-17  4.22e-18   4.1e-18 -5.35e-17
54  258.14  2831.5  1176.4 -6.17e-17  4.16e-18  4.04e-18 -5.35e-17
56  254.07  2799.1  1175.6 -6.16e-17  4.07e-18  4.01e-18 -5.35e-17
58  250.21  2764.2    1175 -6.14e-17  3.97e-18  3.98e-18 -5.35e-17
. . .
```

Donde cada una de las líneas es un registro que almacena un paso de la simulación.

El segundo archivo de salida contiene el número de partículas (átomos) de la simulación, un nombre del modelo, y las coordenadas de cada una de los átomos, así como, un identificador de elemento, de acuerdo al formato x,y,z , reconocido por algunos de los programas de representación de modelos moleculares. Algunos de ellos son de uso libre.
Este archivo es reescrito cada paso, por lo que si se detiene la simulación, se dispone del último modelo generado para ser analizado en algún otro programa de visualización molecular. En caso de que la simulación se ejecute hasta el fin se dispondrá del modelo final.

```
9
muestra 0.0
O  0.508841  1.9585  1.87861
H  0.437873  2.6503  2.53634
H  1.30861  2.16739  1.39594
O  2.68289  2.40756  0.138328
H  3.62528  2.36888  0.301544
H  2.54075  1.80448  -0.591281
O  2.03962  5.077  6.9504
H  2.79135  5.39938  7.44759
H  2.15464  4.12694  6.9305
```

El tercer archivo contiene la siguiente información:

Los tres primeros valores serán las coordenadas x,y,z del centro de masa y los cuatro restantes los valores del cuaternion (x,y,z,w). Cada molécula comprenderá un registro y estos se separarán por saltos de línea. Ejemplo:

```
5.49592e-11   2.0088e-10   1.88832e-10 -0.803344   -0.406966  -0.118869  -0.418196
2.72754e-10   2.37154e-10   1.06633e-11 0.821995   0.407178  0.103582  -0.38445
2.08802e-10   5.04165e-10   6.97678e-10 -0.925635   -0.114218  0.251898  0.258267
4.33213e-11   5.99054e-10   1.65207e-10 -0.169674   0.916123  0.255273  -0.25839
2.43519e-10   2.27608e-10   6.86119e-10 -0.371059   0.119256  -0.496887  0.775369
3.83841e-10   3.70013e-10   3.79823e-10 -0.0785509   0.435651  -0.891726  -0.0941435

     . . .
```

El último archivo generado contiene valores de las velocidades de traslación y angulares del modelo en el paso actual. Los tres primeros valores de cada registro se refieren a las velocidades de traslación en los ejes x,y,z respectivamente. Los tres valores restantes corresponden a las velocidades angulares en x,y,z. El siguiente es un ejemplo de un fragmento del archivo.

```
80.3781    37.7418   76.4819 -2.09209e+12  2.71393e+12  1.41847e+12
18.4393    -104.887   -51.7377 -8.22425e+11  2.19584e+12  -2.61003e+11
124.484    -128.957   -64.3949 1.6688e+12  2.0983e+11  1.85969e+13
-64.0824    47.3057   -96.6346 8.59779e+11  -1.19797e+12  1.44232e+11
-13.3296    12.7944   28.2258 1.19276e+12  1.75187e+12  -9.8591e+11
-41.181    -3.08075   -48.5647 1.05011e+12  9.29417e+11  -1.46059e+11
-132.307    -94.2496   98.0718 -7.98505e+11  3.09284e+12  1.04347e+12
-97.3289    -126.44   73.9839 2.29554e+12  -6.36334e+12  -3.31629e+11

. . .
```

En estos dos últimos, archivos el formato es el mismo de los archivos de entrada, y tienen como objeto poder continuar con la simulación en caso de que esta sea suspendida por el usuario, por alguna interrupción en el equipo, o bien, iniciar una simulación a partir de los datos finales de una ejecución anterior. Generalmente este tipo de programas en equipos de escritorio tardan varios días o incluso semanas corriendo con modelos reales, dependiendo entre otras cosas del número de moléculas, las características de la mismas, el número de pasos y el tamaño de paso. Para poder continuar con una ejecución interrumpida, los archivos de entrada son remplazados por estos archivos.

Todos los archivos utilizados están en formato de texto para su fácil edición, revisión y exportación a otros programas como modeladores moleculares, procesadores de palabras y gestores de bases de datos.

5.5 MODELO DE PRUEBA

Para las pruebas del sistema se utilizo un modelo de agua en estado sólido conocido como hielo III.

El agua es uno de los compuestos más estudiados dentro de la simulación de líquidos, por sus propiedades, así como sus cambios de estado[PETRENKO 1999].

Las moléculas de agua tienen las siguientes características:

Están formadas por 2 átomos de hidrogeno y uno de oxigeno. Véase **figura 5-5**.Los átomos de hidrogeno se unen al átomo de oxigeno mediante enlaces covalentes, es decir cada átomo aporta un electrón a la unión, quedando implicados 4 electrones en la unión molecular. Los núcleos de las moléculas de hidrogeno se encuentran a una distancia de 0.9572±0.0003 Amstrongs con

respecto al núcleo de la molécula de oxigeno y el ángulo H-O-H es de 104.52±0.05°.[PETRENKO 1999].

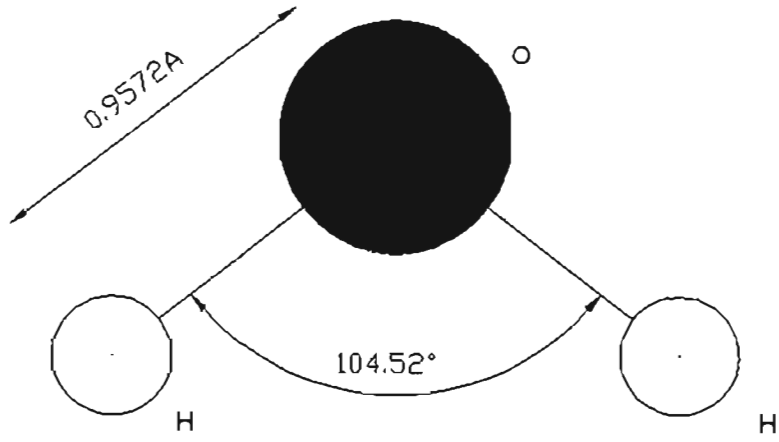


Figura 5-5 Molécula de agua.

La molécula de agua aunque tiene una carga total neutra (igual número de protones que de electrones), presenta una distribución asimétrica de sus electrones, lo que la convierte en una molécula polar, alrededor del oxígeno se concentra una densidad de carga negativa , mientras que los núcleos de hidrógeno quedan desnudos, desprovistos parcialmente de sus electrones y manifiestan, por tanto, una densidad de carga positiva, convirtiendo a la molécula de agua en bipolar. Véase **figura 5-6**.

La distancia H...O es mucho mayor que la de O-H . La molécula a la cual el protón es covalentemente ligado se denomina `donador de protón`, y a la otra ´aceptor´. Cada molécula de H_2O puede actuar como donadora para dos enlaces de hidrógeno y como aceptor para otras dos, con direcciones tetrahedricamente opuestas a los enlaces O-H. [PETRENKO 1999].

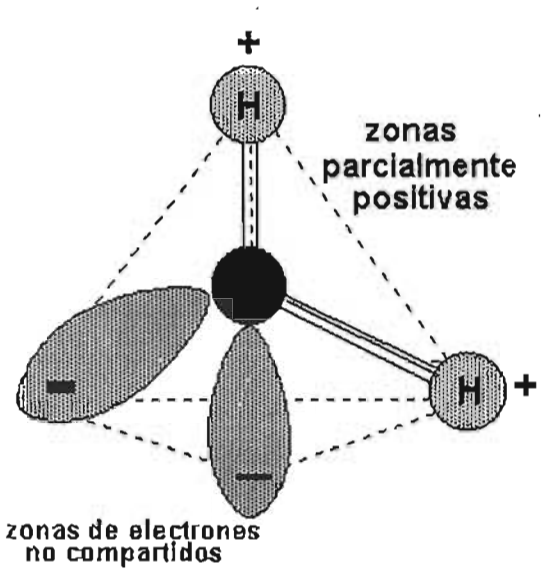


Figura 5-6 Cargas de una molécula de agua.

Al enlace H...O se le conoce como puente de hidrógeno, el hecho de que alrededor de cada molécula de agua se dispongan otras cuatro molécula unidas por puentes de hidrógeno permite que se forme en el agua (líquida o sólida) una estructura de tipo reticular. **Figuras 5-7 ,5-8 y 5-9**.

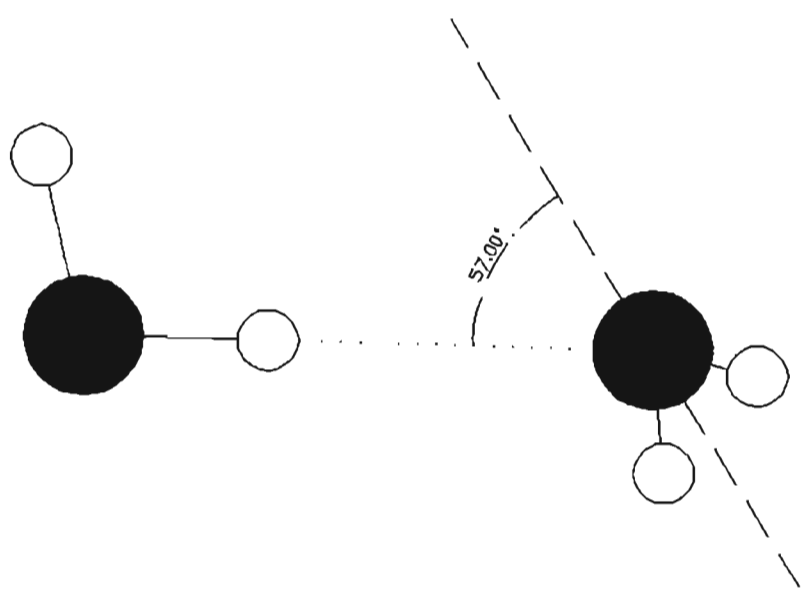


Figura 5-7. Posición de dos moléculas de agua.

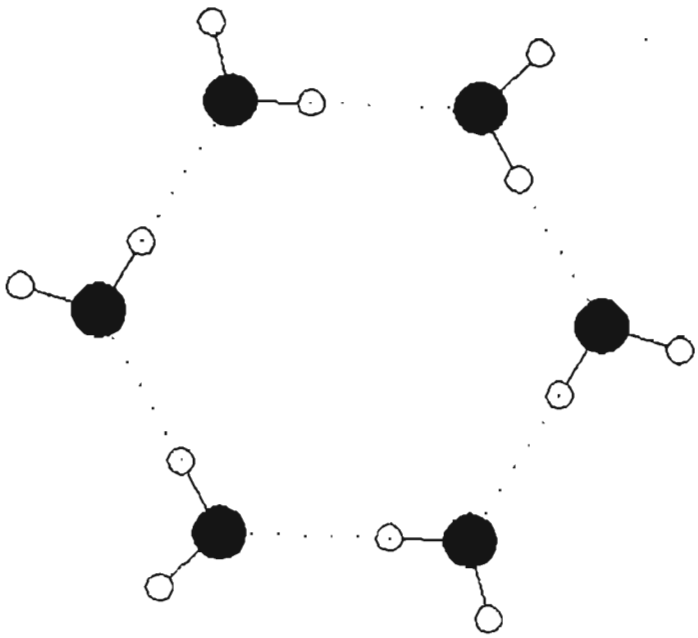


Figura 5-8 Acercamiento de estructura reticular de agua.

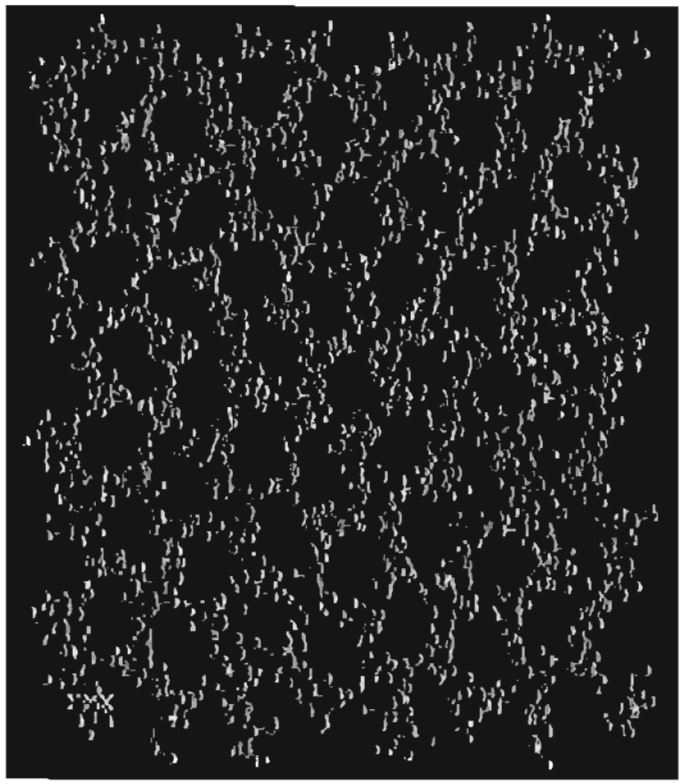


Figura 5-9 Estructura reticular de moléculas de agua.

Dadas sus características, las moléculas de agua pueden ser consideradas como cuerpos rígidos, aunque realmente presentan variaciones en su interacción intermolecular, dependiendo de su estado, el cual puede ser sólido, liquido o gaseoso.

En el caso particular del estado sólido, existen diversas "fases de sólido", que difieren unas de otras principalmente por las condiciones de presión y temperatura necesarias para su formación, siendo hielo Ih el que se obtiene congelando agua a presión atmosférica o por condensación directa de vapor de agua a -100°C[PETRENKO 1999]. Las fases del hielo son identificadas con numerales romanos que van del I al XII, en el orden cronológico aproximado con que fueron producidas de manera experimental, sin que exista necesariamente una relación entre una fase y otra.

Para efectos de simulación, existen modelos moleculares definidos. En el caso de las pruebas del modelo computacional desarrollado en esta tesis se empleo el modelo conocido como TIP4P.

En TIP4P se considera una distancia de 0.9572 Amstrongs entre el átomo de oxigeno y cada uno de los dos hidrógenos de la molécula. El ángulo formado por los dos átomos de hidrógeno es de 104.52°. Cada hidrógeno tiene una carga positiva de 0.52*e*, adicionalmente cada molécula tiene una carga negativa M, de dos veces el valor del hidrógeno, es decir, 1.04*e*; localizada en la bisectriz del ángulo HOH a 0.15 Amstrongs del átomo de oxígeno[BALTAZAR 2003].

El ensamble empleado es NPT, lo que significa que, durante la ejecución del programa, los valores de número de partículas, presión, temperatura deben ser constantes. Por tal motivo, durante la simulación el sistema realiza ajustes en cada paso sobre energía y tamaños de caja de simulación. Estos ajustes permitirán de manera gradual lograr la estabilidad de temperatura y presión deseados. Las ventanas de temperatura y presión, permiten al investigador determinar a partir de que número de pasos se obtiene la estabilidad del ensamble buscada, y si una vez alcanzada esta se llega a perder.

Como se menciono anteriormente para efectos de prueba del sistema se utilizaron modelos de hielo III, con las siguientes características y valores iniciales:

El tamaño de paso: 2.0e-15 segundos.
Número de Pasos: de 1 hasta 256 pasos
Número de Moléculas: 768 ,1200 y 1500 moléculas.
Temperatura inicial: 255.0 K
Presión: 2500.0*100000 Pa

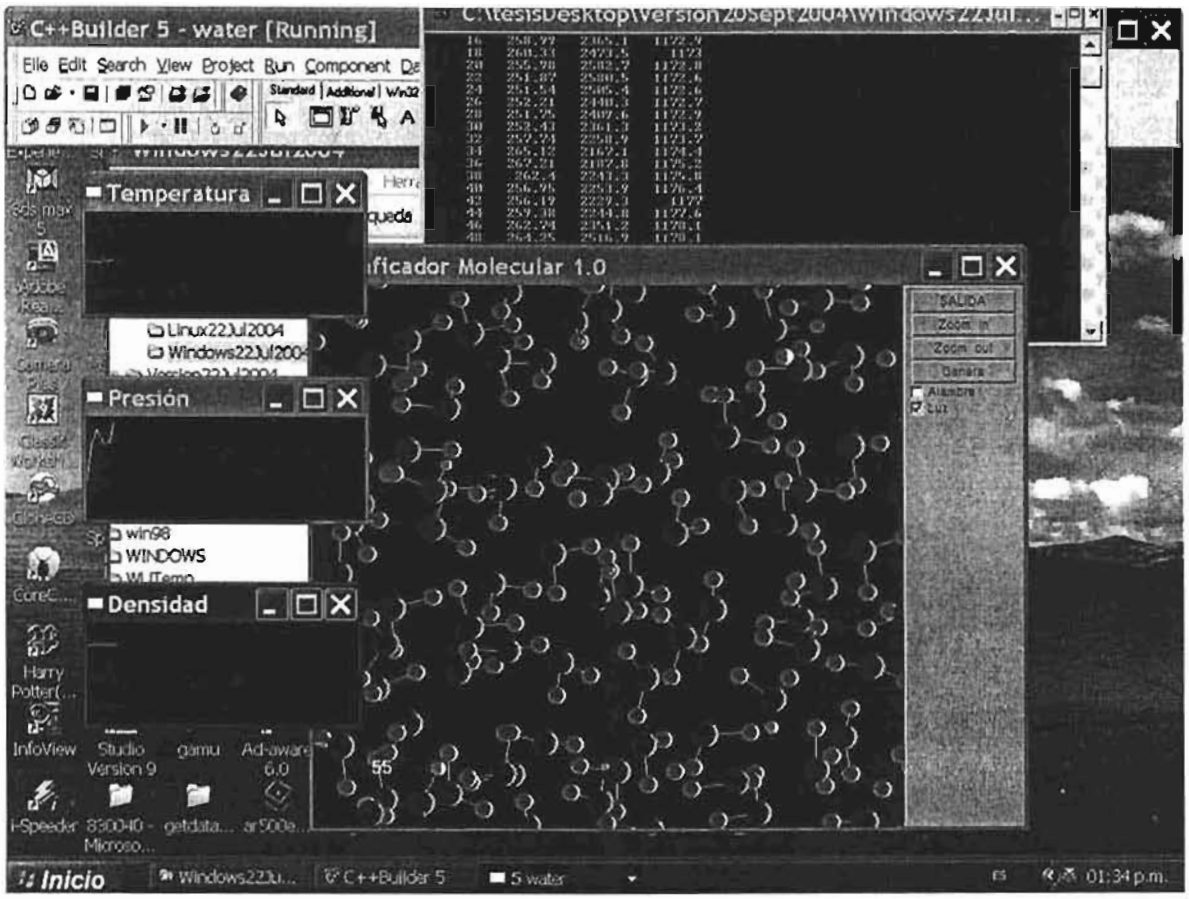


Figura 5-10 Simulación en un sistema con Microsoft Windows XP.

5.6 Resultados

El modelo fue ejecutado en 2 equipos, cada uno con sistema operativo Linux y Windows, en particiones distintas.

Equipo 1:

Sistemas Operativos:	Mandrake Linux 9.2 Núcleo 2.4.21 Microsoft Windows Millenium Edition (Me)
Modelo:	Acer Travel Mate 527TXV Laptop
Procesador:	Intel Pentium III 800 Mhz
Memoria RAM:	384 Mb. 133 Mhz
T. video	ATI Radeon. Rage Mobility-M 64 Mb. Ram compartida. AGP Integrada.
Disco Duro:	IDE 5400 RPM.

Equipo 2:

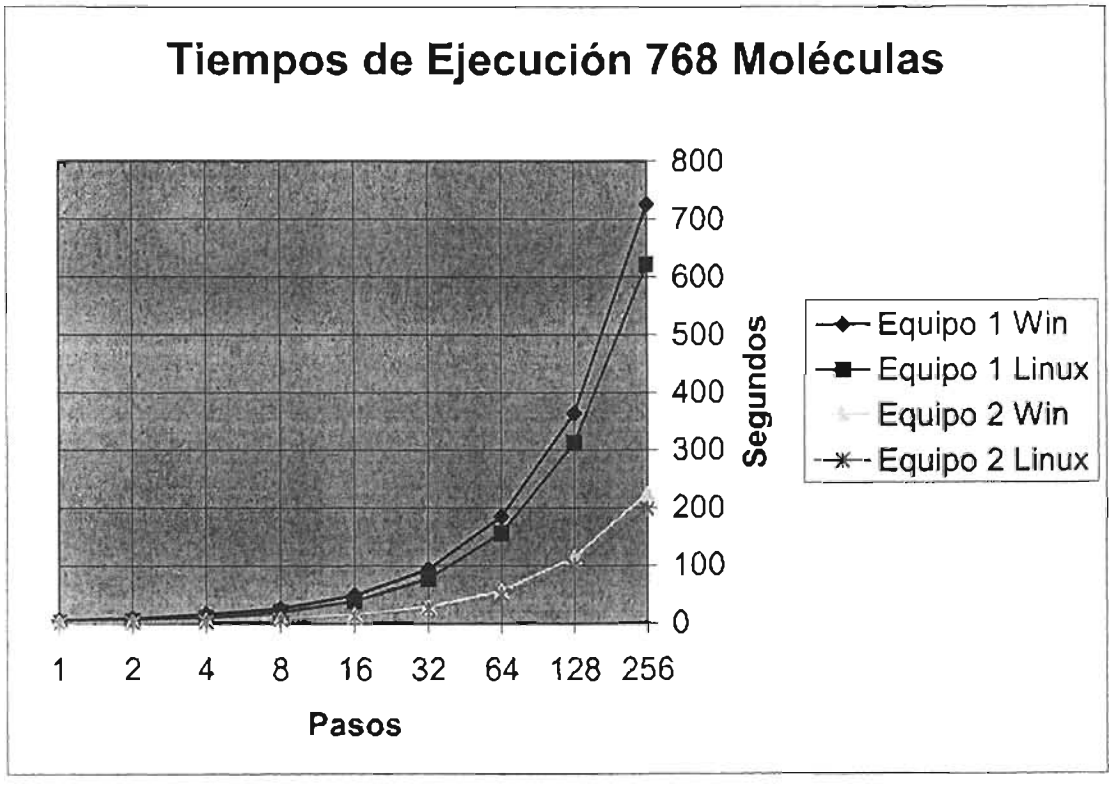
Sistemas Operativos:	Mandrake Linux 10.0 Official. Núcleo 2.6.8. Windows XP Professional SP 1
Modelo:	Ensamblada. Tarjeta Madre. MSI KT6 Delta.
Procesador:	AMD Athlon XP 2600+. 1.9 Ghz.
Memoria RAM:	256 Mb. DDR 400 Mhz
T. video	Gforce 4 MX 400. 64 Mb Ram Independiente. AGP 8X.
Disco Duro:	IDE 7200 RPM

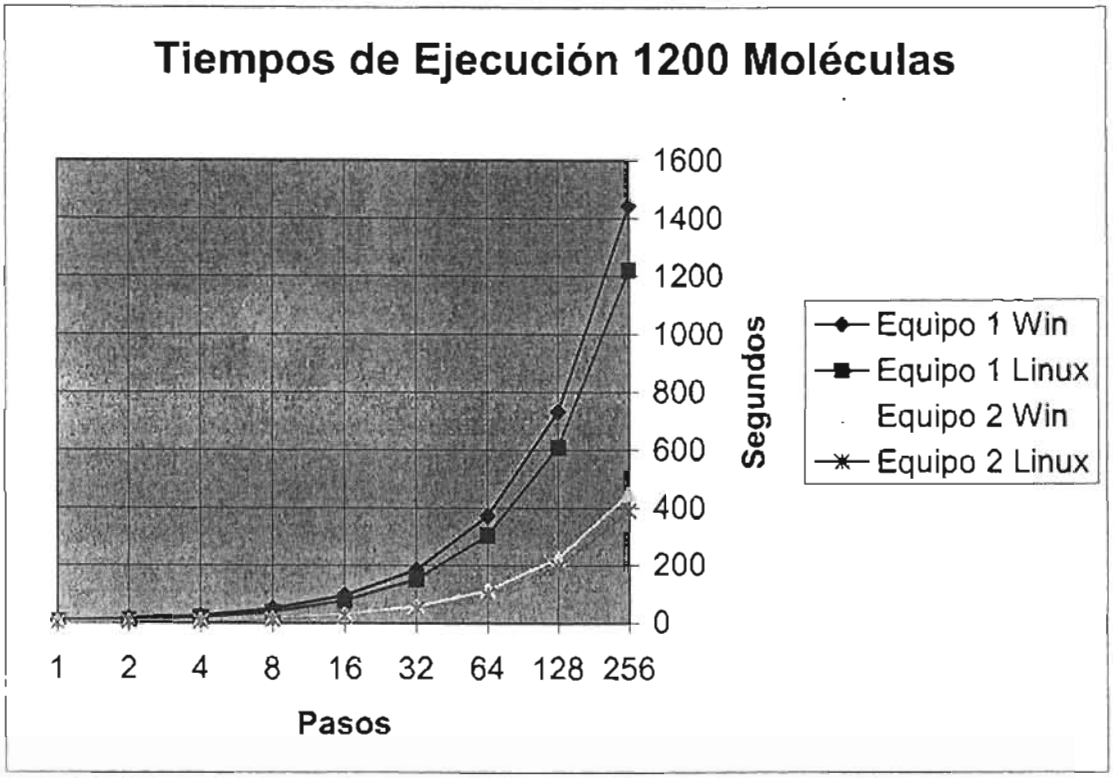
Para realizar las mediciones se empleo la instrucción **clock()** de la cabecera time.h, por estar presente en los compiladores de ambos sistemas operativos, y funcionar de la misma manera.

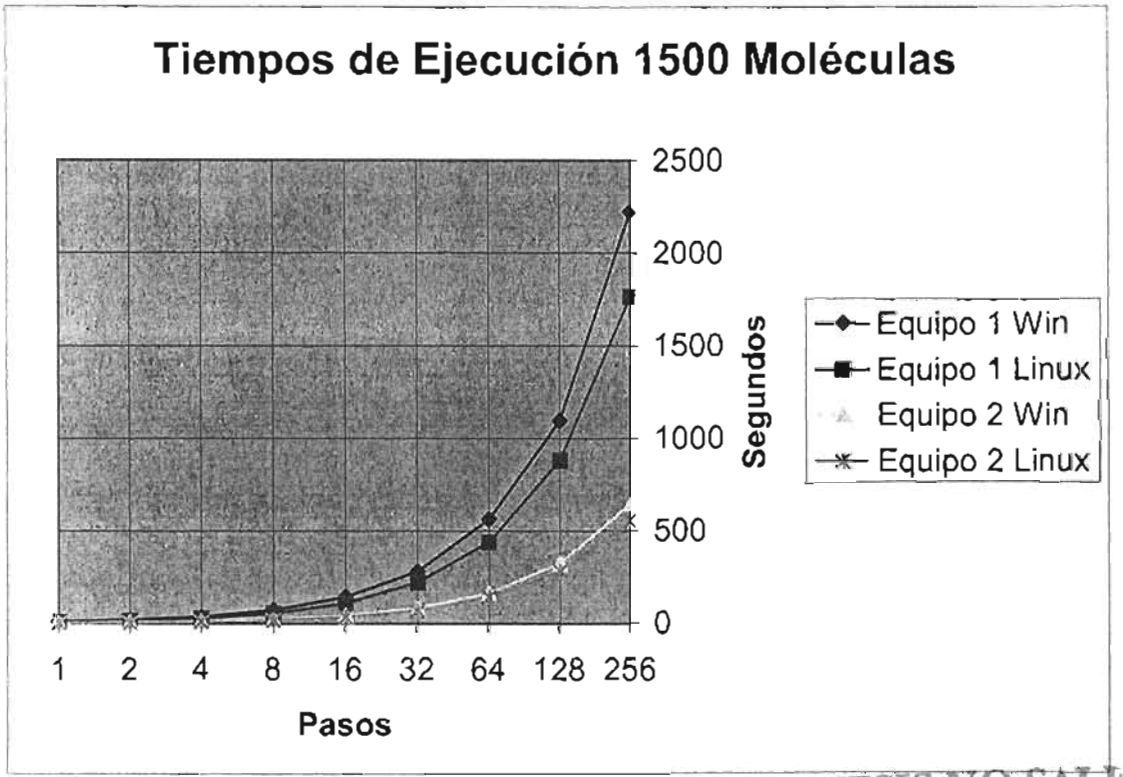
Tabla de Tiempos. Equipos 1 y 2. 768,1200 y 1500 Moléculas.

	768 Moléculas				1200 Moléculas				1500 Moléculas			
	Equipo 1 (segundos)		Equipo 2 (segundos)		Equipo 1 (segundos)		Equipo 2 (segundos)		Equipo 1 (segundos)		Equipo 2 (segundos)	
No. Pasos	Win	Linux	Win	Linux	Win	Linux	Win	Linux	Win	Linux	Win	Linux
1	6	2	2	<1	8	5	4	1	10	7	5	2
2	9	5	3	1	14	10	6	3	20	15	8	4
4	16	10	5	3	26	20	9	6	37	30	13	8
8	26	20	8	6	50	39	16	12	72	55	23	17
16	49	39	15	12	96	77	30	24	143	109	43	35
32	94	78	29	25	186	153	58	49	280	221	88	70
64	185	156	57	50	372	306	114	98	564	440	163	140
128	364	313	115	99	732	610	227	196	1096	882	325	280
256	725	622	225	198	1443	1221	450	390	2224	1764	648	556

En el caso de las corridas con Microsoft Windows, los resultados mostrados son un promedio de 5 ejecuciones, debido a las variaciones de tiempo entre ejecuciones.







CONCLUSIONES

Se desarrollo un sistema integral de Dinámica Molecular que permite la solución numérica, así como la visualización de modelo. El sistema desarrollado trabaja sobre modelos moleculares rígidos. Por tal motivo, fue empleado un modelo de agua para las pruebas del sistema.

La principal contribución de este sistema, es su integración a los actuales trabajos de investigación realizados en el Centro de Investigaciones Teóricas de la Facultad de Estudios Superiores Cuautitlán. En uso real, una simulación llega a comprender varios cientos de miles de pasos, que generalmente implican varios días de ejecución continua. La visualización del modelo, así como de las propiedades macroscópicas en tiempo real permiten al investigador la revisión y en caso de ser necesario la interrupción del proceso para realizar ajustes, sin necesidad de esperar que esté llegue a finalizar.

Se obtuvieron versiones del sistema para ser ejecutadas en sistemas operativos Microsoft Windows y Linux. El primer sistema operativo se eligió por ser ampliamente utilizado, y el segundo por ser de uso libre, soporta múltiples plataformas de hardware y de código abierto.

Para la realización de las pruebas se instalaron versiones de Microsoft Windows y Linux en los mismos equipos con los mismos recursos. Los tiempos de ejecución en sistemas Microsoft Windows fueron variables, aun corriendo el mismo modelo. Las pruebas en Linux generaron tiempos de ejecución constantes al probar varias veces el mismo modelo. Adicionalmente, se pudo observar que la interactividad y velocidad de generación gráfica fue mayor en Linux, inclusive los tiempos totales de simulación fueron menores en Linux (véase capítulo 5) .

Las versiones de Microsoft Windows probadas fueron: Windows ME y Windows XP. En el caso de Linux se utilizó la distribución Mandrake tanto con núcleo 2.4.x como 2.6.x. Para efectos de la comparación todas las pruebas y desarrollos se realizaron sobre procesadores Intel x86 o compatibles.

El sistema es compatible con equipos de escritorio, estaciones de trabajo, equipos multiprocesador, es decir todos aquellos que trabajen bajo el modelo de memoria compartida y permitan la implementación de multihilos.

En cuanto a la los requerimientos de hardware de gráficos, el sistema es capaz de funcionar en sistemas con cualquier tarjeta gráfica soportada por Microsoft

Windows; en el caso de Linux cualquier tarjeta que permita el funcionamiento del modo gráfico y el uso de OpenGL. Sin embargo, se recomienda el uso de tarjetas con aceleración 3D.

En un desarrollo futuro se planea trasladar el sistema a equipos de procesamiento distribuido y multiprocesador, en busca de paralelizar la simulación numérica, ya que se observo mediante perfilamiento que la parte de cálculos de Fuerzas consume un 96% del tiempo de ejecución del programa. Para tal propósito, podrían ser empleados equipos multiprocesador, alguno de los clusters de la Facultad o bien un Grid.

El principal reto en esta migración radica en los acceso de lectura-escritura, sincronización entre unidades de ejecución paralelas y técnicas de prueba. (véase capítulo 4).

APÉNDICES

7.1 OPENGL

7.1.1 Primitivas de Dibujo

Para la construcción de imágenes en dos o tres dimensiones dentro de OpenGL, se emplean elementos simples llamados primitivas. Las primitivas son objetos de una o dos dimensiones, que van desde puntos y líneas hasta polígonos complejos.

Un elemento clave para la formación de primitivas son los vértices (*vertex*), los cuales, indican puntos en el espacio de dos o tres dimensiones. Los vértices también son referidos como "el mínimo común denominador" de todas las primitivas de dibujo.

A partir de la especificación de sus vértices, se pueden construir primitivas que conformen gráficos más elaborados. **Figura 7.a**

OpenGL cuenta con diversas instrucciones para definir primitivas bidimensionales como triángulos, cuadriláteros y otros polígonos determinados por sus vértices. En todos los casos existen algunas reglas básicas para su construcción:

1. Todos los vértices del polígono deben estar en el mismo plano, es decir el polígono no puede doblarse ni torcerse. Razón por la cual, los triángulos son ideales para la construcción de modelos.
2. Sus bordes no deben interceptarse y el polígono debe ser convexo. Una forma simple de verificar si es convexo es trazar líneas imaginarias que lo atraviesen, dichas líneas deberán entrar y salir solamente una vez del polígono.

Estas restricciones se deben básicamente a los algoritmos que constituyen OpenGL ,los cuales trabajan de manera más eficiente al operar bajo las condiciones antes mencionadas. La mayoría del hardware de aceleración de gráficos esta optimizado para dibujar triángulos. De hecho, muchas de las pruebas de rendimiento y comparación de generación de gráficos 3D están dadas en triángulos por segundo. [WRIGHT 1999].

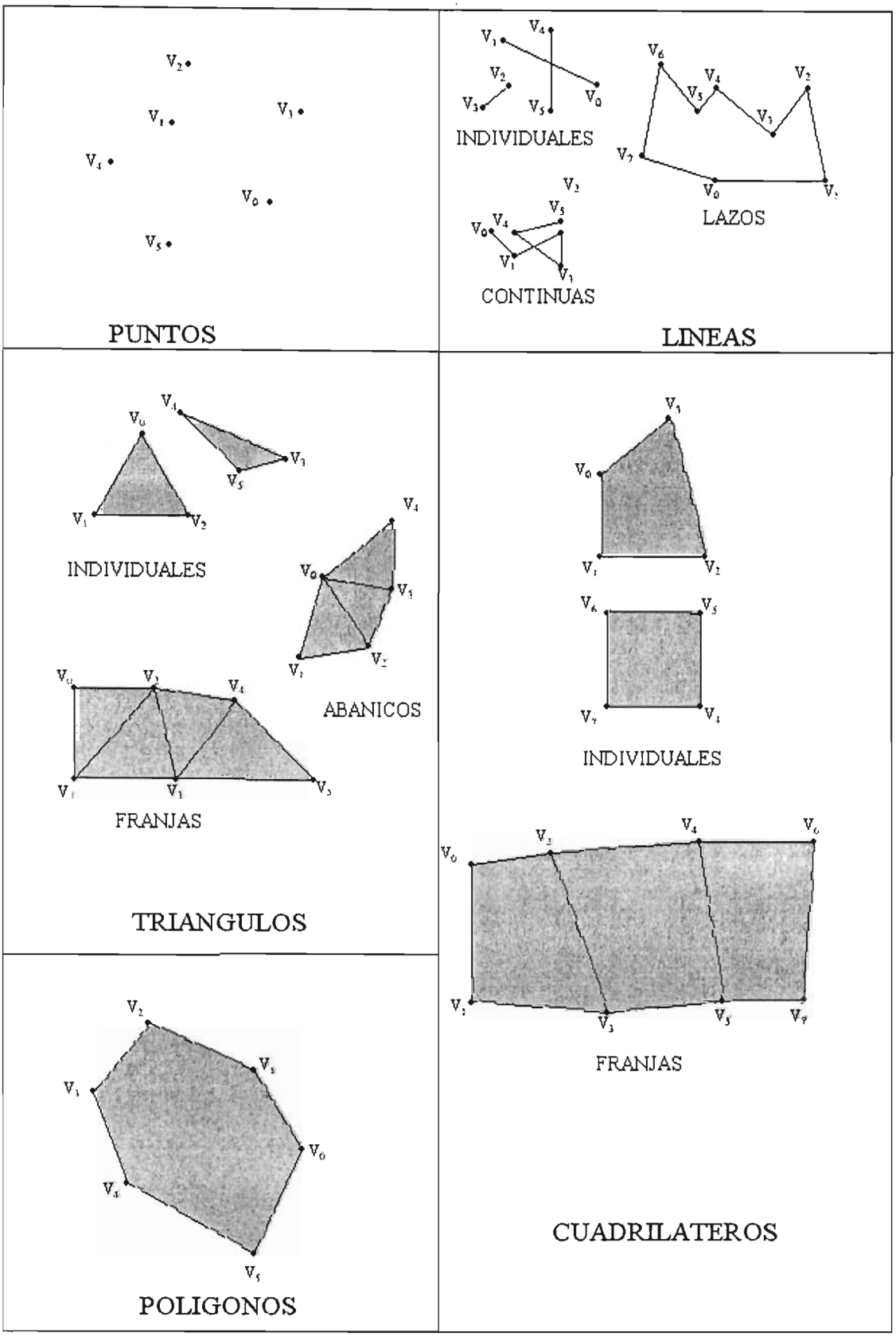


Figura 7.a. Primitivas de dibujo OpenGL.

7.1.2 Matriz De Modelado

Esta matriz de 4x4 representa el sistema de coordenadas transformado que se utilizara para colocar y orientar los objetos del modelo.

Mediante su modificación es posible realizar operaciones como traslación, escalado y rotación del modelo.

Inicialmente esta matriz esta constituida como una matriz identidad, y sus valores son modificados de acuerdo a la transformación que se quiera realizar.

Traslación
Si se considera a x, y z como coordenadas del vector de translación (desplazamiento en x, y,z) .Se necesitara multiplicar la actual matriz de modelado por la siguiente matriz

$$\begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Escaldado
Tomando a x, y, z como factores de escalado sobre sus respectivos ejes. Será necesario multiplicar la matriz de modelado actual por:

$$\begin{bmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotación
Considerando x, y, z como un punto dentro del sistema de coordenadas actual, se traza un eje imaginario de este punto al origen, y se rota el sistema en sentido contrario a las mancillas del reloj la cantidad de grados definido por A, la matriz de modelado actual deberá multiplicarse por la siguientes matrices:

En caso de rotar *a* en el eje X:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos a & -sena & 0 \\ 0 & sena & \cos a & 0 \\ 0 & 0 & 01 & \end{bmatrix}$$

Para rotar *a* en el eje Y:

$$\begin{bmatrix} \cos a & 0 & sena & 0 \\ 0 & 1 & 0 & 0 \\ -sena & 0 & \cos a & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Si se requiere rotar *a* en el eje Z:

$$\begin{bmatrix} \cos a & -sena & 0 & 0 \\ sena & \cos a & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

7.1.3 Matriz de proyección

Existen dos tipos diferentes de proyección que pueden ser utilizados, los valores de las matrices de proyección dependerán del tipo seleccionado, ya sea, Proyección Ortográfica ó Perspectiva.

Proyección Ortográfica

En este tipo de proyección la pantalla es paralela al plano xy y el punto de vista del observador es paralelo al eje z del plano de visión. Una proyección paralela se forma cuando se extienden líneas paralelas desde cada vértice del polígono hasta que dichas líneas interceptan el plano de la pantalla de la computadora. Esto tiene como consecuencia que el volumen de visualización sea una paralelípedo rectangular. La distancia a la que un objeto se encuentre sobre el eje z con respecto al punto de visualización no influye sobre el tamaño de la representación en pantalla del mismo.

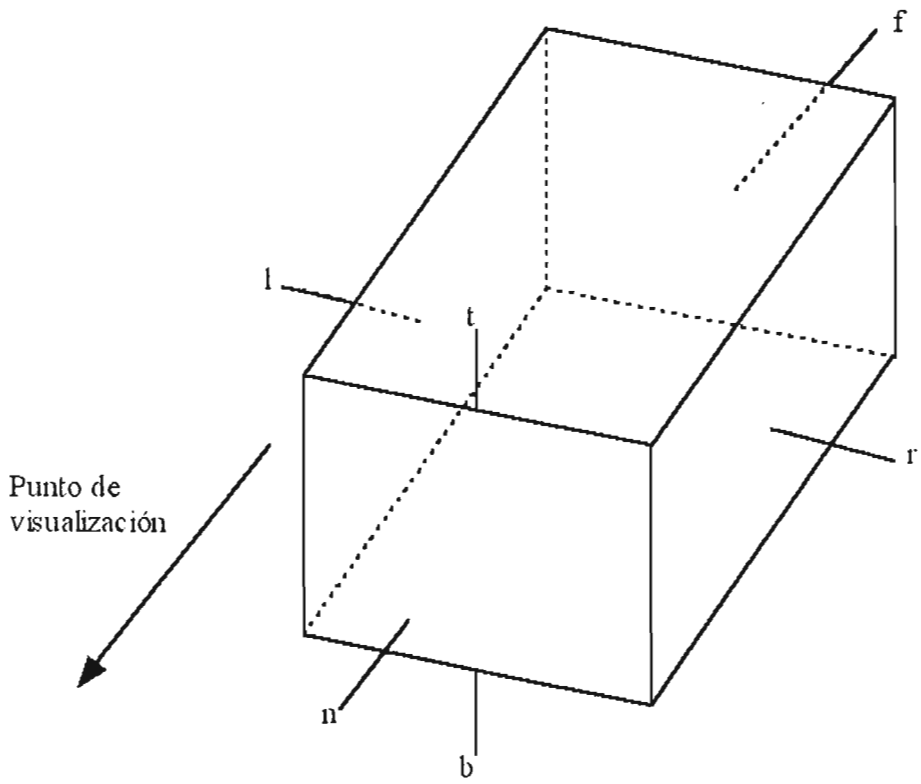


Figura 7-1Volumen de proyección ortográfica.

La matriz de proyección en el modo ortográfico esta constituida por:

$$\begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

donde

l = valor mínimo de x
r = valor máximo de x
b = valor mínimo de y
t = valor maximo de y
n= distancia desde el punto de visualización hasta el plano mas cercano del volumen de visualizacion.

f=distancia desde el punto de visualización hasta el plano mas lejano del volumen de visualizacion.
siempre que $l \neq r, t \neq b, n \neq f$

Proyección Perspectiva

En la proyección perspectiva, mientras más lejano está un objeto del observador, más pequeño se dibujará dicho objeto. Este cambio de tamaño de un objeto basado en la distancia del objeto con respecto al observador es una importante forma de dar una referencia de la profundidad de un escenario. Cuando hay muchos objetos en una escena, los tamaños relativos de objetos similares ayudan a reforzar la sensación de profundidad.

En la proyección perspectiva, las líneas de proyección no son líneas paralelas, sino líneas que convergen en un punto llamado centro de proyección. El centro de proyección es la intersección de todas esas líneas convergentes con el plano de la pantalla que determinan la imagen proyectada.

El volumen de visualización en una proyección en perspectiva es una pirámide truncada "frustrum".

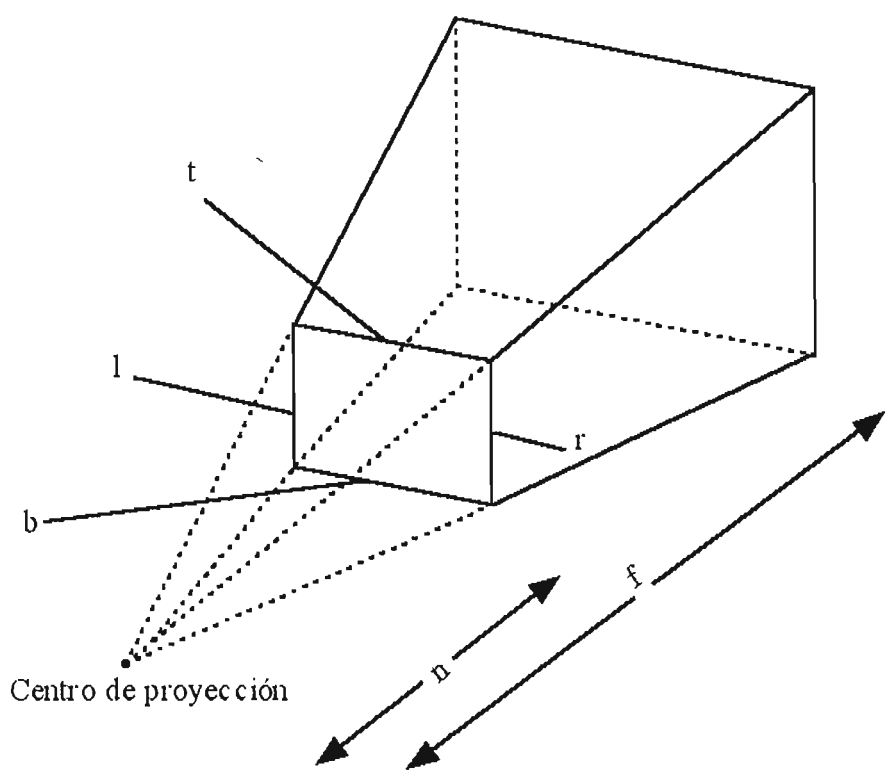


Figura 7-2 Volumen de proyección perspectiva.

La matriz de proyección en modo perspectiva se conforma de la siguiente manera:

$$\begin{bmatrix} \frac{2}{r-l} & 0 & 0 & \frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & \frac{t+b}{t-b} \\ 0 & 0 & \frac{-2}{f-n} & \frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

l= valor mínimo de x
r = valor máximo de x
b = valor mínimo de y
t = valor máximo de y
n= distancia desde el punto de visualización hasta el plano más cercano del volumen de visualización.
f= distancia desde el punto de visualización hasta el plano más lejano del volumen de visualización.

siempre que $l \neq r, t \neq b, n \neq f$

7.1.4 COLOR, ILUMINACION Y MATERIALES

Color
Dentro de OGL existen 2 modos de trabajo con respecto al color: RGBA e Indexado.

En RGBA los colores de las primitivas dibujadas se forman por una combinación de rojo, verde y azul. En valores que van desde 0 a 1. Siendo 0 la ausencia del color y 1 su máxima intensidad (0 a 255 en representación tipo byte). El número de colores o variantes de cada color dependerá de la profundidad de color definida en el sistema computo que se este utilizando, también conocida como "*ColorDepth*". Por ejemplo en un sistema con una profundidad de color de 24 bits destinará 8 bits para el rojo, 8 para el verde y 8 para el azul proporcionando un total de 2^8 variantes de cada color dando un total de 2^{24} colores distintos que van desde el negro hasta el blanco.

El cuarto valor en el modo RGBA conocido como Alpha se refiere a la opacidad y no afecta la manera en que se asigna un color de forma individual, sino la forma en que se calculan los colores al colocar objetos con diferentes valores de opacidad unos delante de otros con respecto al punto del observador o cámara dentro de la escena de OGL, así como el mezclado "*blending*" al utilizar fuentes de luz.

En el modo indexado se crea una tabla o paleta de colores a utilizar, seleccionados de todos los colores posibles del sistema. Al crear las primitivas, sus colores están dados con referencia a valores o índices de la tabla o paleta de colores. Al cambiar un color de la tabla, todos los elementos que tengan colores referenciados a ese índice se modificaran.

El número de colores de la paleta depende de la cantidad de memoria dedicada para almacenar la tabla la cual estará dada en potencias de 2 (dependiendo del número de bits utilizados para dicho propósito) , sin embargo no podrá ser mayor a la profundidad de color del sistema. La ventaja principal del uso del modo indexado es el ahorro de memoria utilizada para el color.

Independientemente del modo que se desee utilizar, este debe seleccionarse al inicio de la sesión OpenGL.

LUZ

Una fuente de luz tiene un efecto en la escena cuando existen objetos en la misma con superficies que absorban o reflejen la luz, dependerá de la propiedades de los materiales utilizados. Incluso, una superficie es capaz de emitir su propia luz, reflejar la que recibe en todas direcciones o en alguna dirección particular.

Existen 4 tipos diferentes de fuentes de luz en OpenGL: Ambiental, difusa, especular y emisiva.

Ambiental.- Este tipo de luz, ha sido reflejada un número de veces tal que es imposible determinar su origen y dirección. Cuando toca una superficie es reflejada de igual manera en todas direcciones.

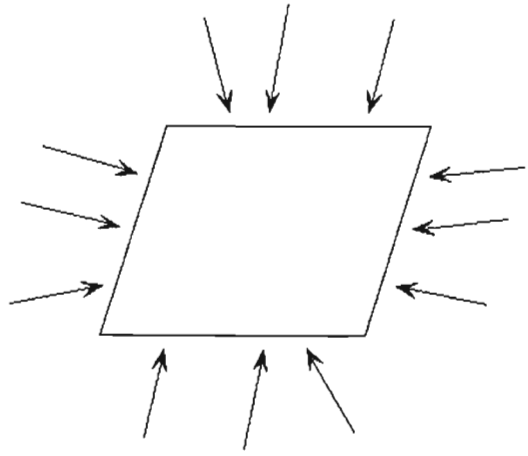


Figura 7-3 Luz ambiental.

Difusa: Es aquella que proviene de una dirección en particular, pero es reflejada en varias direcciones.

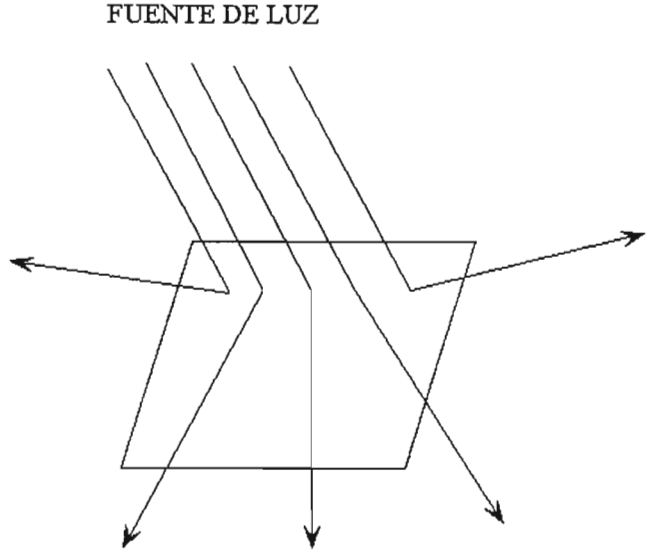


Figura 7-4 Luz difusa.

Especular.- Proviene de una dirección, y tenderá a reflejarse sobre las superficies en una dirección particular, en el caso de luces especulares intensas, estas tienden a producir manchas brillantes en el sitio donde inciden sobre los objetos del modelo.

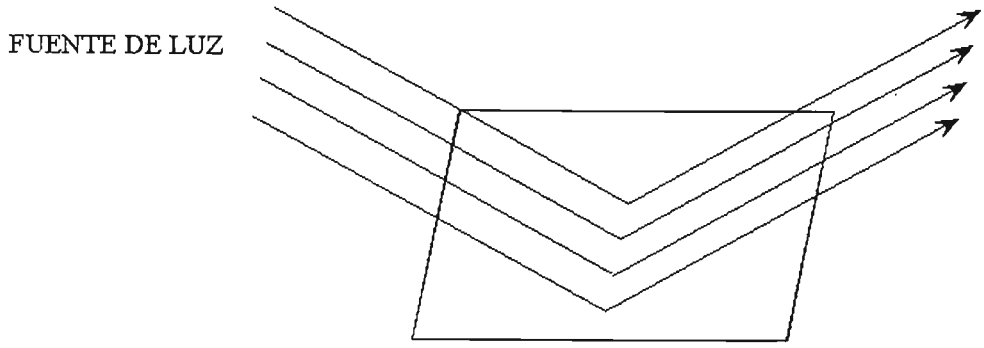


Figura 7-5 Luz especular.

Emisiva.- Se produce debido a las propiedades de ciertos materiales de ser fuentes de luz, se simula dando mayor intensidad al color del material.

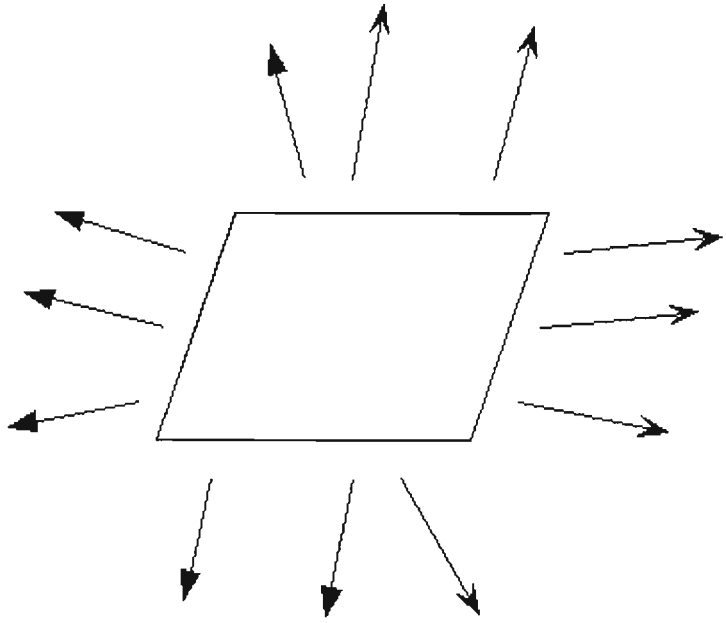


Figura 7-6Luz emisiva.

En cuanto a la ubicación de las fuentes de luz, dichas fuentes pueden clasificarse en **direccionales** cuando se encuentran colocadas a una distancia infinita de la escena; **posiciónales** cuando se ubican en un punto exacto dentro de la escena que determinará su efecto. Entre las características de las fuentes de luz posicional se encuentra la atenuación que consiste en la disminución de la intensidad de la luz emitida por la fuente de manera directa a la distancia de la misma con respecto a la superficie o vértice, determinada por su factor de atenuación FA.

$$FA = \frac{1}{k_c + k_l d + k_q d^2}$$

Donde:

d= distancia entre la fuente de luz y el vértice o superficie
k_c=Constante de Atencuación GL. (Por omisión 1.0)
k_l= Atenuación lineal
k_q= Atenuación cuadrática

En el caso de la luz direccional, el factor de atenuación (FA) es siempre 1 por lo que no tiene efecto alguno.

En lo que respecta al color de la luz emitida por la fuente se utiliza el modelo RGBA explicado en la sección de color.

Una fuente de luz tiene un efecto cuando existen objetos en al escena con superficies que absorban o reflejen la luz, dependerá de las propiedades del material que se utilice. Incluso un material es capaz de emitir su propia luz, reflejar la que recibe en todas direcciones o en alguna dirección en particular.

Al igual que las fuentes de luz, los materiales tienen propiedades de reflexión con respecto a la luz ambiental, difusa y especular. El efecto final dependerá de las fuentes de luz existentes, así como las propiedades de reflexión de los materiales existentes, los colores utilizados tanto en las fuentes de luz, los objetos y sus valores de transparencia. Siendo necesario el calculo de los valores por separado para rojo, verde y azul en el caso del modo RGBA. El valor Alpha será igual al valor Alpha difuso del material en el vértice. Debido a que el modo indexado es más reducido, la obtención de muchos de los efectos de iluminación es mas complicada o no posible obtenerlos, por lo que el Modo RGBA es utilizado preferentemente cuando se manejan valores de iluminación y fuentes de luz.

El valor final de color obtenido en un vértice en modo RGBA estará dado por:

CV= EM + AE + CT

Donde
CV = Color del Vertice
EM = Emisión del material en dicho vértice. Valor asignado al material.

AE = Luz ambiental Escalada.
CT = contribución de luz ambiental, difusa y especular de fuentes de luz con sus respectivas atenuaciones.

Luz ambiental Escalada
Se obtiene al multiplicar el valor definido como luz global ambiental de todo el modelo por la propiedad de luz ambiental asignada al material.

(Luz global ambiental) (propiedad ambiental del material)

Contribución de fuentes de luz
Cada una de las fuentes de luz puede contribuir a al color final de un vértice, el calculo se realiza de manera separada para cada fuente de luz, y finalmente es sumado.

Contribución de la fuente= Factor de Atenuación * efecto de reflector * (Termino de ambiente + Término de Difusión + Término Especular).

En el caso del efecto de reflector "spotlight" existen 3 posibilidades.

a) El valor es igual a 1 si la fuente de luz no es un reflector.
b) El valor es igual a 0 si la fuente de luz si es un reflector, pero el vértice queda fuera del cono de iluminación de la fuente.
c) $\left(\max\{v \bullet d, 0\}\right)^R$

donde:
$v=(v_x,v_y,v_z)$ vector unitario que expresa la posición de la fuente de luz.
$d=(d_x,d_y,d_z)$ dirección de el reflector.
R= exponente de reflector.

Al evaluar la expresión $\left(\max\{v \bullet d, 0\}\right)$ en caso de que el valor obtenido sea menor al coseno del ángulo de apertura del cono de iluminación, el vértice cae fuera del volumen del cono de iluminación y se aplica el valor del punto b.

El **termino de ambiente** se obtiene al multiplicar el valor de iluminación ambiental de la fuente por la propiedad de reflexión ambiental del material el en vértice.

El **término de difusión** se calcula con la expresión:

$$\left(\max\{L \bullet n, 0\}\right) * difuso_{fuente} * difuso_{material}$$

donde:
L= vector unitario que expresa la posición de la fuente de luz.
n= vector normal unitario con respecto al vértice.

Finalmente el **término especular**

$$\left(\max\{s \bullet n, 0\}\right)^{exponente\_especular} * especular_{fuente} * especular_{material}$$

Si $L \bullet n = 0$ entonces el término especular = 0.

donde:
n= vector normal unitario con respecto al vértice.
L= vector unitario que expresa la posición de la fuente de luz.
s= Suma de L y vector que expresa el punto de visualización, dividiendo cada componente entre la magnitud del vector, llegando a $s=(s_x,s_y,s_z)$.

7.2 MANEJO DE HILOS

A continuación se describen las funciones básicas para el manejo de hilos de WINAPI y Posix, aunque en ambas modalidades existen funciones avanzadas para el manejo de sincronización y control de acceso a las variables compartidas.

7.2.1 WINAPI32

WinAPI32 permite el manejo de hilos mediante la biblioteca \windows\sytem\kernel32.dll. El archivo de cabecera es winbase.h.

7.2.1.1 Creación

La función necesaria para le creación de un hilo es:

```
HANDLE CreateThread(LPSECURITY_ATTRIBUTES lpThreadAttributes,
DWORD dwStackSize,
LPTHREAD_START_ROUTINE lpStartAddress,
LPVOID lpParameter,
DWORD dwCreationFlags,
LPDWORD lpThreadId
);
```

La estructura de tipo LPSECURITY_ATTRIBUTES, determina si otros procesos pueden modificar el objeto o si un proceso hijo puede heredar un manejador de este objeto. Si **lpThreadAttributes** contiene un valor NULL, entonces el descriptor de seguridad por omisión es asignado.

El parámetro **dwCreationFlags** especifica las banderas de creación del hilo. Si se envía el valor CREATE_SUSPENDED, el hilo se creará en modo suspendido y su ejecución comenzará con el llamado de la función ResumeThread().

El parámetro **lpThreadId** apunta a una variable de tipo DWORD que recibirá el valor del identificador de el hilo (TID).

En **lpStartAddress** se envía el nombre de la función asociada al hilo.

Los parámetros **dwStackSize** y **lpParameter** contienen el tamaño de la pila y los parámetros pasados a la función asociada al hilo (lpStartAddress).

El siguiente es un ejemplo de creación de hilo.

```
main()
{
HANDLE Hilo;
DWORD Identi_Hilo;

Hilo=CreateThread(0,0,calculos,0,CREATE_SUSPENDED,&Identi_Hilo);
}
```

7.2.1.2Suspensión

La siguiente función suspende la ejecución de un hilo hasta el momento en que este sea reanudado llamando a la función de activación correspondiente.

```
DWORD SuspendThread(HANDLE hilo)
```

hilo corresponde a la variable de tipo HANDLE que contiene el identificador el hilo que fue asignado en el momento de la creación del hilo mediante CreateThread().

Si la función tiene éxito, el valor de retorno es el valor previo del contador de suspensiones. En caso contrario, es 0xFFFFFFFF.

Cada hilo posee un contador de suspensiones (con un valor máximo de MAXIMUM_SUSPEND_COUNT). Si el contador de suspensiones es mayor que cero, el hilo se suspende; en otro caso, el hilo no se suspende y se puede ejecutar. Llamar a SuspendThread hace que el contador de suspensiones del hilo especificado sea incrementado. Intentar incrementar más del máximo ese contador produce un error sin incrementarlo.

Ejemplo:

```
main()
{
HANDLE Hilo;
DWORD Identi_Hilo;
```

```
Hilo=CreateThread(0,0,calculos,0,CREATE_SUSPENDED,&Identi_Hilo);
...

SuspendThread(Hilo);
}
```

7.2.1.3 Activación

En caso de que un hilo se cree en modo suspendido o se hubiese suspendido mediante SuspendThread(), la función encargada de la activación o reactivación del hilo es ResumeThread().

DWORD ResumeThread(HANDLE hilo)

hilo corresponde a la variable de tipo HANDLE que contiene el identificador de hilo que fue asignado en el momento de la creación del hilo mediante CreateThread().

Si la función tiene éxito, el valor de retorno es el valor previo del contador de suspensiones.

Si la función falla, el valor de retorno es 0xFFFFFFFF

La función **ResumeThread** comprueba el contador de suspensiones del hilo. Si el valor del contador es 0, el hilo no está suspendido. En caso contrario, el contador de suspensiones del hilo se decrementa. Si el valor resultante es 0, entonces la ejecución del hilo se reanuda.

Si el valor de retorno es 0, el hilo especificado no está suspendido. Si el valor de retorno es 1, el hilo especificado fue suspendido pero ha sido reanudado. Si el valor de retorno es mayor de 1, el hilo especificado aún esta suspendido

Ejemplo:

```
main()
{
HANDLE Hilo;
DWORD Identi_Hilo;
```

```
Hilo=CreateThread(0,0,calculos,0,CREATE_SUSPENDED,&Identi_Hilo);
ResumeThread(Hilo);

...

SuspendThread(Hilo);

...
ResumeThread(Hilo);
}
```

7.2.2 POSIX Pthreads

7.2.2.1 Creación

Se requiere de la función pthread_create. Esta función devuelve cero si el hilo se creo con éxito, en caso contrario devuelve un valor diferente de cero.

```
#include <pthread.h>
int pthread create(pthread t *tid, pthread attr_t *attr, func *f, void *arg);
```

Esta función crea un hilo nuevo asociado con la función f. Los parámetros enviados se colocan en arg, en caso de no requerirse se envía el valor NULL. attr_t permite modificar los atributos de creación del hilo, NULL crea el hilo con los valores por omisión. El valor TID del nuevo hilo creado será colocado en tid.

Ejemplo:

```
pthread_t Identi_Hilo;
main()
{
   pthread_create(&Identi_Hilo,NULL,calculos,NULL);
}
```

7.2.2.2 Terminación

Un hilo puede terminar cuando la función asociada alcanza el final de su código, cuando se encuentra un return, o bien, cuando se emplea la función pthread_cancel:

```
#include <pthread.h>
int pthread_cancel(pthread_t tid);
```

La función regresa un valor cero cuando el hilo se canceló con éxito. El valor tid, especifica el TID del hilo a cancelar.

REFERENCIAS

1. [ALLEN 1987] ALLEN, M.P.,TILDESLEY, D.J. Computer Simulation of Liquids. Gran Bretaña. Oxford Press, 1987.

2. [BALTAZAR 2003] Baltazar Ayala, Ricardo. Simulación Dinámica Molecular de propiedades termodinámicas de fases sólidas de agua a altas presiones. México. Programa de Maestría y Doctorado en Ciencias Químicas.UNAM. 2003.

3. [BIRREL 1989] Birrel, Andrew. An introduction to programming with threads. Palo Alto,California.EUA. Digital systems research center,1989.

4. [BRYANT 2001] Bryant, Randal E., O'Hallaron. David. Concurrent Programing with Threads., Pittsburg, PA,EUA, School of Computer Science, Carnegie Mellon University. 2001.

5. [CABRERA 2003a] Cabrera Flores, Eduardo. Introducción al uso de AlphaServer SC. México, Dirección General de Computo Académico-UNAM, Semana de Supercomputo 2003.

6. [CABRERA 2003b] Cabrera Flores, Eduardo. Optimización en AlphaServer SC. México, Dirección General de Computo Académico-UNAM, Semana de Supercomputo 2003.

7. [CATALAN 2003] Catalan Colt, Miguel. El manual para el clustering con OpenMosix. Madrid, España,2003.

8. [CHANDRA 2001] Chandra, Rohit. Parallel programming in OpenMP, San Francisco, California. Morgan Kaufmann, 2001.

9. [CLARCK 1997] Clarck , Tim. Molecular Modeling Annual 1997 . Berlin Alemania, 1997, Springer, 1997.

10. [ERCOLESSI 1997] Ercolessi, Furio. A molecular dynamics primer. Italia, Internacional School for Advanced Studies, Spring college of computational physics, 1997.

11. [FOLEY 1990] Foley, James. Computer Graphics: Principles and practice. Massachusetts, EUA. Addison-Wesley, 1990.

12. [GARCIA 2003a] García Reyes, Marisol, et al. Optimización en Origin 2000. México, Dirección General de Cómputo Académico-UNAM, Semana de Supercomputo 2003.

13. [GARCIA 2003b] García Reyes, Marisol. Introducción al uso de Origin 2000. México, Dirección General de Cómputo Académico-UNAM, Semana de Supercomputo 2003.

14. [GARZON 1992] Garzón, Ignacio. Dinámica Molecular. Temas Selectos de Física Estadística. México. El Colegio Nacional. Instituto de Física UNAM. 1992.

15. [GLAESER 1998] Glaeser, Georg. Hellmuth, Stachel. Open Geometry: OpenGL+Advanced geometry. New York, EUA. Springer 1998.

16. [GOLDSTEIN 1980] Goldstein, Herbert. Clasical Mechanics. EUA. Addison-Wesley. 1980.

17. [GORDILLO 2004]. Gordillo Ruiz, José Luis. Programación de aplicaciones numéricas paralelas.México. Dirección General de Servicios de Computo Académico-UNAM. 2004.

18. [GUTIERREZ 2001] Gutierrez, Gonzalo. Elementos de simulación computacional: Dinámica molecular y método de Monte Carlo. Santiago de Chile. Universidad de Santiago de Chile. Departamento de Física, 2001.

19. [GROGONO 1999] Grogono, Peter. The Evolution of Programming Languages. Canada. Concordia University, Department of Computer Science Quebec. 1999.

20. [HAILE 1992] Haile J.M, Molecular Dynamics Simulation, EUA, Jhon Wiley & Sons, 1992.

21. [HARLOW 1999] Harlow, Eric, Developing Linux Applications, EUA, New Riders, 1999.

22. [HERRERA 2003] Herrera Revilla, Ismael. Modelación Computacional. Columna Vertebral de la Ingeniería Avanzada. Seminario de Modelación Computacional. Universidad Nacional Autónoma de México. Instituto de Geofísica. México 2003.

23. [HILL 1990] Hill, Francis. Computer Graphics. New York,EUA. Macmillan, 1990.

24. [HOLLINGWORTH 2001] Hollingworth, Jarrod, et al. C++ Builder Developer's Guide. EUA, Sams Publishing, 2001.

25. [INPRISE 2000a] Insprise Corporation, Borland© C++ Builder 5 for Windows 2000/98/95/NT.Quick Start. EUA, Insprise Corporation. 2000.

26. [INPRISE 2000b] Insprise Corporation, Borland© C++ Builder 5 for Windows 2000/98/95/NT. Developer's Guide. EUA, Insprise Corporation. 2000.

27. [KHEIR 1988] Kheri, Naim. Systems Modeling and Computer Simulation. EUA. Marcel Dekker, 1988.

28. [KILGARD 1996a] Kilgard, Mark. OpenGL. Programming for the X Window System. EUA, Addison Wesley, 1996.

29. [KILGARD 1996b] Kilgard, Mark. The OpenGL Utility Toolkit (GLUT) Programming Interface. API version 3, Silicon Graphics, Inc., 1996

30. [LAW 2000] Law, Averill. Kelton David. Simulation modeling and Analysis. EUA. McGraw Hill, 2000.

31. [LEWIS 1992] Lewis, Theodore. Introduction to parallel computing. New Jersey, EUA. Prentice Hall, 1992.

32. [LEWIS 1996] Lewis, Bil., Threads primer : a guide to multithreaded programming , Mountain View, Calif. : SunSoft Press, c1996

33. [LIBERTY 2001] Liberty, Jesse. Horvath, David. Aprendiendo C++ para Linux en 21 Dias. Pearson Education, México, 2001.

34. [NORTHRUP 1996] Northrup, Charles, Programming with UNIX Threads,EUA, Jhon Wiley & Sons, 1996.

35. [NORTON 1996] Norton, Scott J., Thread time : the multithreaded programming guide, Upper Saddle River, New Jersey : Prentice Hall, 1996.

36. [PETRENKO 1999] Petrenko, Victor. Physics of Ice. EUA, Oxford University Press, 1999.

37. [PRESS 1992] Press, William. Numerical recipies in C. The art of scientific computing. Cambridge. Cambridge University, 1992.

38. [RADEMACHER 1999] Rademacher, Paul. GLUI. A GLUT Based User Interface Library. EUA,1999

39. [RAPPAPORT 1998] Rapoport, D.C., The art of molecular dynamics simulation, EUA,Cambridge University Presss, 1998.

40. [REFSON 2001] Refson , Keith, Moldy User's Manual. Oxford, Inglaterra. Department of Earth Sciences Parks Road, 2001.

41. [RIUS 2000] Rius Alonso, Carlos. Curso de Modelación Molecular. México. Universidad Nacional Autónoma de México. Facultad de Quimica, 2000.

42. [QUINN 1994] Quinn, Michael. Parallel computing: Theory and practice. New York, EUA. McGraw Hill, 1994.

43. [SEVERANCE 2001] Severance, Frank. System modeling and simulation. EUA. Wiley, 2001.

44. [SHREINER 1999] OpenGL Architecture Review Board, Shreiner, Dave, editor. OpenGL Reference Manual, 3ª ed. EUA, Addison Wesley, 1999.

45. [STALLINGS 1995] Stallings, William. Operating systems. 2ª Ed. EUA. Prentice Hall, 1995.

46. [TANEMBAUM 1992] Tanembaum, Andrew. Modern Operating Systems. New Jersey, EUA. Prentice Hall, 1992.

47. [VAN 2004] Van Roy, Peter. Concepts, techniques, and models of Computer Programming. EUA. MIT Press, 2004.

48. [VERA 2003] Vera, Martin. Métodos Numéricos de la Simulación Computacional. Seminario de Modelación Computacional. Universidad Nacional Autónoma de México. Instituto de Geofísica. México 2003.

49. [WALL 2000] Wall, Kurt, Programación con ejemplos, Buenos Aires: Prentice Hall, 2000.

50. [WOO 1999] Woo, Masson. Neider, Jackie. OpenGL Programming Guide . 3ª Ed., EUA, Addison Wesley , 1999.

51. [WRIGHT 1999] Wright, Richard. Sweet, Michael. OpenGL SuperBible. 2ª. Ed. ,EUA, Waite Group Press, 1999.