TESIS DE MAESTRIA

MODELADO BASADO EN FISICA: TECNICAS DE INTEGRACION FLEXIBLES Y REUTILIZABLES

José David Alanís Urquieta
Facultad de Ciencias de la Computación
Benemérita Universidad Autónoma de Puebla

Dra. Blanca Bermúdez Juárez
Facultad de Ciencias de la Computación
Benemérita Universidad Autónoma de Puebla

Dedicatoria y Agradecimientos

A Dios por haberme permitido cumplir todos mis sueños

A la BUAP y a la FCC por todos los beneficios recibidos.

Especialmente a mi esposa:
Dulce María Castro Coyotl
Gracias por su amor y apoyo incondicional, siempre.

A mi madre:
Gloria F.S. Urquieta Jímenez

A mis hermanos:
Claudia Fabiola,Carlos Daniel y Diana Rosa

A la Familia Castro Coyotl

A mis hijas María José y María Itzel

A mis amigos de la maestría:
Miguel Rodríguez Hernández, Hilario Salazar Martínez, Liset Fraguela Cuesta
Rosa María Rosas, Javier Horacio Pérez Ricardez

A mis mejores amigos
Ricardo Ruiz Rodríguez y Tomás Balderas Contreras

A mis asesores: Dra. Blanca Bermúdez Juárez, por todas las facilidades otorgadas y su apoyo total, Dr. Gustavo Rodríguez Gómez del INAOEP, por todo su apoyo, al Dr. Daniel Vallejo Rodríguez por sus consejos, y a la M.C. Leticia Mendoza Alonso por todo el apoyo brindado durante mi estancia en FCC.

A mis profesores

Especialmente a:
Dra. Lourdes Sandoval, por su preocupación e interés en mi,
Dra. Gladys Linares Fleites, Dra. Darnes Vilariño Ayala, Dr. Santiago Domínguez Domínguez, Dr. Héctor Jímenez Salazar, Dr. Carlos Celaya Borges, Dra. María Auxilio Osorio, Dr. Alejandro Rangel Huerta, Dr. Raúl Founier (+), también un agradecimiento especial al **Dr. Roberto Contreras Juárez** por su valiosa ayuda en la realización de este trabajo y molestarse en leerlo, y a todos los que de una u otra forma contribuyeron a la obtención del grado.

Indice General

Capítulo 1
Introducción

El propósito de esta tesis es el de simular con visualización el gobernador centrífugo de Watt, utilizando el modelado basado en física, con técnicas númericas que describan el comportamiento de dicha válvula ó gobernador, y usando el lenguaje Java3D, como herramienta de programación, todo esto sustentado en las bases de la mecánica clásica y la teoría de control pertinentes.

La simulación computacional es una rama tanto de las matemáticas aplicadas como de la computación aplicada a las ciencias y a la ingeniería, además de conformarse de elementos como el computo matemático, el análisis numérico, etc[2].

Actualmente la simulación de fenómenos físicos ha cobrado importancia puesto que existe la necesidad de abatir costos no sólo en tiempo sino también en espacio y en dinero. La simulación resuelve estos problemas con un método científico y cada vez más dependiente de la computadora.

La simulación puede darse de dos formas, cuando se habla de simulación computacional: una describiendo el comportamiento del fenómeno a simular mediante el uso de gráficas ó esquemas y la otra utilizando visualización. Este último enfoque es el que utilizamos en este trabajo[7].

Los pasos para simular se mencionan con un algoritmo cíclico que comienza con la observación del fenómeno físico de la realidad que se quiere simular, despúes ese mismo fenómeno se delimita (ya que no es posible abarcarlo en su totalidad), y entonces se describe en términos matemáticos, esto es lo que se llama modelo del sistema. El modelo del sistema establece un conjunto de ecuaciones generalmente integro-diferenciales que lo describen (pueden ser sistemas de ecuaciones más sencillos, para problemas sencillos y viceversa).

Es en este momento donde se utilizan los métodos para solucionar de forma numérica los sistemas antes mencionados. Se experimenta con los métodos numéricos y se elige al que más se adecúa a las necesidades del modelo determinado.Una vez que se tienen los resultados numéricos se programan los métodos en un lenguaje y se efectúa la visualización de los resultados.Por último realizan las pruebas a la visualización ya mencionada, y dependiendo del resultado de estas se regresa al principio del proceso que se describió.

Tomando en consideración lo anterior, se aplica el método de simulación a la válvula gobernadora de Watt, partiendo del hecho de que pertenece a la teoría de control por retroalimentación.

Primeramente la teoría de control se ha constituido, más allá de la física, como una de las áreas más importantes del saber humano[13]. Es una ciencia que estudia el comportamiento de las variables de un sistema físico y sus cambios, y que tiene su máxima expresión en el concepto de retroalimentación.

Precisamente es en la teoría de control donde se inserta nuestro estudio. El control por retroalimentación es un objeto de estudio desde tiempos remotos. No fue sino hasta el siglo XIX que se formalizó toda una teoría al respecto[13]. A este estudio le sucedieron científicos rusos e ingenieros que trabajaron por encontrar métodos y mecanismos para la predicción del funcionamiento y modelación matemática de los sistemas en cuestión.

Uno de los problemas clásicos en la teoría de control por retroalimentación es la válvula gobernadora de Watt. Esta válvula puede controlar diversos dispositivos que necesitan del vapor para su funcionamiento. También se le ha llamado gobernador debido al hecho de que a través de esta se puede controlar de forma absoluta el funcionamiento de un mecanismo completo[23].

Este gobernador empezó a presentar problemas debido a diversos factores que afectaban su eficiencia, esto propicio que el científico ruso Vichnegrdski[14] realizará un cuidadoso estudio enunciando en su tesis doctoral una solución elegante al problema hacia el siglo XIX, además de obtener un sistema que describía la estabilidad del gobernador y de las condiciones para el mismo, así mismo, se constituyó junto con Liapunov[13][14] en el padre de la teoría de control moderno por restroalimentación.

En la época actual, como ya se dijo, es necesaria la simulación de este y otros dispositivos, debido a que algunos representan riesgos para el ser humano, otros son demasiado costosos para admitir errores, etc. De esta forma es como se utiliza de forma conveniente la simulación.

Por lo anterior, el problema de simular la válvula gobernadora de Watt ha sido realizado por científicos y desarrolladores, expertos en mecatrónica e inclusive ingenieros que han simulado este problema valiéndose de herramientas de software diversas[22] como C, C++, Matlab entre otros. Logrando incluso resultados bastante cercanos a la realidad[13]. En este punto es donde encontramos una motivación para este trabajo.

Esta tesis utiliza una técnica denominada: modelado basado en física para la realización de la simulación con visualización del gobernador centrífugo de Watt. Esta técnica sigue los pasos del algoritmo cíclico para simular que ya se mencionaron. La física está presente en el análisis previo al conjunto de ecuaciones que dan como resultado el modelo, y es imprescindible para comprender estos fenómenos, más aún, el modelado basado en física se ha constituido como un pilar de las técnicas de simulación y modelado en animaciones de películas, juegos, gráficas por computadora, etc[1].

En el Capítulo 2, se describe de forma general en el contexto de la teoría de control, y lo sitúa dentro de la misma. Como resultado de esto obtenemos un modelo (sistema de ecuaciones diferenciales) que describe su funcionamiento.

En seguida el Capítulo 3 se desarrolla la solución numérica y mediante Java3D describiendo tanto la forma como se aplicaron los métodos numéricos, así como su programación en el lenguaje mencionado.

Se continúa en el Capítulo 4 con los resultados obtenidos al utilizar el simulador que se desarrolló, además se comentan las pruebas a los métodos numéricos utilizados y al software en general.

En el siguiente capítulo (Capítulo 5) se mencionan las conclusiones y trabajos futuros, así como las dificultades en el desarrollo del software que se realizó.

Finalmente se concluye con dos apéndices:

- Apéndice 1: Contiene una justificación del porqué utilizar Java3D y al mismo tiempo ofrece una descripción y comparación con OpenGL.
- Apéndice 2: Es un pequeño manual del usuario que sirve como guía para el manejo del simulador.

Capítulo 2
Planteamiento del Problema

En el presente capítulo se hace una descripción general del problema que ocupa a esta tesis: El regulador centrífugo de Watt, de tal forma que se conozcan los antecedentes del mismo y así situarlo dentro de un campo específico de conocimiento: la Teoría de Control y además dentro de los métodos de la misma.

Comenzamos describiendo de forma general la teoría de control, introduciéndose a las nociones y principios básicos, para continuar con una estructura de los problemas que a esta teoría ocupan. Posteriormente se llega a las nociones de control por retroalimentación y finalmente se habla acerca de los métodos de la teoría de control.

Después se narra brevemente la historia del control automático, colocando en primer término los antecedentes, continuando con la historia e hitos necesarios para la comprensión del mismo y terminando con la descripción de la teoría matemática del control.

Una vez que se ha situado el problema en la teoría de control dentro del contexto teórico e histórico y que se ha descrito de forma general dentro de la misma, podemos continuar en la segunda parte con la descripción formal del problema en lenguaje matemático ó sea la teoría matemática del control, lo cual se logra mediante dos herramientas que son útiles para tal fin: Teoría de la Estabilidad y las Ecuaciones Diferenciales.

Resumiendo esperamos obtener una ubicación del problema en el marco teórico que se plantéa, además mediante la aplicación de los métodos de la teoría de control obtener una descripción formal. Esto nos servirá de entrada para aplicar técnicas computacionales que nos lleven a una simulación del comportamiento de la válvula mediante la visualización de los resultados.

2.1 Introducción de la Teoría de Control

La teoría de control, es una disciplina científica que se ocupa del estudio de la evolución de las variables involucradas en el control de un proceso o sistema

determinado, prácticamente de cualquier tipo. La teoría moderna de control se puede definir como una rama de la teoría de sistemas que se ocupa del cambio de comportamiento de un sistema complejo mediante acciones externas.

Un dispositivo de contol es un método por el que se consigue que una variable (o conjunto de variables) evolucione de una forma preestablecida, ya sea porque las mantiene constantes, ó porque hace que evolucionen según un criterio dado.

Existe una diferencia importante entre el dispositivo físico de control y el sistema matemático que lo regula, el primero es real, existe y es suceptible de error, mientras que el segundo solamente describe las situaciones que deberián sucederse una a otra, con un margen de error que sólo se puede conocer si se tiene el sistema físico para experimentar[13][22].

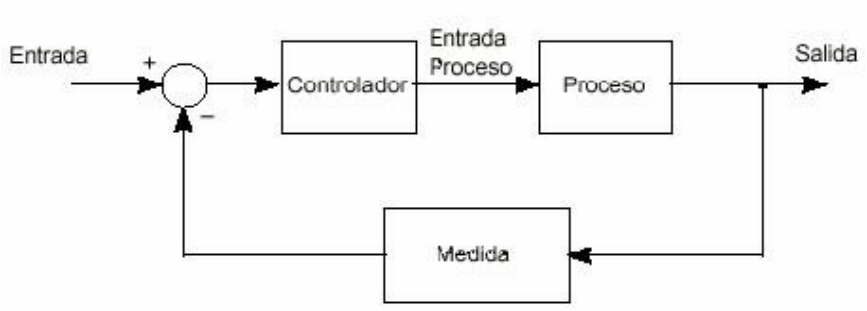

Figure 2.1: Tipo de Problema de la Válvula Gobernadora de Watt

2.2 HISTORIA DEL CONTROL AUTOMÁTICO

La teoría de control es una disciplina práctica. Los desarrollos clave de la historia de la humanidad que han afectado al desarrollo del control por realimentación han sido(para mayor referencia veáse [13]):

1. La preocupación de los Griegos y Árabes por controlar de forma precisa la evolución del tiempo. Esto representa un período comprendido entre los años 300 A.C. hasta 1200 D.C.

2. La Revolución Industrial en Europa, Generalemente se sitúa su comienzo en el tercer cuarto del siglo XVIII; sin embargo, sus raíces pueden encontrarse ya en los años de 1600.

3. El comienzo de las comunicaciones de masas y la Primera y Segunda Guerra Mundiales. Esto representa el período entre 1910 y 1945.

4. El comienzo de la era espacial y del computador 1957.

En un punto intermedio entre la Revolución Industrial y las Guerras Mundiales, hubo un desarrollo extraordinariametne importante. La teoría del control comenzó a adquirir su lenguaje escrito, el lenguaje de las matemáticas.

J.C. Maxwell proporcionó el primer análisis matemático riguroso de un sistema de control por retroalimentación ó feedback en 1868.

De esta forma, y en cuanto a su formulación matemática, se puede dividir la historia del desarrollo matemático de la teoría de control en varias etapas:

1. Prehistoria del control automático: sería el perióod anterior a 1868 y los primeros años del siglo XX el período primitivo del control automático.

2. Según Friedland, podemos llamar al período comprendido entre 1868 y los primeros años del siglo XX el período primitivo del control automático.

3. Es habitual llamar al período desde entonces hasta 1960 el período clásico;

4. Y el período posterior a 1960 hasta el presente el período moderno.

A través de la historia como podemos ver hay una evolución de los sistemas de control.

La máquina de vapor alcanzó su madurez en 1783 con la venta del primer prototipo de máquina de vapor del escocés James Watt. Dicha máquina, ya adeucada para su uso industrial, fue diseñada mucho antes del primer enunciado de la primera Ley de la Termodinámica, que data de 1842. El incentivo principal para su desarrollo era evidentemente el deseo de introducir una máquina motriz en la molienda del grano. Usando la máquina de vapor rotaroria, el molino de vapor de Albion comenzó su operación en 1786.

En 1784, Boulton y Watt se involucraron en la construcción de un molino que funiconase con ayuda de una máquina de vapor.

La salida del péndulo era aplicada a un dispositivo que presionaba hacia abajo las piedras del molino, para mantener constante el espaciado entre las piedras, a pesar de variaciones en su velocidad de giro, y asegurar de esta forma una molienda fina. Boulton y Watt se dieron cuenta de que el péndulo centrífugo podía servir como un medio para controlar de forma directa la velocidad de la máquina de vapor.

Todo lo que necesitaba era aplicar la salida del péndulo centrífugo a la presión de entrada a los pistones de la máquina a través de la válvula de entrada. De hecho, Boulton y Watt acababan de mejorar el diseño de esta válvula, de tal forma que fue sencillo adaptar a dicha válvula el péndulo centrífugo.

A finales de 1788, el regulador que había sido diseñado fue instalado en la máquina de vapor. De esta forma el controlador de velocidad de los molinos de viento se convirtió en el regulador de velocidad de la primera máquina de vapor de la Era Industrial, para convertirse despúes en todo un símbolo de la rama del saber dedicada al control automático.

Alrededor de 1790 en Francia, los hermanos Périer desarrollaron un regulador flotante para controlar la velocidad de una máquina de vapor, pero su técnica no igualaba los resultados del regulador centrífugo, por lo que fue pronto suplantado.

Finalmente, en 1799 William Murdock inventó el distribuidor de vapor, que permite el control automático de las válvulas, así como la excéntrica, que simplifica el proceso de transformación del movimiento de ascenso y descenso del émbolo en otro giratorio.

2.2.1 Nacimiento de la Teoría Matemática de Control

Los ingenieros del siglo XIX encontraron experimentalmente que cuando el "feedback" del regulador centrífugo aumentaba (esto es, se reduciá más y más la presión de vapor para un incremento fijo de la velocidad de salida), se alcanzaba un punto donde el regulador dejaba de controlar, y la salida (velocidad) comenzaba a oscilar. El regulador se volvía inestable. El regulador, diseñado para superar el efecto de las perturbaciones externas, podía, bajo ciertas circunstancias, generarlas. El responsable de este cambio de comportamiento es el "tiempo de respuesta" finito desde que el regulador detecta que se genera la salida de retroalimentación. Este tiempo de respuesta hace que la reatroalimentación negativa (cuando las perturbaciones a la salida se reducen) pueda pasar a ser retroalimetnación positiva (cuando las perturbaciones son amplificadas).

En la mitad del siglo XIX fue cuando por primera vez se analizó la estabilidad de los sitemas de control por reatroalimentación. Se le puede llamar al período anterior a esta época prehistórica de la teoría de control.

Ecuaciones Diferenciales

En 1840 el astrónomo del Observatorio Astrónomico Británico en Greenwich, B.B. Airy, desarrolló un dispositivo de realimentación para el apuntamiento de un telescopio. Desafortunadamente, Airy descubrió que debido al diseño erróneo del bucle de control automático, se introducen grandes oscilaciones en el sistema. Así Airy fue el primero en discutir las inestabilidades de sistemas de bucle cerrado y el primero en utilizar las ecuaciones diferenciales para su análisis. El uso de las ecuaciones diferenciales en el análisis del movimiento de sistemas dinámicos fue establecido por Lagrange y Hamilton.

Teoría de la Estabilidad

James Clerck Maxwell fué el primero en analizar la estabilidad del regulador centrífugo de Watt, en su trabajo de 1868.

En sí la ténica que utilizó se puede resumir a lo siguiente:

1. Linealizó las ecuaciones diferenciales de movimiento para encontrar la ecuación característica del sistema.

2. Estudió el efecto de los parámetros del sistema en la estabilidad, demostrando que el sistema es estable si las raíces de la ecuación característica tienen partes reales negativas.

Finalmente el científico ruso Vichnegradski resolvió el problema que planteaba la modificación de la estructura del regulador centrífugo de Watt y que conllevó a una falta de seguridad en su funcionamiento.

Aún cuando muchos ingenieros y científicos trataron de dar solución a este problema, el científico dió una solución especialmente sencilla y elegante.

Su memoria "Sobre los reguladores de acción directa" (1876) constituyo el punto de partida de la teoría de la regulación de las máquinas, haciendo frente a las exigencias de la práctica industrial[13][14].

Vichnegradski es considerado como el fundador de la teoría de la regulación automática, dadas las aportaciones que realizó introduciendo el concepto de "constante de tiempo del sistema", además de plantear el problema de la determinación del a ecuación característica a A. Hurwitz quien lo resolvió de forma independiente[11][14][25].

El trabajo de Liapunov fue la base de toda la teoría de control posterior. Estudió la estabilidad de ecuaciones diferenciales no lineales usando una noción generalizada de la energía en 1892[5][14].

Una vez que hemos establecido un punto de partida para la teoría de control automático por retroalimentación y que sabemos el contexto histórico y del marco teórico pasaremos a describir de manera formal nuestro problema.

2.3 Descripción Matemática

Teorema.- (de Liapunov)

Sea

$$\dot{x}^i = f^i(x^1, ..., x^n), \forall i \in N, \text{ (1)}$$

un sistema normal autónomo, y

$$\dot{\mathbf{x}} = f(x), \text{ (2)}$$

su expresión vectorial.

Supondremos que $f^i(x^1, ..., x^n), \forall i \in N$ son continuas y son derivables en el abierto Δ del espacio $x^1, ..., x^n$.

Se dice que $\mathbf{a} = (a^1, ..., a^n)$, de (2) es estable si toda solución que parte de t=0 a un punto suficientemente próximo a $\mathbf{a}$, se mantiene en el curso de su variación ulterior en el punto del entorno del punto $\mathbf{a}$.

Definición

Sea $\varphi(t, \xi)$ la solución de la ecuación (2) se dice estable según Liapunov:

1° si $\exists\rho$ suficientemente pequeño tal que para $\mid \xi - \mathbf{a} \mid < \rho$ la solución $\varphi(t, \xi)$ está determinado $\forall t > 0$

2°$\forall\epsilon > 0 \;\; \exists\delta > p$ tal que $\mid \xi - \mathbf{a} \mid < \epsilon, \forall t > 0$, sea $\mid \Phi(t, \xi) - \mathbf{a} \mid < \epsilon$.

Además es asíntoticamente estable si $\exists\sigma < p$, suficientemente pequeño y tal que para $\mid \xi - \mathbf{a} \mid < \sigma$. Entonces: Enunciaremos las condiciones suficientes de estabilidad de la posición de equilibrio para un sistema lineal homógeneo con coeficientes constantes.

Sea

$$\dot{\mathbf{x}}=\mathrm{Ax} \;\; \text{(3)},$$

una ecuación lineal homogénea con coeficientes constantes. Sea $\Psi(t, \xi)$ la solución correpondientes a los valores iniciales 0,ξ. Si todos los valores propios de la matriz A tienen partes reales negativas, $\exists\alpha, r$ tal que

$$\mid \Psi(t, \xi) \mid \leq r \mid \xi \mid e^{-\alpha t}, t \geq 0. \text{ (4)}$$

Se deduce inmediatamente de (4) que la posición de equilibrio x=0 de (3) es estable según Liapunov y también asíntoticamente estable.
Tomemos la desigualdad (4), para establecerla hagamos

$$\text{A}=(\text{a}^i_j);\ \text{L(p)}=\ (\text{a}^i_j\ -\ \text{p}\delta^i_j)\ ,$$

con el símbolo de derivación p se escribe (3) como:

$$\sum_{j=1}^{n} L^i_j(p)x^j = 0, i = 1, 2, ..., n \quad (5).$$

Sea $\text{M}^j_i(p)$ el menor de L^i_j en la matriz L(p) con el signo apropiado tal que se cumple:

$$\sum_{j=1}^{n} M^i_j(p)L^i_j(p) = \delta^k_j D(p)\ ,$$

donde D(p) es el determinante de L(p). Si multiplicamos (5) por $M^i_j(p)$ y efectuando la suma con respecto a i:

$$\begin{aligned} 0 &= \sum_{i=1}^{n}\sum_{j=1}^{n} M^i_j(p)L^i_j(p)x^j \\ &= \sum_{j=1}^{n} \delta^k_j d(p)x^j = D(p)x^k, \end{aligned}$$

y si $x = (x^1, ..., x^n)$ solución a (3) cada uno de x^i satisface $D(p)x^i = 0$,suponga-mos que las raíces de D(p) tienen partes reales negativas, la función x^iverifica:

$$\mid x^i \mid \leq \text{Re}^{-\alpha t}, i = 1, ..., n;\ \ t \geq 0$$

donde R y α son positivos ind de i, entonces se deduce que:

$$\mid x^i \mid \leq \sqrt{n}\,\text{Re}^{-\alpha t},\ \ t \geq 0.$$

Sea e_i el vector unitario que corresponde a la i-esima coordenada:

$$e_i = (0, ..., 1, ...0),$$

y sea $\Psi_i(t)$ solución a (3) para un valor inicial e_i tal que:

$$\Psi_i\,(0) = e_i, i = 1, ..., n;$$

por lo tanto la solución $\Psi((t)\,,\xi)$ de la ecuación (3) corresponde al valor inicial:

$$\boldsymbol{\xi} = (\xi^1, ..., \xi^n),$$

se escribe:

$$\Psi\,(t, \xi) = \sum_{i=1}^{n} \xi^i \Psi_i\,(t) \quad (6),$$

dado que:

$$|\Psi_i\,(t)\ | \leq n\,\text{Re}^{-\alpha t}\ (\text{t}\geq 0)\ \text{cumple (4)}.$$

La estabilidad según Liapunov de la posición de equilibiro x=0 resulta de (4), en efecto si ϵ es un número positivo dado, basta que $\delta = \epsilon/r$. Igualmente la estabilidad asintótica se deduce de la misma desigualdad.

Función de Liapunov

Con el fin de establecer un criterio de estabilidad de la posición de equilibiro para el sistema no lineal (1) se usa de la derivación respecto a un sistema de ecuaciones.

Sea

$$F(x^1, ..., x^n) = f(\mathbf{x}),$$

una función de las variables $x^1, ..., x^n$, definida y diferenciable en Δ. Su derivada con relación a t, "respecto al sistema de ecuaciones(1)", en el punto $\mathbf{x}$=($x^1, ...,$ x^n), se define como sigue:
Sea $\varphi(t)$ solución de (2) que verifica para cierto $t = t_0$ la condición inicial:

$$\varphi(t_0) = \mathbf{x}.$$

La derivada $F_1(x)$ respecto a (1) esta dada como:

$$\dot{F}_1(x) = \frac{d}{dt} F\left(\varphi(t)\right) \mid_{t=t_0}$$

ó la expresión de la diferencial total:

$$\dot{F}_1(x) = \sum_{i=1}^{n} \frac{\delta F(x)}{\delta x^i} f^i. \quad (7)$$

Esta última expresa que $F_{(1)}(x)$ no depende de $\varphi(t)$ sino que se determina por la selección de **x.** Ahora bien, sea $\varphi(t, \xi)$ la función $\varphi(t, \xi)$ verifica:

$$\varphi(t, \varphi(s, \xi)) = \varphi(s + t), \xi) \quad (8).$$

Propiedades de las formas cuadráticas definidas prositivas y la construcción de la función de Liapunov Sea

$$x = \left(x^1, ..., x^n\right) (12), \ (9)$$

un vector variable del espacio.

Se llama *forma cuadrática* a la función W(x) del vector x definida por la fórmula:

$$W(x) = \sum_{i,j=1}^{m} w_{ij} x^i x^j,$$

donde $w_{ij} = wji \in R$ y es definida positiva si para x$\neq$ 0 entonces $W(x) \succ 0$.

Para toda forma cuadrática positiva W(x) $\exists \mu$y v tal que $\forall \mathbf{x}$ se verifica:

$$\mu \mid x \mid^2 \leq Q(\mathbf{x}) \leq v \mid x \mid^2, \ (10)$$

de donde deduce que para todo vector arbitario **x**,se tiene la desigualdad:

$$\mid x^i \mid \leq \frac{1}{\mu} W(x). \ (11)$$

Construcción de la función de Liapunov:

Sea

$$\dot{x}^i = \sum_{j=1}^{n} a_j^i x^j, i = 1, ..., n \quad (12)$$

un sistema lineal homógeneo de ecuaciones con coeficientes constantes tal que todos los valores propios de la matriz A=(a_j^i) tienen partes reales negativas.

Existe entonces una forma cuadrática defnida positiva W(x) cuya derivada con respecto al sistema (12) satisface:

$$\dot{W}_{(12)}(\mathbf{x}) \leq -\beta W(\mathbf{x})$$

en la que **x** es un vector arbitario y $\beta > 0$ independiente de **x.**

Supongamos que (12) es la expresión de la ecuación vectorial y sea $\Psi(t, \xi)$ la solución de (3) correspondiente a los valores iniciales 0,ξ, se tiene según (6):

$$\Psi t, \xi) = \sum_{i=1}^{n} \xi^i \Psi_i(t). \quad (13)$$

Hagamos:

$$W(\xi) = \int_{i,j=1}^{\infty} |\Psi(\tau, \xi)|^2 \, d\tau (14).$$

Se tiene por (13) :

$$W(\xi) = \sum_{i,j=1}^{n} \xi^i \xi^j \int_0^{\infty} (\Psi_i(\tau), \Psi_j(\tau)) \, d\tau. \quad (15)$$

Finalmente enunciemos el teorema de Liapunov.

Teorema de Liapunov

Sea **a** =$(a^1, ..., a^n)$ una posición de equilibrio del sistema autónomo (1):

$$\mathrm{x}^i = f^i(x^1, ..., x^n), i = 1, ..., n.$$

Hagamos:

$$x^i = a^i + \Delta x^i, i = 1, ...n; \quad (16)$$

luego las nuevas funciones desconocidas son:

$$\Delta x^1, ..., \Delta x^n. \quad (17)$$

Efectuando un cambio de variables en (16) y desarrollando en serie de Taylor los segundos miembros se tiene que:

$$\Delta\dot{x}^i = f^i(a) + \sum_{j=1}^{n} \frac{\delta f^i(a)}{\delta x^i}\Delta x^i + R^i,\ i = 1, ..., n,\ (18)$$

donde R^ies el infinitesimo de segundo orden respecto a (17).

Como **a** es una posición de equilibrio (1) entonces:

$$f^i(\mathbf{a}) = 0.$$

Luego hagamos:

$$a^i_j = \frac{\delta f^i(a)}{\delta x^j} \qquad (19)$$

que se puede escribir en la forma:

$$\Delta\dot{x}^i = \sum_{j=1}^{n} a^i_j \Delta x^j + R^i, i = 1, ..., n.\ (20)$$

Teorema

Si todos los valores propios de la matriz A=$\left(a^i_j\right)$tienen partes reales negativas, la posición de equilibrio del sistema (1) **a** es asintóticamente estable; con mayor presición si $\exists\sigma > 0$ suficientemente pequeño tal que $\mid \xi - \mathbf{a} \mid < \sigma$ entonces:

$$\mid \varphi\,(t, \xi) - \mathbf{a} \mid \leq r \mid \xi - \mathbf{a} \mid e^{\alpha t},\ (21)$$

en la que r y α son positivos e independientes de ξ.

2.3.1 El regulador centrífugo (Investigaciones de Vichnegradski)

El regulador centrífugo (Fig. 2) está formado por una barra vertical S, capaz de girar sobre su eje. En el extremo superior de la misma están articuladas dos varillas idénticas L_1 y L_2 provistas de cargas iguales en sus extremos (veáse [14]).

Las varillas L_1 y L_2 están sujetas por articulaciones complementarias, de modo que sólo pueden separarse de su posición de equilibrio simultáneamente y con el mismo ángulo φ.

Las varillas se hallan en un mismo plano vertical ligado rígidamente a la barra S.

Cuando S gira con velocidad angular $\theta = \dot{\varphi}$ y las varillas L_1 y L_2 se han separado del ángulo φ de la posición vertical, forman un ángulo sobre cada una de las cargas sobre las que actuán la fuerza centrífuga:

$$m\dot{\varphi}^2 sen\varphi.\ \ (1)$$

Luego, sobre cada masa, actúa una fuerza

$$\mathrm{W}{=}mg.\ (2)$$

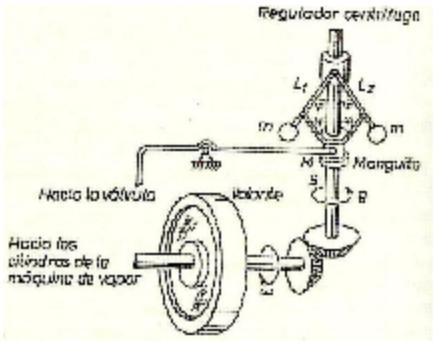

Figure 2.2: Regulador de Watt

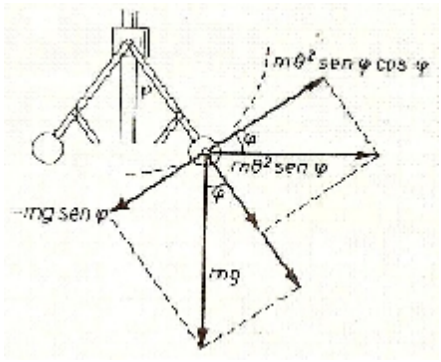

Figure 2.3: Fuerzas en el Regulador de Watt

A lo largo de la varilla L_i las fuerzas están en equilibrio por la reacción de la varilla en el enlace L_i. Para calcular la fuerza se descompone en dos,una coincidente con la varilla y otra perpendicular y orientada en sentido creciente del ángulo φ (Fig. 2).

A continuación se explica de forma simplificada el funcionamiento del regulador centrífugo. Si observamos, para una velocidad angular dada $\dot{\varphi}$,las varillas L_1 y L_2 se separan bajo la acción de las fuerzas (1) y (2) un ángulo φ, determinado por la ecuación:

$$m\dot{\varphi}^2 sen\varphi\cos\varphi - mgsen\varphi = 0, \qquad (3)$$

obtenida de igualar (3) la suma de las fuerzas (1) y (2) a cero.

Además las varillas se consideran con masa despreciable, y cuyo momento de inercia no afecta al modelo encuestión, y que además estan unidas a otras varillas complementarias que se observan en la parte inferior unidas al manguito especial que tiene una fuerza resistente del tipo viscoso, hacia la acción que la

válvula y que hace que el sistema se halle en un estado de equilibrio, tanto para mantener las bolas en su lugar e impedir que salga de su "orbita" , como para deslizar el manguito y transmitir el ángulo necesario, para su posterior regulación[14][22][23].

Otros factores que se creen descartados como el rozamiento y la irregularidad de la marcha se tratan a continuación.

Bajo la acción de la fuerza (3),la masa m realiza un movimiento descrito por una ecuación diferencial.Además de (3), la fuerza de rozamiento actúa sobre las articulaciones.

Esta fuerza simplificada y considerándola proporcional y de sentido contrario a la velocidad de φ se expresa como:

$$\text{-b}\dot{\varphi}, \quad (4)$$

donde b es constante y dado que φ indica la posición tenemos para φ la siguiente ecuación diferencial:

$$m\ddot{\varphi} = m\dot{\varphi}^2 \sin\varphi\cos\varphi - mg\sin\varphi - b\dot{\varphi}, \quad (5)$$

con φ y $\dot{\varphi}$ suponiéndolas constantes, y aún cuando estas no fueran constantes ocasionadas por la inercia y momento de inercia ejercidos por las uniones de las varillas (que ya se mencionaron), estas se complementan entre sí obligando a que las articulaciones muevan a las varillas sobre le mismo plano, lo cual hará cierta la ecuación antes mencionada.

La máquina de vapor se puede reducir a un volante cuyo momento de inercia es J; luego la ecuación diferencial se puede escribir como:

$$J\dot{\omega} = P_1 - P,$$

donde: ω es la velocidad angular de rotación, P_1 es el momento de la fuerza del vapor, P es el momento del peso de la jaula sobre el volante.

Por otra parte el manguito M del regulador estará conectado a la válvula que controla la admisión del vapor; entonces:

$$P_1 = F_1 + k(\cos\varphi - \cos\varphi^*), \quad (6)$$

donde φ^* es un valor "medio" de φ próximo al cual debe mantenerse la magnitud φ sometido a regulación; F_1 es la fuerza del vapor para $\varphi = \varphi^*$ y $k \succ 0$ coeficiente constante de proporcionalidad.

Como se deduce de (6) hay una relación de proporcionalidad inversa entre la acción del regulador y la máquina de vapor.

Al detectarse que el funcionamiento del regulador no proporcionaba los resultados esperados, Vichnegradski[6][14] hizó un estudio riguroso de la dinámica del sistema máquina-regulador y su estabilidad, como se enuncia a continuación:

Según (5) y (6):

$$\begin{cases} m\ddot{\varphi}=m\dot{\varphi}^2\sin\varphi\cos\varphi\text{-}mg\sin\varphi\text{-}b\dot{\varphi} \\ J\dot{\omega}=k\cos\varphi\text{-}F, \end{cases} \quad (7)$$

donde F=P-F$_1 + k\cos\varphi^*$es una magnitud que depende de la carga. La primera ecuación es de segundo orden. Para llevar (7) a la forma normal, e introduciendo Ψ se tiene $\Psi = \dot{\varphi}$ por lo cual (7) se transforma en:

$$\begin{cases} \dot{\varphi}=\Psi \\ \dot{\Psi}=n^2\omega^2\sin\varphi\cos\varphi\text{-}g\sin\varphi\text{-}\frac{b}{m}\Psi \\ \dot{\omega}=\frac{k}{J}\cos\varphi\text{-}\frac{F}{J} \end{cases} \quad (8)$$

Así el sistema (8) se debe solucionar de forma númerica para describir el comportamiento real de la válvula gobernadora de Watt.

Cabe mencionar que la entrada de vapor del volante hacia la jaula es inversamente proporcional al ángulo, mientras mayor ángulo se mida, menor es la entrada de vapor y viceversa.

2.3.2 Simulación de la Válvula Gobernadora de Watt

Ahora tomaremos en cuenta el sistema anterior para analizar la solución de dicho sistema y las implicaciones que conllevarán a su solución de forma númerica para ser trasladada a la computadora.

Igualando a cero los segundos miembros de las relaciones (8) y resolviendo las ecuaciones obtenidas, se obitnen las coordenadas de la posición de equilibrio:

$$\begin{cases} \Psi_0 = 0 \\ \cos\varphi_0 = \frac{F}{k} \\ n^2\omega_0^2 = \frac{g}{\cos\varphi_0} \end{cases} \quad (9)$$

Para un régimen de funcionamiento normal, la velocidad ω es constante, para una carga P(o sea F constante) y la válvula está móvil; luego φ es constante. La solución del sistema (8) debe ser de la forma:

$$\varphi = \varphi_0, \Psi = 0, \omega = \omega_0,$$

esto es la posición de equilibrio del sistema, y luego el problema es hallar su estabilidad. Igualando a 0 los lados derechos de (8) y resolviendo las ecuaciones, obtenemos:

$$\begin{cases} \Delta\dot{\phi} = 0(a) \\ \Delta\dot{\Psi} = n^2\omega_0^2\cos 2\phi_0\Delta\phi + n^2\omega_0 sen2\phi_0\Delta\omega - g\cos\phi\Delta\phi(b) \\ \Delta\dot{\omega} = -\frac{k}{J}sen\phi_0\Delta\phi(c) \end{cases} \quad (10)$$

Hagamos: $\phi = \phi_0 + \Delta\phi, \Psi = \Psi_0 + \Delta\Psi, \omega = \omega_0 + \Delta\omega$; sustituyendo y linealizando las ecuaciones de (8) obtenemos el siguiente sistema:

$$\begin{cases} \Delta\dot{\phi} = 0 \\ \Delta\dot{\Psi} = n^2\omega_0^2\cos 2\phi_0\Delta\phi + n^2\omega_0 sen2\phi_0\Delta\omega - g\cos\phi\Delta\phi \\ \Delta\dot{\omega} = -\frac{k}{J}sen\phi_0\Delta\phi \end{cases} \quad (11)$$

Sustituyendo en 10 b) el valor de la magnitud de $n^2\omega_0^2$ de (9) se obtiene:

$$\Delta\dot{\Psi} = -\frac{g\sin^2\varphi_0}{\cos\varphi_0}\Delta\varphi - \frac{b}{m}\Delta\Psi + \frac{2g\sin\varphi_0}{\omega_0}\Delta\omega.$$

El polinomio característico del sistema lineal de ecuaciones halladas para $\Delta\varphi, \Delta\omega, \Delta\Psi$ es:

$$\mathrm{D(p)}=\begin{vmatrix} -p & 1 & 0 \\ -\frac{g\sin^2\varphi_0}{\cos\varphi_0} & -\frac{b}{m}-p & \frac{2g\sin\varphi_0}{\omega_0} \\ -\frac{k}{J}\sin\varphi_0 & 0 & -p \end{vmatrix}$$

Calculando el determinante y despúes de multiplicar por -1, obtenemos:

$$\text{-D(p)} = \mathrm{p}^3 + \frac{b}{m}p^2 + \frac{g\sin\varphi_0}{\cos\varphi_0}p + \frac{2kg\sin^2\varphi_0}{jw_0}.$$

Según el Teorema de estabilidad para los polinomios de tercer grado (n=3) tenemos que[5][13][14]:

Sea $\mathrm{L(p)} = a_0p^3 + a_1p^2 + a_2p + a_3; a_0 > 0$, de coeficientes reales, L(p) es estable si y sólo sí los números a_1, a_2, a_3 son positivos y además se verifica la desigualdad:

$$a_1a_2 > a_0a_3.$$

Aplicando este teorema a -D(p) tenemos que:

$$\frac{b}{m}\frac{g\sin^2\varphi_0}{\cos\varphi_0} > 1\frac{2kg\sin^2\varphi_0}{J\omega_0}\ ;$$

se debe cumplir para que -D(p) sea estable ó bien :

$$\frac{bJ}{m} > \frac{2k\cos\varphi_0}{\omega_0} = \frac{2F}{\omega_0}, \quad (11)$$

por lo tanto los valores propios de la matriz $\mathrm{A}=\mathrm{a}_i^j$ tienen partes reales negativas.

Por el teorema de Liapunov, (11) expresa la condición suficiente para la estabilidad del sistema máquina-regulador.

Para definir el segundo miembro de la desigualdad debe indicarse la irregularidad de la marcha de una máquina de vapor a partir de (9). Si varía $F = P - F_1 + k\cos\varphi^*$, esto es, si cambia la carga P, también varía la velocidad estable ω_0. Sabemos que $\mathrm{d}\omega_0/dP$ es la velocidad de variación de ω_0 con respecto a P. El valor absoluto $v = \left|\frac{dw_0}{dP}\right|$ se denomina irregularidad de la marcha de la máquina de vapor.

Luego por (9) tenemos:

$$\mathrm{F}\omega_0 = cte.$$

y derivando obtenemos:

$$\frac{dw_0}{dF} = -\frac{\omega_0}{2F}$$

luego:

$$v = \frac{2\omega_0}{2F},$$

i.e. la condición de estabilidad se expresa como:

$$\frac{bJ}{m} v > 1. \quad (12)$$

Basado en lo anterior Vichnegradski formuló las siguientes afirmaciones:

1° El aumento de la masa m de las bolas afecta desfavorablemente la estabilidad

2° La disminución del coeficiente de rozamiento b perjudica la estabilidad.

3° La disminución del momento de inercia J del volante afecta desfavorablemente la estabilidad

4° La disminución de la irregularidad v reduce la estabilidad.

Para facilitar a los ingenieros la comprensión de sus conclusiones y atraer la atención sobre los resultados más importantes, Vichnegradski[14] enunció sus dos famosas "tesis":

PRIMERA TESIS: El rozamiento es la característica esencial de un regulador sensible, que funcione correctamente; en pocas palabras: "sin rozamiento no existe regulador"

SEGUNDA TESIS: Los reguladores astáticos (es decir reguladores de irregularidad nula), incluso los dotados de rozamiento no deben utilizarse ; simplemente: "sin irregularidad no existe el regulador".

Debido al incremento del peso de las válvulas se utilizaron bolas cada vez más pesadas. El acabado más perfecto de las superficies de las piezas hizo disminuir el rozamiento. Las mayores velocidades de trabajo de las máquinas obligaron a reducir el movimiento de inercia J del volante. Por último, la tendencia a debilitar la dependencia de la velocidad respecto de la carga, condujo a disminuir la irregularidad de marcha.

Vichnegradski[14] recomendaba en sus tesis que se aumentase artificalmente el rozamiento (mediante un dispositivo especial) y que se incrementase la irregularidad de la marcha (haciendo variar los parámetros n y k, que dependen de la estructura de la máquina).

En este capítulo se ha descrito de manera general el problema dentro de regulador centrífugo de Watt. Ubica de dos formas el postulado anterior: un contexto histórico y dentro del marco teórico, y una descripción formal matemática en términos de ecuaciones diferenciales ordinarias, ambas son complementarias entre sí.

En la primera forma sítua al regulador centrífugo como un problema de regulación dentro de la teoría de control, además de ser un problema de "malla cerrada", es decir, por retroalimentación.

De forma histórica el pronunciamiento anterior también nos habla de que el regulador fue resuelto en el período clásico del desarrollo de la teoría de control.

Por último concluimos que este problema fue atacado dentro de un marco teórico que utiliza las ecuaciones diferenciales ordinarias y el análisis de estabilidad como herramientas para su solución.

En la segunda forma dada la teoría expuesta, se concluye el siguiente sistema de ecuaciones diferenciales ordinarias que modelan y gobiernan el regulador:

$$\begin{cases} \dot{\varphi} = \Psi \\ \dot{\Psi} = n^2\omega^2 \sin\varphi\cos\varphi - g\sin\varphi - \frac{b}{m}\Psi \\ \dot{\omega} = \frac{k}{J}\cos\varphi - \frac{F}{J} \end{cases} \quad (12)$$

Las tesis de Vichnegradski nos son útiles para la comprensión del modelo que se va a construir, y así hemos obtenido un modelo formal y una situación del problema dentro del marco teórico e histórico.

Dado lo anterior procederemos a hablar sobre la solución númerica del problema para que esto nos lleve a la simulación mediante visualización, que es el propósito del siguiente capítulo.

Capítulo 3
Solución Numérica y mediante Java3D

En este capítulo se presenta de manera general la solución númerica del problema planteado anteriormente. Dicho problema ha sido reducido a un sistema de ecuaciones diferenciales que se debe resolver. Cada una de las soluciones al sistema de ecuaciones diferenciales ordinarias del sistema mencionado representa el comportamiento de la válvula en un momento determinado de tiempo t. De esta forma la solución del sistema se instrumenta mediante Java3D.

Se comienza con una descripción general del método de Runge-Kutta orden 4, observando el desarrollo del algoritmo que conlleva a la solución de una ecuación diferencial y determinando su generalización para el caso de que exista más de una ecuación; para después hacer una descripción de su instrumentación en Java3D.

Continúa con una descripción de cómo se diseño el modelo mediante Java3D sin considerar la parte matemática del problema (esto se hace con el fin de solucionar el problema como un prototipo), dotándolo de elementos y atributos que sirvan para su unión posterior con la solución numérica del problema conformando una visualización en Java3D, y a su la simulación con los parámetros de entrada.

Finalmente se mencionan los elementos utilizados para la construcción de la interfaz de forma muy general y los aspectos que se deben de considerar para que la visualización resulte lo más real posible y cercana a la realidad.

Como resultado de este capítulo se obtiene un panorama general del cómo se realizó la instrumentación de la solución del problema para llegar a una visualización realista. Esperamos obtener algunas conclusiones importantes derivadas de la instrumentación que se ha realizado.

3.4 SOLUCIÓN NUMÉRICA

3.4.1 Método Runge - Kutta Orden Cuatro

Los métodos de Runge-Kutta aparecieron por primera vez en 1895 y los métodos explicitos fueron reconsiderados unicamente hasta los años 60[5][14].

El método más simple para la solución numérica de ecuaciones diferenciales es la regla de Euler, que parte de la siguiente fórmula:

$$y_{n+1} = y_n + hf_n.$$

Este método es un método lineal en las variables y_n y f_n, siendo un método de un solo paso, además de que no presenta dficultdades cuando queremos cambiar el tamaño de paso; pero desde luego esto se compensará con una muy mala aproximación a la solución requerida.

Los métodos multipaso alcanzan las más altas aproximaciones mediante la anticipación de linearidad con respecto a y_{n+j} y f_{n+j} , $j = 0, 1, ..., k$,pero sacrificando el formato de un solo paso[11][14].

El resultado de lo anterior es que nos permite tener un error local que tiene una estructura relativamente simple, lo cual es posible de estimar mediante el método de Milne[11]. El costo de cambiar hacia un formato de multipaso es encontrar considerables dificultades cuando queremos cambiar el tamaño de paso.

Los métodos de Runge Kutta se desarrollan a partir de la regla de Euler pero en la dirección exactamente opuesta. Los ordenes más altos son alcanzados mediante la anticipación de la forma de un solo paso, pero sacrificando la linearidad. De ahí resulta que no exista dificultad en el cambio de tamaño de paso, y por consecuencia una estructura del error local más complicada y que no exista un estimado fácil y barato comparable con el método de Milne.

Por otra parte los métodos que se usan pára la aproximación numérica como los de Taylor tienen una propiedad deseable de un error de truncamiento local de orden grande, pero la desventaja de requerir el cálculo y evaluación de las derivadas de $f(t, y)$. Esto redunda en una gran cantidad de tiempo para la solución de los problemas determinados.

En cuanto a los métodos de Runge-Kutta utilizan el error de truncamiento local de orden grande de los métodos de Taylor y al mismo tiempo eliminan el cálculo y la evaluación de las derivadas de $f(t, y)$.[5][6].

Aún más, la naturaleza de las ecuaciones diferenciales puede dar lugar a confusión entre las bondades de utilizar Runge-Kutta, sobre los de Taylor. Así el método de Runge Kutta Orden 4, es el método más utilizado en la práctica para evidenciar una diferencia significativa entre esta familia de métodos y otros.

El método en sí se basa en tomar los extremos del intervalo a,b, un número entero que indicará las iteraciones a realizar y un valor α de la condición incial. Como salida obtenemos una aproximación de w de y en los (N+1) valores de t.[14]

A continuación se plasma el método de Runge-Kutta (orden cuatro)[6]:

$Paso1 : Tomar\ h = (b-a)/N;$
$t = a$
$w = \alpha$
$Salida\ (t, w),$
$Paso2 : Para\ i = 1, 2, ..., N\ seguir\ lospasos3 - 5$

$Paso3 : Tomar\ K_1 = hf(t, w);$
$K_2 = hf(t + h/2, w + K_1/2);$
$K_3 = hf(t + h/2, w + K_2/2);$
$K_4 = hf(t + h/2, w + K_3/2);$
$Paso4 : Tomar\ w = w + (K_1 + 2K_2 + 2K_3 + K_4)/6;$ (*se realiza el cálculo de* w_i)
$t = a + ih$ (Calcular t_i)
$Paso5 : Salida\ (t, w)$
$Paso6$: $PARAR$

La derivación de tales métodos invariablemente asumían un problema escalar y era tacitámente supuesto que ninguno de los significados podría cambiar cuando estos métodos fueran aplicados a los sistemas.

Los métodos para resolver sistemas de ecuaciones diferenciales de primer orden son simplemente una generalización de los método para una sola ecuación diferencial de primer orden.

Sea un sistema de orden m de problemas de valor incial de primer orden se puede expresar de la forma:

$$\begin{aligned}
\tfrac{du_1}{dt} &= f_{1(t,u_1,u_2,...,u_m)} \\
\tfrac{du_2}{dt} &= f_{2(t,u_1,u_2,...,u_m)} \\
&\vdots \\
\tfrac{du_m}{dt} &= f_{m(t,u_1,u_2,...,u_m)},
\end{aligned}$$

para $a \leq t \leq b$, con las condiciones iniciales:

$$\begin{aligned}
u_1(a) &= \alpha_1 \\
u_2(a) &= \alpha_2 \\
&\vdots \\
u_m(a) &= \alpha_m
\end{aligned}$$

Tomamos ahora el método de Runge-Kutta orden cuatro que se utiliza para resolver el problema de valor inicial de primer orden de la forma:

$$y' = f(t, y),\ a \leq t \leq b\ ,\ y(a) = \alpha,$$

el cual se puede genralizar de la siguiente forma:

Escogemos un entero $N \succ 0$ y tomamos $h = (b-a)/N$ para dividir el intervalo $[a,b]$ en N para cada $j = 0,1,...,N$ e $i = 1,2,....,m$; esto es, w_{ij} aproximará a la i-esima solución del sistema anteriormente planteado en el j-ésimo punto de red t_i. Para las condiciones iniciales, tomamos:

$$\begin{aligned} w_{1,0} &= \alpha_1 \\ w_{2,0} &= \alpha_2 \\ &\vdots \\ w_{m,0} &= \alpha_m \end{aligned}$$

Si suponemos que los valors $w_{1,j}, w_{2,j}, ..., w_{m,j}$, han sido calculados, obtenemos $w_{1,j+1}, w_{2,j+1}, ..., w_{m,j+1}$ calculando primero:

$k_{1,i} = hf\left(t_j, w_{i,j}, w_{2,j}, ..., w_{m,j}\right)$ *para cada* $i = 1,2,...,m$

$k_{2,i} = hf\left(t_j + \frac{h}{2}, w_{1,j} + \frac{1}{2}k_{1,1}, w_{2,j} + \frac{1}{2}k_{1,2}, ..., w_{m,j} + \frac{1}{2}k_{1,m}\right)$ *para* $i = 1,2,...,m$ cada $i = 1,2,...,m$

$k_{2,i} = hf\left(t_j + \frac{h}{2}, w_{1,j} + \frac{1}{2}k_{2,1}, w_{2,j} + \frac{1}{2}k_{2,2}, ..., w_{m,j} + \frac{1}{2}k_{2,m}\right)$ *para* $i = 1,2,...,m$ para cada $i = 1,2,...,m$

$k_{3,i} = hf\left(t_j + \frac{h}{2}, w_{1,j} + \frac{1}{2}k_{3,1}, w_{2,j} + \frac{1}{2}k_{3,2}, ..., w_{m,j} + \frac{1}{2}k_{3,m}\right)$ *para* $i = 1,2,...,m$ para cada $i = 1,2,...,m$; y entonces:

$w_{i,j+1} = w_{i,j} + \frac{1}{6}\left[k_{1,i} + 2k_{2,i} + 2k_{3,i} + k_{4,i}\right]$ para cada $i = 1,2,...,m$.

Así para este método en particular debemos calcular el conjunto de $k_{1,1}, k_{1,2}, ..., k_{1,m}$ deben ser calculados antes de que $k_{2,1}$sea determinado. En general, cada $k_{l,1}, k_{l,2}, ..., k_{l,m}$, debe ser calculado antes de que cualquiera de las expresiones $k_{j+1,i}$.

Luego para aproximar a la solución del sistema de orden m de problemas de valor inicial de primer orden se tiene como entrada del algoritmo los puntos extremos a,b; el número de las ecuaciones m; el entero N; y las condiciones iniciales $\alpha_1, \alpha_2, ..., \alpha_m$, obteniendo como salida las aproximaciones w_j a $u_j(t)$ en los $(N+1)$ valores de t.

A continuación el algoritmo de Runge Kutta Orden 4 para sistemas de ecuaciones diferenciales de primer orden[14]:

Paso 1: Tomar $h = (b-a)/N$;
$t = a$

Paso 2: Para $j = 1,2,....,m$ tomar $w_j = \alpha_j$

Paso 3: SALIDA $(t, w_1, w_2, ..., w_m)$.

Paso 4: Para $i = 1,2,....,N$ seguir los pasos 5-11

Paso 5: Para $j = 1,2,....,m$ tomar
$k_{1,j} = hf_j\left(t_j, w_1, w_2, ..., w_m\right)$

Paso 6: Para $j = 1,2,....,m$ tomar
$k_{2,i} = hf_j\left(t + \frac{h}{2}, w_{1,j} + \frac{1}{2}k_{1,1}, w_2 + \frac{1}{2}k_{1,2}, ..., w_m + \frac{1}{2}k_{1,m}\right)$

Paso 7: Para $j = 1,2,....,m$ tomar
$k_{3,j} = hf_j\left(t + \frac{h}{2}, w_1 + \frac{1}{2}k_{2,1}, w_2 + \frac{1}{2}k_{2,2}, ..., w_m + \frac{1}{2}k_{2,m}\right)$

Paso 8: Para $j = 1,2,....,m$ tomar
$k_{4,i} = hf_j\left(t + h, w_1 + k_{3,1}, w_2 + k_{3,2}, ..., w_m + k_{3,m}\right)$

Paso 9: Para $j = 1, 2, ..., m$ tomar

$$w_{,j} = w_j + [k_{1,j} + 2k_{2,j} + 2k_{3,j} + k_{4,j}]/6$$

Paso 10: Tomar $t = a + ih$.

Paso 11: SALIDA $(t, w_1, w_2, ..., w_m)$

Paso 12: PARAR

Este algoritmo es propuesto para solucionar el sistema de ecuaciones que se dedujo en el capítulo anterior acerca del planteamiento del problema.

3.5 SOLUCION MEDIANTE JAVA3D

3.5.1 Programación de los Métodos Numéricos

Para la instrumentación numérica se hizo uso de clases que se programaron en Java (Java3D realmente es un conjunto de bibliotecas de Java, por tanto el método se programó en Java y es 100% transparente para ser usado por Java3D), programandolas como sigue: (además cada programa en Java y/o Java3D constituye una clase[ ]:

Clase RungeKutta

Clase DerivnFunction

Clase Deriv

3.5.2 Instrumentación del Modelo en Java3D

Programación de Runge Kutta

Primeramente se programa la clase Runge Kutta y se adiciona el método fourthOrder, el cual es el método principal de la clase y se describe a continuación:

public static double[] fourthOrder(DerivnFunction g, double x0, double[] y0, double xn, double h)

Se trata de un método público y estático que regresará un vector del tipo double, sin importar el tamaño.

Obtiene los parámetros necesarios a través de los siguientes objetos:

g para el conjunto de ecuaciones, x0 como punto de inicio en x, y0 como un arreglo sin tamaño como un conjunto de valores iniciales, xn un punto final en el intervalo, y h el tamaño de paso

```
double ns = (xn - x0)/h;
ns = Math.rint(ns);
int nsteps = (int) ns;
h = (xn - x0)/ns;
```

Calcula el número de pasos dentro del intervalo y el tamaño de paso de primera intención

```
for(int i=0; i<nequ; i++)y[i] = y0[i];
```

Inicializa los valores iniciales de la aproximación que en este caso quedan en y (arreglo de tipo double), además el conjunto de datos esta determinado por nequ (número de ecuaciones).

for(int j=0; j<nsteps; j++){
x = x0 + j*h;
dydx = g.derivn(x, y);
for(int i=0; i<nequ; i++)k1[i] = h*dydx[i];
for(int i=0; i<nequ; i++)yd[i] = y[i] + k1[i]/2;
dydx = g.derivn(x + h/2, yd);
for(int i=0; i<nequ; i++)k2[i] = h*dydx[i];
for(int i=0; i<nequ; i++)yd[i] = y[i] + k2[i]/2;
dydx = g.derivn(x + h/2, yd);
for(int i=0; i<nequ; i++)k3[i] = h*dydx[i];
for(int i=0; i<nequ; i++)yd[i] = y[i] + k3[i];
dydx = g.derivn(x + h, yd);
for(int i=0; i<nequ; i++)k4[i] = h*dydx[i];
for(int i=0; i<nequ; i++)
{
y[i] += k1[i]/6 + k2[i]/3 + k3[i]/3 + k4[i]/6;

Realiza las iteraciones teniendo como límite el número de pasos que se ha dado, en cada ciclo intermedio nuevamente el número de ecuaciones es nequ.

return y;

Regresa el arreglo completo y.

La clase DerivnFunction sirve como una interfaz, utilizando la clase derivn1 para "traducir" de alguna forma las ecuaciones expresadas en la clase RungeKutta.

Programación de Elementos Gráficos en Java3D

Por separado se crea una clase ó programa en Java3D que unicamente constituye la animación sin tomar en cuenta la solución numérica que se realizó en el epígrafe anterior.

Primeramente mediante la clausula *extends JFrame* se permite que el programa corra como aplicación y como parte de un código en HTML además de permitir la inclusión de elementos gráficos especializados como son los controles clásicos en la programación web..

En seguida se construyen los elementos para la Escena dentro del Universo Virtual:

setLayout(new BorderLayout());

Se configura un administrador de Diseño en este caso BorderLayout[12][8]

GraphicsConfiguration config =

SimpleUniverse.getPreferredConfiguration();

Se crea un universo virtual que se añade a la configuración de las gráficas en Java3D.

Canvas3D c = new Canvas3D(config);

add("Center", c);

El objeto Canvas3D es instanciado como lienzo donde se insertan los objetos gráficos y d

Panel p =new Panel();

p.add(go);
add("North",p);
go.addActionListener(this);

El panel es creado y se añaden el boton "Go" que da pie al inicio de la animación, esta animación se invoca a través de un escuchador de eventos, en este caso el evento que se debe escuchar es el click sobre el boton "Go", y este dispara otras acciones (this hace una referencia hacia el mismo objeto)[12].

BranchGroup scene = createSceneGraph();
SimpleUniverse u = new SimpleUniverse(c);
u.addBranchGraph(scene);

Crea una escena, mediante el método createSceneGraph, después se coloca el objeto Canvas (en este caso c) dentro del universo virtual, y posteriormente se añade la escena tambien (pero como "hijo" al universo **u).**[19][12]

Así con elementos básicos y formas elementales conformamos una simulación de la válvula gobernadora de Watt como se ve en la Fig. 1

Figure 3.4: Frame de Inicio de la Animación

Se aprecia un cilindro, como eje principal de la válvula, esta estático, además se logró la instrumentación de los brazos mediante la construcción de dos líneas para simularlos, conectando luego dos esferas que hacen las veces de las masas esfericas en el dispositivo real..

Adición de la Animación

Para lograr la animación se utiliza un método particular RotationInterpolator que viene incluido como parte de las transformaciones elementales de Java3D[19] y que permite la inclusión de un comporamiento como se desee a través de ciertos parámetros, como se describe a continuación:

RotationInterpolator rotator2 = new RotationInterpolator(rotationAlpha, obj5Trans, yAxis,0.0f,vel);

RotationInterpolator rotator = new RotationInterpolator(rotationAlpha, obj4Trans, yAxis, 0.0f, vel);

Ambos métodos generan un objeto que es rotator y rotator2 que serán después adicionados a la estructura arbórea[19] de Java3D. Los paramétros necesarios son en este caso un ciclo infinito determinado por rotationAlpha, obj5Trans y obj4Trans son los objetos sobre los que se va a actuar (en este caso las masas esféricas y los brazos), yAxis el eje sobre el cual va rotar, punto de inicio (0.0f), y la velocidad que es uno de los paramétros fundamentales que se unen con la parte númerica (más adelante se explica dicha conexión).

rotator2.setSchedulingBounds(bounds);

rotator.setSchedulingBounds(bounds);

Se define el espacio de actuación de la rotación, donde se desea que ocurra, en este caso en el Universo Virtual.

objRoot.addChild(rotator2);

objRoot.addChild(rotator);

Se añaden a la estructura arbórea, como una hoja al final de la misma.[8][12]

La conexión con la parte númerica se lleva a cabo mediante la llamada al método de Runge Kutta. En este caso los métodos han sido programados en el enfoque de programación orientada a objetos, esto permite la facilidad de la declaración e interfaz entre los métodos de las clases que en este momento se invocan entre sí.

Como ya habíamos descrito los parámetros que utiliza el método principal RungeKutta.fourthOrder se dan a través de los objetos ya descritos. El método al ser invocado utiliza los valores de la interfaz que se describe en la parte última de este capítulo.

La llamada a este método dentro de la clase RungeKutta es la siguiente:

yn = RungeKutta.fourthOrder(dn, x0, y0, xn, h);

Los valores que regresa en sí la llamada de este método en este caso son 3, y corresponden a la aceleración rotacional de la válvula y[0], el ángulo de separación de los brazos y[1], y por último la velocidad de rotación de la válvula y[2].

Construcción de la Interfaz de Usuario

La construcción de la interfaz del usuario se lleva a cabo mediante el método principal que se inicializa cuando el programa comienza su ejecución(para comprender mejor la ejecución y puesta a punto del programa veáse Apendice B).

La siguiente figura ilustra la introducción de los parámetros con los que el usuario puede experimentar y que corresponden con la entrada de datos necesaria para el funcionamiento, además vale la pena mencionar que todos estos elementos fueron programados más no diseñados por ningun conjunto de herramientas visuales[22][19].

Después de ingresar los parámetros de Masa, Carga y Angulo, existe un método asociado al Click del boton OK. Se describe el método como sigue:

```
public void actionPerformed(ActionEvent e)
{
```

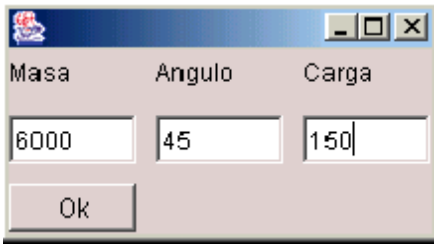

Figure 3.5: Interfaz del Usuario

```
if(e.getSource()==go)
....
else
{
  if(e.getSource()==ok)
{
  nmf.setVisible(false);
  valor= Integer.parseInt(angulo.getText());
  valor=(float)Math.sin((float)valor/57.3f);
 valor2= Integer.parseInt(masa.getText());
 valor2=(float)valor2;
 valor3= Integer.parseInt(carga.getText());
valor3=(float)valor3;
mf = new botar3(valor,valor2,valor3);
......
}
```

Cuando el usuario ha presionado click sobre ok el simulador comienza a funcionar, de tal forma que se asocie el frame correcto a esta parte de la interfaz.

Las variables valorl,valor2 y valor3 captan los parámetros antes mencinados convirtiendolos mediante cast a un tipo float deseado como se observa, posteriormente asocian el Frame deseado pasando como parametros a un método sobrecargado llamado botar3.

Una vez que estos valores son tomados dentro del Frame adecuado podemos llamar con dichos parámetros al método de Runge Kutta estableciendo las variables que se deseen y que en realidad son las condiciones iniciales para el sistema de ecuaciones diferenciales.

3.5.3 SIMULACION Y VISUALIZACIÓN

La categorización de los resultados aplicados mediante la solución usando Java3D se realiza a través de los colores y su significado:

Amarillo: Región Estable

Azul: Por debajo de la regíon estable

Rojo: Arriba de la región estable

Así para determinados parámetros se obtienen distintos colores y velocidades en la rotación, además de una inclinación de los brazos del regulador. El usuario además puede introducir otros valores si lo desea y presionar click en ok dando pie a un nuevo Frame el cual invocará el proceso otra vez y eso facilita la comparación entre varios resultados.La figura 3 ilustra este hecho, auxiliandose de los valores introducidos en la figura de la interfaz.

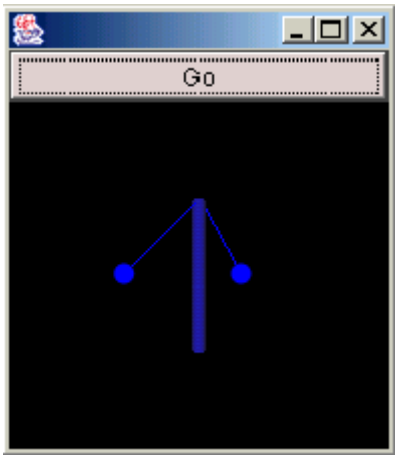

Figure 3.6: Visualización en Colores del Simulador

Se han presentado en este capítulo los aspectos más importantes a considerar concernientes a la solución del problema planteado, en un sentido la solución del problema que se resuelve de forma numérica gracias a la deducción e investigaciones hechas por Vichnegradski[], y en otro al utilizar Java3D para la solución computacional y la visualización. Por último al tomar ambos resultados se unen para formar la simulación completa con una interfaz del usuario que concluye la solución(veáse).

El método de Runge Kutta Orden Cuatro resulta entonces un método efectivo y adecuado para la realización de la solución del sistema mencionado. También se realizó una descripción breve del método.

En cuanto al software de elección, sus características y elementos fundamentales han cumplido con las expectativas deseadas. La explotación de las características de Java y Java3D es también un beneficio que hemos obtenido de la realización de este trabajo, a pesar de los problemas de bibliografía e información disponible acerca del software.

En el siguiente capítulo se habla acerca de los resultados obtenidos de forma número ica con el fin de demostrar la operación real del simulador y probarlo y en un apendíce posterior hablamos del manejo del simulador de la válvula.

Capítulo 4
Resultados y Pruebas

En este capítulo se muestran los resultados y pruebas que se obtienen acerca del simulador en Java3D. Como sabemos el simulador, como todos los dispositivos debe empezar con ciertos valores iniciales que no sabemos que efecto tendrán en el comportamiento del mismo. Una vez que sabemos cuáles son los resultados de aplicar estos valores, podemos variarlos y encontraremos forzozamente una serie de valores inciales que hacen que éste se estabilize y contninue ahí "indefinidamente". Así un problema interesante sería encontrar si existen además de los valores antes mencionados, otros con las mismas cualidades y saber cual es la tolerancia a la variación de los mismos[21].

Aún cunado se ha terminado la presentación de la instrumentación, es decir, del cómo se realizó la solución del problema, es conveniente ahora presentar los resultados que se obtuvieron y las pruebas que se realizaron en el momento de manejar la instrumentación anteriomente descrita.

Se presenta primero la variación de los paramétros iniciales que son: Carga, Masa, y Angulo y su papel dentro de la simulación y para el cálculo de los valores finales. En seguida se presentan las principales diferencias entre Matlab y el simulador en Java3D. Por último se muestra cual es una propuesta para la región de stabilidad, que es la región en la cual es estable el funcionamiento de la vávula

4.6 RESULTADOS

4.6.1 VARIACIÓN DE PARÁMETROS DE ENTRADA

La variación de paramétros dentro de la simulación permite determinar que paramétros iniciales harán que el regulador se comporte de forma estable se llegó entonces a los siguientes valores para lograr la estabilidad[1]:

[1]Este análisis fué hecho por el Dr. Gustavo Rodríguez Gómez en el INAOEP puede consultarlo en http://www.inaoep.mx/~grodrig

$\Psi = 0.0$; Derivada de psi
φ= 0.191; angulo de los brazos en radianes
$\varphi^* = 0.8726$; Valor "medio" de φpara mantener la magnitud de la regulación
$\Omega = 30$; velocidad angular de eje
F1= 300; fuerza maquina de vapor
P = 700; Fuerza del peso del dispositivo sobre el volante
g = 9.81; Gravedad
J = 200; Fuerza del Vapor
m=1; Masa de las Bolas
k = 6000; Constante de Proporcionalidad
b = 4; Coeficiente de Rozamiento
nc = 0.01; Constante de proporcionalidad para la relación entre el manguito y el mástil principal

Así tenemos un conjunto de valores que nos aseguran que el funcionamiento despues de cierto tiempo (aún cuando no se hizo el trabajo para tiempo real) se estabilizará. Entonces los valores que se requiere dar a la entrada para el código amarillo son los siguientes:

Carga: 200 Unidades de Fuerza.
Angulo: 0.191 En Radianes.
Masa: 1 Unidades de Masa.
Como se ve en la siguiente figura:

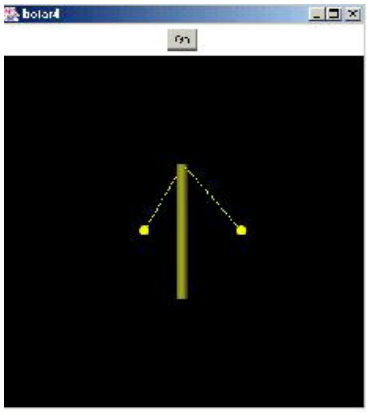

Figure 4.7: Estado Estable del Regulador (Color Amarillo)

Por otra parte la afectacion de los factores de la masa inicial para la simulación (la cual se simula a través de los códigos de colores) al proponer valores como una masa de 6000, que parece un valor exagerado y una carga de 500, suponiendo que con esto se obtiene una velocidad de giro estable, se ve disminuida ampliamente obteniendo una velocidad de giro muy pequeña, que se representa como sigue:

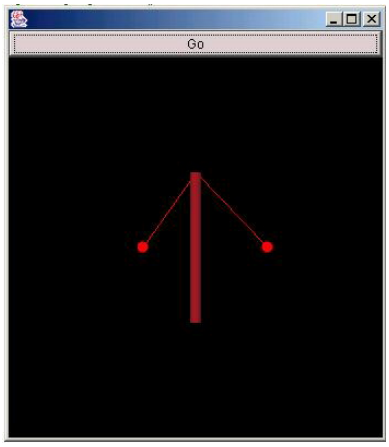

Figure 4.8: Región Inestable, por "arriba" de los valores iniciales estables

Para finalizar, si tenemos una disminución importante en el tamaño de la carga y además un aumento "proporcional" de la masa, de tal forma que se quiera estabilizar el regulador, obtenemos valores que corresponden a una velocidad de giro y de su representación por "debajo" de la región de los valores estables, como se comprueba en el siguiente epigrafe, y la comparación hecha mediante Matlab. Así obtenemos el código azul como se ve a continuación:

4.6.2 Comparación entre Matlab y Simulador en Java3D

Para los valores establecidos con condiciones iniciales obtenidas para la región de estabilidad se tienen los siguientes resultados en ambos, tomando en consideración 3 cifras decimales significativas y sin redondeo[2]

	Matlab	Simulador	Diferencia Absoluta	Diferencia Relativa
Aceleración	0.0	0.0	0.0	0.0
Velocidad	37.190	37.184	0.006	$1.613e-4$
Ángulo	0.711	0.720	0.009	0.0115

Los resultados de la tabla anterior muestran una diferencia poco significativa entre los resultados obtenidos con matlab y Java3D, lo cual hace que los resultados obtenidos sean confiables y que el método de Runge - Kutta muestra ser eficiente para este caso en particular.

REGION ESTABLE

[2]Este mismo trabajo en Matlab se realizó para comprobar que los resultados se aproximan de manera conveniente; hacerlo en Matlab resulta trivial

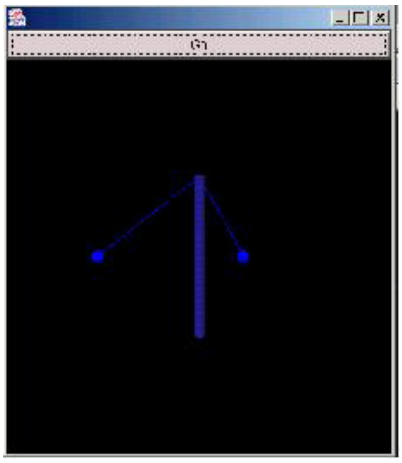

Figure 4.9: Regulador Inestable por "debajo" del nivel estable de funcionamiento

Dados los experimentos de variación de parámetros y comparación entre los resultados obtenidos observamos que se ha hecho la simulación para 60 iteraciones, que representa 60 unidades de tiempo[13][7].

En el punto t=45 el regulador se comporta de manera estable y no hace variación alguna de los parámetros, es decir, su funcionamiento cumple con la regulación.

Este último resultado es el que se simula aquí, o sea, se toma una serie de valores que se han introducido y se muestra al usuario el efecto que tiene, si se cambian los valores (veáse Apéndice 2), el usuario ve otro efecto, hasta que al final encuentre o de los valores que le concedan estabilidad.

4.7 PRUEBAS

4.7.1 Pruebas del método numérico

Con el fin de validar el método numérico que se ha programado en el presente trabajo, se realizaron dos pruebas[14][6]:

- Con una Ecuación Diferencial
- Con un Sistema de Ecuaciones Diferenciales Ordinarias

Obteniendo error del orden de 10E -08 para los experimentos con una sola ecuación diferencial y de hsta 10E-06 en sistema de ecuaciones diferenciales.

4.7.2 PRUEBAS DEL SOFTWARE

Al restringir a una interfaz del usuario como se habló en los capítulos anteriores, se restringen los errores en el manejo del mismo. Los valores que se han restringido al código de colores para la región de estabilidad mencionada en el capítulo anterior. Por último la viabilidad de programar un software en sus detalles más triviales nos da un mejor panorama de la programación orientada a objetos.

Hemos presentado los principales resultados que se han obtenido al aplicar las técnicas tanto numéricas como de software. Además de esto hemos presentado una forma de validar los resultados de que se habla.

El método de Runge Kutta Orden cuatro resulta eficiente para la realización del software descrito con anterioridad.

Podemos emitir ahora las conclusiones que se desprenden de utilizar el modelado basado en física con Java3D, después conclusiones acerca del sistema.

Capítulo 5
Conclusiones

A continuación se presentan, en base a los resultados obtenidos en los capítulos precedentes, conclusiones que nos parece interesante revisar con el fin de completar el trabajo, dejando un precedente para otros trabajos que se pueden derivar de éste y que ahondarían más en estos conceptos, que aunque técnicos ocupan a otras ramas del saber humano y que incluso pueden ayudarle a un mejor entendimiento de otras disciplinas.

Se comienza con las conclusiones que arrojan el hecho de utilizar la técnica del modelado basado en física para continuar con la parte matemática propiamente dicha y las consecuencias que de ello se obtuvieron, posteriormente se revisan los puntos acerca de las disciplinas participantes que conllevan a un trabajo interdisciplinario, para concluir con una revisión de los trabajos futuros que se pueden realizar con las mismas tecnicas utilizadas aquí.

5.8 Java3D y Modelado Basado en Física

Como ya se ha estudiado en el presente trabajo e incluso en anteriores(vease [19]) vemos que el modelado basado en física es una técnica eficiente para modelar los cuerpos y situaciones dentro de la mecánica clásica y la teoría de control, como en este caso. A nivel computacional existen herramientas que pueden de alguna forma resolver el problema a nivel visual, tales como OpenGL, y otros lenguajes como C/C++, etc. En nuestro caso particular al elegir Java3D hemos obtenido resultados eficientes con relativamente poca complejidad utilizando el paradigma orientado a objetos, esta herramienta ha resultado adecuada para la instrumentación de las técnicas del modelado basado en física y se ha argumentado el uso de Java3D y el porqué de su elección(vease Apendice 1).

Java3D, como ya vimos, contiene una serie de bibliotecas de clases, métodos y objetos que simplifican la abstracción además de contar con diversas herramientas que ambientan de alguna forma el trabajo y que dan un realce y un realismo bastante aceptable en términos de los objetos que observamos en la realidad.

Lo cual aunado al rápido desarrollo y puesta a punto de los programas, con la independencia de la plataforma, simplicidad de escritura, correr como aplicación y como applet, y demás ventajas ya expuestas, es una de las principales ventajas del uso de dicho lenguaje.

Ahora bien cuando las herramientas que proporciona Java3D son adecuadas para el manejo del modelado basado en física, la dificultad se encuentra dado que no existen manuales y documentación suficiente para la elaboración de los programas.

Otra dificultad importante es el hecho de la "incompatibilidad" entre los objetos del tipo Shape y los objetos Geometry, provocando que las operaciones entre ellos y su acoplamiento resulte en términos de visualización bastante complejo. Esto significa que se debe utilizar un conjunto de herramientas ya sea de un tipo ó clase, logrando con esto que se pueda operar de forma consistente, pero con pérdida de realismo. Esto se soluciona mediante el cambio de atributos y adecuación de materiales que incluye Java3D[12][16].

El comportamiento de varios objetos interactuando entre sí, la aplicación matemática y física al programa son una aportación importante.

5.9 Método de Runge-Kutta Orden Cuatro

El método de Runge-Kutta Orden Cuatro como ya se estudió es un método para la solución numérica de ecuaciones diferenciales ordinarias. En este caso se programó un método en el paradigma orientado a objetos. Aunque es cierto que el number crushing de Java es bastante bajo comparativamente con otros software, resulta ser un buen ejemplo de la aplicación de la tecnología OO y de un paradigma distinto al tradicional, que para el caso particular y con la tolerancia que se específica funcionó y tuvo resultados aceptables para un primer prototipo.

En el capítulo correspondiente no solamente se describió el método de Runge Kutta Orden Cuatro sino que además se comentó cómo se construye y cómo es posible generalizarlo a un sistema de ecuaciones diferenciales ordinarias, aún cuando no es esto el sentido principal de la tesis.

5.10 Trabajo Interdisciplinario

El presente trabajo es un trabajo interdisciplinario que combina las siguientes áreas del conocimiento:

- Mecánica Clásica
- Ecuaciones Diferenciales
- Solución Numérica de Ecuaciones Diferenciales
- Teoría de Control

- Java, Java3D y Simulación

La dificultad de los tópicos y del trabajo en sí radica en la necesidad de conocer e incluso dominar ciertas ramas de los tópicos anteriores.

En cuanto a la mecánica clásica es necesario de dominar y conocer la parte de la leyes de Newton y aplicarlas de manera conveniente al estudio del problema y por otra parte en las matemáticas puras deben conocerse las ecuaciones diferenciales y su objeto de estudio así como las categorías de las soluciones y sus variantes.

Las ecuaciones diferenciales se analizan y se utilizan de manera conveniente para realizar una descripción del comportamiento de la válvula. A su vez este sistema resuelto numéricamente mediante los métodos que se resultaron eficientes como es el caso de Runge-Kutta Orden 4.

La teoría de control de forma elemental es abordada para situar su contexto histórico y su relevancia así como para comprender los mecanismos y formas del comportamiento de la válvula que se modela. El modelo matemático nos lleva a un conujunto de ecuaciones diferenciales.

Es necesario dominar de manera apropiada el lenguaje de programación java, además de conocer de manera importante los métodos y objetos del Java3D. Por otra parte la simulación es parte de una conjunción de algoritmos y requisitos que son necesarios para el sistema en particular que se desee modelar, y además el grado ó nivel de simulación y los ambientes necesarios para la conservación de un nivel optimo de resultados redunda en conocimientos del modelado y de la simuación en sí.

Este trabajo como ya se ha mencionado integra diferentes temas para su realización como son: Física, Programación Orientada a Objetos, Teoría de Control, etc.

5.11 Trabajos Futuros

El sistema actual podría mejorarse simulando el cambio de masa de las bolas pegadas a los brazos, además de la irregularidad de la marcha. Sin embargo un aporte considerable sería simular el comportamiento en tiempo real y poder parar el simulador, cambiar los paramétros, y ver el nuevo comportamiento de la válvula.

Se pueden realizar otras simulaciones derivadas del presente trabajo, en particular pueden ser útiles estas ideas para otros problemas de control, simulación de fluídos, etc.

Dado que hemos encontrado que Java3D es adecuado para la simulación podemos ocupar el mismo lenguaje y las mismas propiedades de los métodos e inclusive añadir métodos de ecuaciones diferenciales, para su sustitución dentro de otros problemas, aprovechando que los métodos expuestos han sido programados usando el paradigma orientado a objetos.

Otros sistemas dinámicos hechos mediante otras herramientas pueden ser probados e incluso mejoradas mediante el uso del modelado basado en física y el lenguaje Java junto con la extensión de Java3D. En resumen la aplicación que aquí se refiere hace alusión a una serie de conocimientos técnicos de distintas disciplinas para su realización, a distintos niveles, pero sin olvidarnos de la necesidad de conocer y en ocasiones dominar las partes del conocimiento requerido para el modelado del sistema que se planteó originalmente y dar como resultado su simulación.

Por otra parte para realizarlo ha sido sufciente la implementación del método de Runge Kutta Orden 4 para la solución de las ecuaciones diferenciales que se derivan del modelo, y hemos encontrado una herramienta adecuada y eficiente para la realización de simulaciones que parten del modelado basado en física y estos resultados son posibles de utilizar para modelar sistemas de máquinas complejas ó simples, sistemas de resortes, y otros sistemas.

Apéndice 1: Software de Elección

El propósito del presente apendíce es la elección de un software para el desarrollo de la aplicación que ocupa a este trabajo, ya que es de capital importancia el conocer los posibles obstáculos con los que se enfrentamos si elegimos tal ó cual software.

Aquí se presenta un análisis comparativo entre dos herramientas a considerar: OpenGL y Java3D y se emite, en base a este análisis una elección por Java3D.

Primeramente se analiza a OpenGL, se dan los antecedentes de este software, para continuar con la presentación de sus características generales, terminando con una lista de ventajas y desventajas.

Seguidamente a esto se hace lo mismo para Java3D, enumerando de la misma forma al final de la sección sus ventajas y desventajas.

Por último se hace un análisis comparativo donde se exponen resumidamente las ventajas y desventajas de los dos software, para terminar con una gráfica que justifica la elección del software.

1.12 Open GL

1.12.1 Panorama General

OpenGL (en inglés *Open Graphics Library*) es una interface de software para hardware gráfico. Consiste de cerca de 120 comandos distintos que permiten especificar los objetos y las operaciones involucradas en producir imágenes gráficas de alta calidad, aplicaciones en 3D interactivas e imágenes a color de estos objetos[15].

La terminología de OpenGL no es solamente válida dentro de su ambiente de forma particular, también utilizada en otros lenguajes y ambientes gráficos que se basan en OpenGL.

La *renderización* es el proceso por medio del cual la computadora crea imágenes provenientes de los modelos. Estos modelos en general están constituidos de primitivas geométricas que son especificadas por sus vértices.

Toda imagen como se sabe está constituida por pixeles en la computadora. La información de esos pixeles esta organizada en bitplanos los cuales son áreas de la memoria que mantienen la información para cada pixel sobre la pantalla, así estos están organizados sobre sí mismos en framebuffer que son los que mantienen la información acerca de los desplegados de las gráficas necesarias para controlar la intensidad de todos los pixeles sobre la pantalla.

Puesto que OpenGL no es en primera instancia como una API en si misma y no existen enlaces directos para periférico asociados se incluyen algunas librerías que junto con la instalación del paquete ayudan a establecer estos enlaces.

A continuación se describen las librerías más importantes de OpenGL con funciones específicas que complementan sus comandos y su uso.

·GLU (OpenGL Utility Library) Contiene varias rutinas en el más bajo nivel que se usan para ejecutar las tareas tales como instalar matrices para orientaciones y proyecciones de vistas específicas, polígonos teselares, y renderización de superficies. Esta librería es provista como parte de la implementación de OpenGL. Las rutinas de esta usan el prefijo glu.

·GLX Extensión de OpenGL para el sistema de ventanas X. Esta librería da formas de crear un contexto de OpenGL y está asociada con una ventana dibujada sobre una máquina que habla X. Está dada como una librería adjunta para OpenGL y las rutinas de ésta usan el prefijo glX[18].

1.12.2 Características Generales

Típicamente las funciones que se pueden realizar en OpenGL son las siguientes

·Diseños derivados que muestran los objetos hechos por líneas curvas y rectas que asemejan alambre. Existe además una correspondencia entre cada línea de alambre hacia una arista de una primitiva (un polígono generalmente).

·Versiones de *depth-cued* para los frames hechos de alambre mediante el uso de perspectiva para dar la sensación de profundidad, este se basa en el efecto de la vida real cuando un conjunto de líneas lejanas al observador son difusas.

·El antiescalamiento: El renderizado de los contornos presenta imperfecciones conocidas como efecto escalera. El antiescalamiento trata de reducir este efecto al mínimo.

·Versiones de sombras planas de las escenas: esto hace que los objetos parezcan planos puesto que no aparentan responder a las condiciones de iluminación no aparecen acabados de manera lisa, sino mas bien borrosos en cuanto a las superficies.

·Versiones sombras suavizadas: cuando los objetos en 3D se les quiere dar realismo es conveniente añadir sombreado con el cual se responde a los recursos de iluminación en el cuarto donde se colocan los objetos, así las superficies de los objetos aparecen en acabados planos.

·Versiones con texturas y sombras: Las sombras no son una característica explícita en OpenGL (es decir no hay un comando "sombra") pero se puede crear mediante técnicas que involucran los cálculos necesarios de iluminación. El mapeo de textura permite aplicarse a 2D y 3D, de tal forma que se vean las diferencias entre objetos de distintos tipos.

·Versiones Empañadas: Los objetos son capaces de verse de otras formas y aparecen capturadas como un efecto de un movimiento de los mismos. Así podemos trazar incluso por medio de los efectos cómo se mueve en determinado momento tal o cual objeto y su camino mediante este efecto.

·Enfoque de las imágenes: Existe la posibilidad de cambiar el punto de vista del observador viendo la misma escena desde otros puntos de vista , dar alejamientos y acercamientos de una misma toma o de un mismo objeto.

Además existen otros efectos que se pueden lograr, y que son efectos visuales complejos:

·Apariencia de la atmósfera: cuando una escena se le pueden colocar niebla, humo u otros componentes que tengan inclusive partículas o sistemas de partículas en el aire.

·Profundidad de campo: Esto simula que se inhabilitan los lentes de una cámara sacando las escenas de foco y manteniéndolos en un lugar localizado de la escena estos estarán significativamente tan lejos o tan cerca como estén del lugar que esta empañado.

Muchos de los requerimientos de Open GL son que las gráficas hardware contengan un framebuffer, además de que algunos de las operaciones de OpenGL son específicamente concernientes hacia la manipulación del framebuffer[15].

Para los programadores OpenGL puede ser visto como un conjunto de comandos que permiten la especifiación de objetos geométricos en dos y tres dimensiones que controlan además como los objetos son renderizados dentro del framebuffer.

Desde el punto de vista del instrumentador es un conjunto de comandos que afectan las operaciones de los gráficos de hardware, entendiendo por hardware únicamente un conjunto de framebuffers direccionables. Lo cual hace que se pueda crear una instrumentación casi enteramente sobre el CPU anfitrión.

OpenGL visto como una máquina de estado que controla un conjunto de operaciones específicas de dibujado puede producir una especificación que satisfaga las necesidades de ambos: programadores e instrumentadores.

Fundamentalmente OpenGL concierne sólo con el desplegado dentro del framebuffer. No hay soporte para otros periféricos algunas veces asociados con gráficas hardware tales como ratones y teclados. Los programadores deben depender de otros mecanismos para obtener la entrada del usuario.

Las primitivas de dibujo en OpenGL tienen un número de modos elegibles. Cada primitiva es un punto, segmento de línea, polígono, o rectángulo de pixeles. Cada modo puede ser cambiado independientemente; el establecimiento de una primitiva no afecta el establecimiento de otras. Los modos son conjuntos, primitivas especificas y otra operaciones descritas mediante el envío de comandos en la forma de función o llamadas a procedimientos.

Las primitivas son definidas mediante un grupo de uno o más vértices. Un vértice define un punto, un punto final de una arista, o una esquina de un polígono. Los datos, consistentes de posiciones, coordenadas, color, normales, y coordenadas de textura, son asociados a un vértice. Cada vértice se procesa independientemente, en orden, y en la misma forma[15].

La única excepción a esta regla se da cuando el grupo de vértices se unen de forma que la primitiva indicada dentro de una región específica es modificada, y los vértices son creados nuevamente.

Los comandos son siempre procesados en el orden en el cual fueron recibidos a pesar de que haya un retardo intermedio antes de que los efectos de un comando sean realizados.

En OpenGL, la ligadura de los datos ocurre sobre la llamada. OpenGL proveé controles sobre las operaciones fundamentales sobre los gráficos de 3D y 2D. Esto incluye especificaciones de tales parámetros como matrices de transformación, coeficientes de ecuaciones de iluminación, métodos de antiescalamiento de gráficos y operadores de actualización de pixeles.

A pesar de esto OpenGL no provee en realidad primitivas de alto nivel para crear o describir modelos en 3D, estas operaciones o primitivas le permitirían crear formas complicadas tales como automóviles, partes del cuerpo, aeroplanos, o moléculas. Con OpenGL se podría construir estos modelos que se desean con un pequeño conjunto de primitivas geométricas, puntos, líneas y polígonos (una librería sofisticada que provee estas características ciertamente podría ser construida en la parte alta de OpenGL , de hecho esta librería existe y es el Open Inventor).

El modelo para la interpretación de comandos es cliente-servidor. Esto es un programa (el cliente) emite los comandos, y estos comandos son interpretados y procesados mediante el GL (el servidor). GL es entonces "transparente en la red". Un servidor puede mantener un número de *contextos*, cada uno de los cuales es un encapsulamiento de un estado actual de OpenGL.

Emitiendo los comandos de OpenGL cuando el programa no está conectado a un contexto resultará en un comportamiento indefinido. Los efectos de los comandos de OpenGL sobre el framebuffer son controlados de manera última mediante el sistema de ventanas que redirecciona los recursos del framebuffer.

No existen en OpenGL comandos para configurar el framebuffer o inicializar el OpenGL. Similarmente, desplegar el contenido del frambuffer sobre un monitor CRT (incluyendo las transformaciones individuales de los valores del framebuffer mediante las técnicas tales como corrección gama) no se direcciona mediante OpenGL, esto es, no es desplegado gracias a OpenGL sobre el dispositivo mencionado.

La configuración del Framebuffer ocurre fuera de OpenGL en conjunción con el sistema de ventanas. Esta diseñado para correr sobre un rango de plataformas gráficas con variación de capacidad y ejecución de gráficos. Para acomodar esta variedad especificaremos un comportamiento ideal en lugar del comportamiento actual para ciertas operaciones de GL.

Los estados de las variables están categorizadas un poco arbitrariamente por sus funciones. Se pueden describir las operaciones que OpenGL ejecuta sobre el framebuffer, aunque el framebuffer no es una parte de los estados de OpenGL.

Distinguimos dos tipos de estado: El llamado estado de servidor el cual está residente en el servidor de OpenGL que se encarga de responder a las solicitudes de un número de clientes en la red y que tiene las prmitivas de OpenGL.

El segundo estado llamado el estado de Cliente el cual está en otra máquina que hace las solicitudes al servidor.

La mayoría de las máquinas tienen los estados al mismo tiempo pues OpenGL está residente en una máquina unicamente sin necesidad de conectarse a la red.

Varios grupos de comandos ejecutan la misma operación pero difieren en la forma como sus argumentos les son dados. Los comandos están formados por un nombre seguido, dependiendo del comando particular,por 4 caracteres. El primer carácter indicando el numero de valores de el tipo indicado que debería ser presentado para los comandos. El segundo carácter o par de caracteres indicados es el tipo especificado de los argumentos. El caracter final, sí está presente, es v, e infidca que el comando toma un puntero para un arreglo de valores en lugar de una de argumentos individuales.

Las declaraciones del presente trabajo aplican al ANSI C, y son válidas también para lenguajes tales como C++ y Ada.

A continuación se da una explicación de las operaciones necesarias para el renderizado de imágenes en la pantalla:

·Construcción de formas desde primitivas geométricas,creando así descripciones matemáticas de los objetos. (OpenGL considera los puntos, líneas, y mapas de bits como primitivas).

·Ordenar de manera conveniente los objetos en el espacio tridimensional y seleccionar el punto de ventaja para la visualización de la escena compuesta.

·Calcular el color de todos los objetos. Estos deberían ser explícitamente asignados por la aplicación, y determinados por las condiciones específicamente de iluminación,u obtenidos mediante el pegado de textura sobre del objeto.

·*Rasterización* Es el proceso de convertir la descripción matemática de los objetos y la información del color asociado sobre los pixeles de la pantalla[14].

Las operaciones fundamentales de OpenGL se muestran en el siguiente diagrama:

DIAGRAMA DE LA FIGURA EN OPENGL

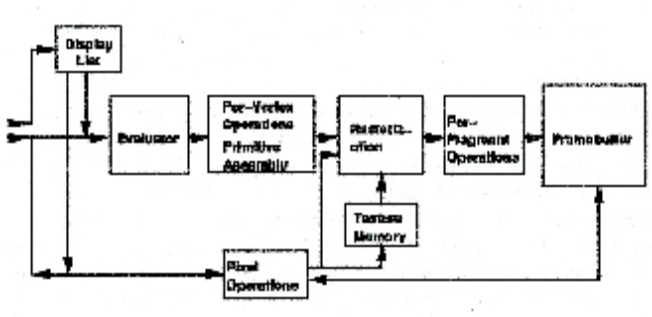

Figure 1.10: Diagrama del funcionamiento de OpenGL

La primera etapa proveé un siginificado eficiente para aproximar curva y superficie geométricas por evaluación polinomial de funciones de los valores de entradas. En la siguiente etapa opera sobre primitivas geométricas descritas por vértices: puntos, segmentos de línea, y polígonos. En esta etapa los vértices son transformados e iluminados, y las primitivas son unidas al volumen de la vista en preparación para la siguiente etapa, la rasterización. El rasterizado produce una serie de direccionamiento de los framebuffers y valores usando una descripción bidimensional de un punto, segmento de línea o polígono. Cada fragmento así producido es alimentado hacia la siguiente etapa que ejecuta operaciones sobre fragmentos individuales antes de que ella finalmente modifique el framebuffer. Estas operaciones incluyen actualizaciones dentro del framebuffer basado sobre de la entrada y valores de profundidad previamente almacenados combinándose con la entrada de colores fragmentados o con colores almacenados, tan bien como enmascaramiento y otras operaciones lógicas sobre valores fragmentados.

Durante las etapas que se han mencionado podría ejecutarse otras operaciones, tales como eliminación de partes de objetos que están ocultas por otros objetos (partes que no son dibujadas, las cuales podrían incrementar el desempeño).

Hay una forma de desviar los vértices procesando porciones de tuberías para mandar un bloque de fragmentos directamente hacia las operaciones de fragmentos individuales, eventualmente causando un bloqueo de pixeles a ser escrito hacia el framebuffer; los valores pueden ser también leídos atrás del framebuffer o copiados de una porción el framebuffer a otras. Estas transferencias pueden incluir algunos tipo de decodificación y codificación.

En el OpenGL, muchos objetos geométricos son dibujados mediante el acotamiento de una serie de conjuntos de coordenadas que especifican vértices y opcionalmente normas, coordenadas texturas y colores entre pares Begin/End. Hay 10 objetos geométricos que son dibujados de esta forma: puntos, segmentos de líneas, ciclos de segmentos de línea, segmentos separados de líneas, polígonos, triángulos sucesivos , abanicos de triángulos, triángulos separados, cuadriláteros agrupados, cuadriláteros separados[14].

1.12.3 Ventajas y Desventajas

Ventajas

Existe una documentación Amplia y profusa, libre de costo.

Se ha adoptado como un estándar para muchas otras interfaces e incluso API profesionales.

Existe una similitud con el lenguaje de programación C en particular con el ANSI C.

La permanencia está garantizada en el mercado.

El tiempo de renderizado es relativamente corto ya que se accede a primitivas del bajo nivel.

Desventajas

No es totalmente portable, requiere de modificaciones propias en cada plataforma.

Se tiene una cierta complejidad al tratar con vértices en lugar de objetos ó con el mínimo nivel

Las primitivas y los gráficos están íntimamente ligados al hardware, lo cual complica el proceso de rasterizado

La flexibilidad de los programas puede volverse en contra del usuario.

No cuenta con una interface propia para interactuar con otros elementos como la web, los ratones, etc, es decir, requerimos de otras herramientas.

1.13 JAVA 3D

1.13.1 Antecedentes

Java3D API (*Aplication Program Interface*) es una interfaz para escribir programas, desplegar e interactuar con gráficas en 3D, es además una extensión de Java 2 JDK, y viene incluido en la mayoría de las versiones que se manejan tales como J2SEE, SDK, etc, y es posible bajarlo de la red gratuitamente[19].

El objetivo que se plantea para la visualización en Java3D es la construcción de métodos que cumplan con los siguiente:

·Observar el estado de los objetos: acceso

·Mutadores de estado de los objetos: manipulación

·Ligado de nodos a la estructura arbórea

Acceso

Para lograr observar el estado de un objeto es necesario saber si los valores de los atributos son los adecuados y si estos valores son posibles de acceder, es decir, si son posibles de "ser vistos" por los métodos que se necesiten para el funcionamienmto del renderizado.

Manipulación

Para que un objeto pueda presentarse de una forma particular y después cambiar hacia otra que se deseé, es necesario cambiar los atributos adecuados. Los cambios pueden ser temporales, como en el caso de una deformación temporal, ó bien definitivos.

Ligado a la estructura árborea

La estructura árborea se refiere a un grafo que es consecuencia directa de la aplicación de la receta para escribir programas en Java3D y que se explica a continuación.

Receta para escribir programas en Java3D

En Java3D existen reglas propias para la escritura de programas, es una "receta" que se describe a continuación:

1.-Crear un objeto Canvas3D

2.-Crear un objeto SimpleUniverse el dcual hace referencia a el objeto anteriormente creado Canvas3D

a) Personalizar el objeto SimpleUniverse

3.-Construir un contenedor branch

4.-Compilar el grafo del contenedor branch

5.-Insertar el grafo del contenedor branch en el Locale del SimpleUniverse.

La receta enunciada se esquematiza como un "Grafo de Escena" el cual es creado usando instancias de clases de Java3D, es decir, objetos que además definen atributos acerca de la apariencia general de la visualización y del audio[9].

El grafo necesita una nomenclatura especial dentro de la cual distinguimos dos partes: los nodos que en este caso representan instancias de las clases de Java3D y los arcos que representan dos clases de relación entre las instancias, como se ve en la siguiente figura:

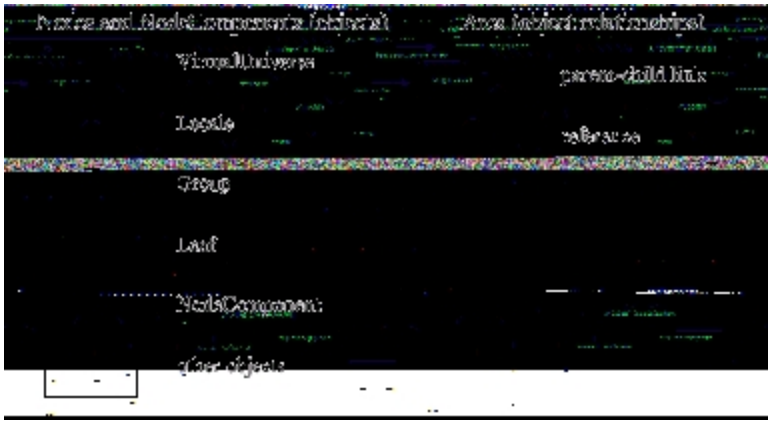

Figure 1.11: Símbolos del Grafo en Java3D

La relación que se muestra parent-child (padre-hijo) representa el hecho de que un grupo de nodos puede tener cualquier número de hijos pero un sólo padre. Las hojas tienen un sólo padre, pero no tienen hijos.

El otro tipo de relación es una referencia, la cual se asocia a objetos NodoComponentes con un nodo del grafo de escena, este objeto define los atributos de la geometría apariencia usados en el renderizado de los objetos visuales.

El resultado de aplicar la nomenclatura anterior a la receta que hemos enunciado se observa en la siguiente figura:

La línea punteada representan el "ahorro" del esfuerzo y de tiempo que se hace para progrmar el universo virtual que es la base donde se colocan realmente los objetos que se necesitan para la visualización de la escena.

Por regla general los objetos que son creados en Java3D deben ser añadidos a la estructura arbórea , esto se logra mediante métodos que se tienen exprofeso para tal fin.

Como sabemos Java como C no exige un orden riguroso y específico para la estructura de los programas que se escriben. Por lo anterior el hecho de que exista una receta para la escritura de programas no implica que debemos seguir un órden riguroso en cuanto a la forma de construir y ligar los objetos, pero sí debemos conservar cierta estructura general.

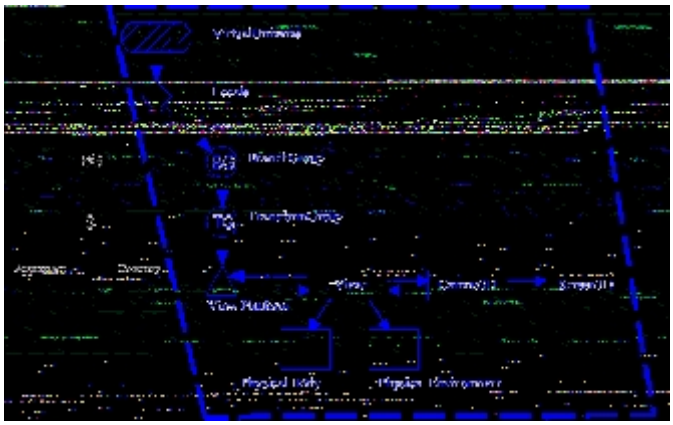

Figure 1.12: Estructura árborea para construir una aplicación en Java3D

1.13.2 Características Generales

Las características de las aplicaciones en esta API son las siguientes:

*Soporte de constructores de alto nivel para figuras geómetricas y estructuras para manejarlas y deplegarlas

*Pueden ralizarse aplicaciones que hagan un renderizado en paralelo, optimizando así el rendimiento.

*Pueden correr como una aplicación simple, ó también como un applet para cualquier navegador de red que soporte Java.

*Provee una gran cantidad de soporte para archivos de diversos formatos, y cargadores para los mismos

*Tiene gran cantidad de objetos y clases que se pueden conformar en código que facilitan al programador substituir cualquier método que no satisfaga sus expectativas, por otro propio.

*Se basa en dos estándares diferentes, una versión que se basa en OpenGL y otra versión que se basa en DirectX.

Através del uso de la receta que se ha definido y de la construcción de un Universo Virtual en 3D, podemos elegir el tamaño e inclusive la forma del mismo, los demás componentes se van añadiendo de forma gradual

A continuación explicamos las características escenciales de los componentes que se han mencionado en la receta para su adición en el grafo principal.

Cuerpos Geométricos y Figuras

Las figuras y cuerpos geométricos en general son generadas mediante el uso de instancias a las clases que se proveen, los atributos del objeto tienen un efecto directo en la forma como se visualiza el objeto en la pantalla.

Además los objetos pueden ser cambiados mediante el uso de transformaciones elementales de translación y rotación. Estos son métodos predefinidos

con paramétros espeíficos para la manipualción ó transformación que se deseé hacer.

En cuanto a las figuras también es posible incorporar elementos de tres dimensiones, que son tradicionales en el entorno 2D, por ejemplo líneas, puntos, polígonos, etc.

La diferencia entre un objeto geométrico y los demás radica en que son ligados directamente a un nodo especial en el grafo llamado Shape3D, así no forman parte de una instancia "común" como los demás objetos.

Transformaciones Elementales

Las transformaciones de rotación y translación que se ejecutan en las API tradicionales aquí se presentan como una colección de métodos que forman en sí un comportamiento.

Basta definir los parámetros de velocidad, posición, inclinación y ejes donde se requiere que se ejecute la transformación, ó bien la nueva posición para que estas transformaciones sean añadidas de nueva cuenta al grafo principal.

Todos los objetos deben permitir ó denegar la posibilidad de ser manipulados con transformaciones elementales, esto se logra a través de la claúsula ALLOW__ y el sufijo que se agrega posteriormente que puede ser WRITE, ó READ. Así cualquier objeto puede permitir ó habilitar y deshabilitar la modificación de sus atributos.

En sí un conjunto de transformaciones elementales pueden forman un comportamiento el cual simula de tal forma que se logra un efecto realmente vistoso y muy acercado a la realidad.

Apariencia

En cuanto a los efectos deseados sobre un material, Java3D ofrece un conjunto de clases que al ser instanciadas y establecidas como variables miembro definen juntas un estado de apraciencia que se asemeje lo más posible a la realidad.

Las clases enunciadas forman parte de la superclase Shape3D y que como mencionamos es una hoja. Por lo tanto las superficies de muchos materiales pueden ser "programadas" mediante el uso de las clases de apariencia.

El control de las variables de un objeto puede darse en los atributos particulares propios, por ejemplo Color, Textura, Materiales, etc, que son activados o desactivados ó asignados mediante bits.

Como todos los objetos se debe permitir que los objetos habiliten o deshabiliten las capacidades de tranformación.

Flexibilidad

Como hemos visto, las aplicaciones en Java3D siguen un proceso bien definido para la escritura de los programas mediante métodos que vienen definidos en bibliotecas de clase.

Sin embargo, Java3D como Java, permite la inclusión de métodos propios para funciones que no satisfagan las expectativas del programador. Las limita-

ciones sobre estas capacidades están relacionadas únicamente con el nombre de las variables y el nombre de los métodos que se van a construir.

Además la construcción de objetos propios del programador no está restringida solamente los provistos por Java3D; el programador también puede construir sus propios objetos y agruparlos en clases, pudiendo insertarlas como superclases o subclases de acuerdo a sus necesidades.

Luces

Al respecto de las luces podemos distinguir cuatro tipos de luces que determinan el efecto que tal ó cual iluminación dará a la visualización.

Las luces de ambiente proveen la misma intensisdad de luz en todos los sentidos y en todas las posiciones, esto sin contar desde luego la influencia de la luz en una posición determinada.

Existe también una regionalización de lo objetos que se pueden denominar de iluminación, esto dentro de las superficies.

La ilumniación se lleva a cabo mediante parámetros matemáticos, tales como vectores y direcciones de los mismos que realizan los efectos necesarios para los materiales y las texturas.

Swing y AWT

AWT Para que los programas hechos en Java3D se puedan relacionar con un browser ó con interfaces interactivas es necesario contar con una herramienta que haga esa conexión.

En Java esta herramienta es la biblioteca AWT (Abstract Windowing Toolkit), el kit de herramientas para el manejo abstracto de las ventanas.

Existen tres elementos para la construcción mediante esta biblioteca de clases de las interfaces, y son los siguientes:

Componentes: Son todos los objetos que van a estar incrustados en el applet ó interfaz que se ha creado, estos pueden ser por ejemplo: botones, listas desplazables, menús contextuales, etc., y que van a interactuar con el usuario.

Contenedores: Los cuales van a contener a los componentes, como ejemplo de estos están las ventanas, como contenedores más utilizados. Otros contenedores son los paneles, los cuadros de diálogo, etc.

Administradores de Diseño: Esto es un organizador de componentes dentro de la interfaz y que constituye una parte oculta para el usuario. Se debe definir como una parte que no varía dentro de las partes que conforman la interfaz.

Un administrador de diseño no se aprecia en cuanto a su funcionalidad sino hasta que se interactúa con él. De hecho al definir un administrador de diseño necesitaremos dos partes:

*Crear todos los componentes

*Llamar al método add() para incluir el componente respectivo.

Swing

El swing es un agregado de la biblioteca awt, y tiene la ventaja de crear una apariencia mejorada de la misma biblioteca, de tal forma que esta trate

de utilizar los mismos componentes que usa la interfaz gráfica de windows ó de solaris ó del sistema en operativo en particular.

Los componentes de Swing son subclases de JComponent y se utilizan para cumplir las siguientes funciones:

"Escuchadores" de Eventos: esto es la captación del cambio ocurrido en los componentes del applet particular.Estos cambios pueden interesar al usuario para ejecutar ciertas acciones sobre los objetos ya definidos ó simplemente que permitan la ejecución ó detención de algún método particular.

Lo anterior se logra gracias a la inclusión de métodos propios para escuchadores, ya que son un tipo de interfaz de objetos particular.

Sabemos que tipo de evento está oyendo el objeto "escuchador" mediante el nombre que se le asocia.

Timer : Timer es un componente que captura uno ó mas eventos después de un retardo (delay) específico. Esto se logra a través de métodos específicos que hacen que los applets, ó bien que las aplicaciones que tengan alguna clase de animación se puedan ejecutar, retardar ó parar.

La instalación de un timer involucra la creación de los objetos del mismo tipo y el registro de las acciones de escucha.

El método start, comienza la ejecución de la acción asociada. Así mismo es necesario el uso de un timer en modo de constructor para que un objeto llame a la instancia de timer. Por default la acción que se ejecuta cuando el timer comienza continuará ejecutándose hasta que se llame al método stop, el cual detiene la ejecución de las acciones.

Rigor de la Estructura en Java3D Aún cuando la estructura del grafo se ha planteado, puede suceder que no se coloquen los elementos en el órden específicado. Esto provocará que el grafo se vuelva "ilegal" en su construcción.

Pero no es suficiente con arreglar la estructura, además se deben vigilar las reglas de las relaciones padre e hijo, y las relaciones de referencia que se han enunciado, de lo contrario, el programa no visualiza nada y en cambio puede tener errores, que se denominan excepciones.

Para lograr arreglar de manera eficiente todos los problemas derivados del órden, estructura y relaciones tenemos la siguiente Jerarquía de clases de Alto Nivel en Java3D API:

javax.media.j3d
Virtual Universe
Locale
Physical Body
PhysicalUniverse
Screen3D
Canvas3D(extends awt.canvas)
ScreenGraphObject
    Node
        Group
        Leaf

NodeComponent
Transform3D
Alpha

Observemos que de la superclase ScreenGraphObject se derivan dos niveles. El primero Node específica un subnivel con otros dos niveles internos.

El primero (Node) define componentes para la orientación de los objeos virtuales y su posición , además de añadir componentes de forma, sonido y comportamiento de los objetos en el universo virtual.

El segundo es una superclase para la especificación de la geometría, apariencia, etc., sus componentes en realidad no son parte del grafo de escena pero son referenciados por este.

1.13.3 Ventajas y Desventajas

Ventajas

·Es una API de alto nivel y orientada a objetos para 3D

·No es necesario acceder a primitivas de bajo nivel para el rendering

·A través de la reutilizabilidad del código es capaz de optimizar las estructuras, optimizando así el redndimiento.

·Numerosos cargadores para archivos en otros formatos, por ejemplo VRML.

·Soporta dispositivos de desplegado de gráficas que son exóticos para realidad virtual.

Desventajas

·Como extensión standard de la API hay opción de instrumentarla pero no la exige, y esto puede ocasionar perdida de portabilidad.

·No hay en realidad Java3D en todas las plataformas, Sun sólo hizo dos plataformas: Solaris y Win32

·Oculta los procesos de rendering y a veces es necesario acceder a esta parte

·No hay documentación suficiente como en OpenGL u otras librerías para 3D

·El rendering es hecho mediante instrucciones que no son código nativo de java lo cual produce componentes heavyweight.

La descripción que se ha hecho de OpenGL y Java3D nos permite realizar un análisis para elegir el software que vamos a utilizar. Primeramente evaluamos las características de cada uno y las equiparamos contra el otro tomando en consideración los problemas que se van a resolver. Después damos una justificación del porque utilizaremos en este trabajo tal o cual software.

OpenGL se centra en el proceso de renderización de las gráficas basándose en primitivas de bajo nivel. Es una interfaz de hardware - software o sea interactúa entre ambos ocultando los detalles de bajo nivel. En realidad no se constituye como una API, es decir, no es formalmente una interfaz de programación. El proceso de desplegado como tal puede ser tedioso e incluso tener repercusiones en el aspecto de la programación, dado todo lo que tenemos que cuidar e inicializar.

Java3D conforma una API que tiene primitivas de alto nivel, en realidad hace interfaz y comunicaciones software - sofware. Al montar sobre el hardware la

máquina virtual no tiene necesidad de acceder a bajo nivel, aunque en ocasiones sea necesario hacerlo, y esto puede constituir una ventaja o una desventaja según el problema que se trate.

En OpenGL no es en realidad autodescriptible, es decir, no tiene la facilidad que tiene el paradigma orientado a objetos de leerse como si fuera un texto que nos va diciendo de manera paulatina lo que se va haciendo.

En el aspecto conceptual Java3D tiene la ventaja de irse construyendo e irnos dando una "guía" de lo que se va realizando pues tiene una cierta secuencia en un lenguaje más coloquial ó cotidiano.

En cuanto al rendimiento de una aplicación hecha en este lenguaje se ha comprobado que no tiene problemas e incluso supera en mucho a otras interfaces de programación y al mismo Java y por consecuencia a Java3D pues no tiene que colocar o instalar de primera instancia una maquina virtual sino que se ejecuta sobre el hardware directamente.

En Java3D se acarrean gastos en el desplegado debido a que el recolector de basura de Java, y a lo que se tarda en traducir los métodos y demás funciones y sentencias desde el programa fuente hacia las instrucciones de la máquina virtual. Aún cuando este problema ha sido superado en buena medida por la efectividad y rapidez del hardware, sigue siendo un problema inherente a Java.

El aspecto de portabilidad no esta completamente resuelto, en OpenGL debido a que se usa en diversas plataformas y esta apegado al hardware particular. Aun cuando las modificaciones del software son realmente pocas, puede resultar un overhead al momento de hacer una migración hacia otros sistemas.

La portabilidad en Java3D esta 100% garantizada, la máquina virtual no depende de ninguna plataforma en específico, sin embargo, no existen versiones de Java3D para todas las plataformas.

La flexibilidad que otorga OpenGL es limitada en el sentido de que muchas de las primitivas como tal necesitan arreglarse de manera conveniente para su uso. Las primitivas no contemplan necesidades particulares para un problema

En Java3D la flexibilidad de los métodos se puede realizar sin problema debido al instanciamiento de objetos de casi cualquier tipo y de tipo "indefinido" que puedan utilizarse en diversas aplicaciones.

OpenGL se ha constituido como un estándar de la graficación y es verdaderamente un pilar en cuanto a las gráficas por computadora. Esto quiere decir que su permanencia en el mercado de la visualización esta garantizada.

Java3D es más que nada un intento por estandarizar ciertas técnicas de programación, aun cuando su madurez ha sido alcanzada pues ya lleva cierto tiempo de estar en el mercado de la graficación.

Las primitivas de alto nivel y de interacción con el usuario que podrían ser actualmente tomadas como una necesidad obligada para la interfaz no están presentes. Para poder realizar una interfaz que actué efectivamente a través de multimedios, ventanas, etc, se pueden hacer usando otras interfaces derivadas del lenguaje.

1.14 Análisis Comparativo entre Java3D y OpenGL

Las comparaciones entre los dos software se dan en la siguiente tabla tomando como parámetros los siguientes:

+ Bajo
++ Medio
+++ Alto
—— Ninguno

	C	B	P	$Fac.\ Uso$	$Aplic.\ \mathrm{Pr}ob$	\$	$Tipo\ Prog.$	$Lib.\ Adic.$
$OpenGL$	+++	+++	++	++	+	0	T/OO	+++
$Java3D$	++	++	+++	+	+++	0	OO	+

	$S\tan d.$	$2D$	$3D$	$C.L$
$OpenGL$	+++	+++	++	+++
$Java3D$	——-	+	+++	——

Los criterios que se han discutido para llegar a esta tabla son los siguientes:

C (Costo): No se refiere al costo de comprar o adquisición del software determinado, sino a lo que cuesta su instalación, corrida compilación etc.

B (Beneficio): Beneficios del uso hacia el aprendizaje, no solo para la API particular sino para otras.

P. (Portabilidad): Transferencia a otras plataformas de hardware.

Fac. Uso: Existe Bibliografía (tutoriales, libros, cursos en red, etc,..) para un desarrollo efectivo

Aplic.Prob (Aplicabilidad al Problema): Que tanto se puede resolver el problema usando esta API determinada con relativa facilidad

$: Costo monetario del software

Tipo Prog (Tipo de Programación): Tradicional (T) u Orientada a Objetos (OO)

Lib Adic (Librerías Adicionales): Si se necesitan para propósitos específicos librerías o software adicional.

Stand (Standar): Si el software constituye ó no un estándar.

2D: Si se puede ó no construir figuras ó formas en 3D con facilidad considera ble.

3D: Si se puede ó no construir figuras ó formas en 2D con facilidad considera ble.

Compatibilidad Lenguajes: Si soporta variedad de lenguajes dentro de la programación ó unidades, librerías, código en sí de diversos lenguajes de programación.

La siguiente gráfica describe mejor los criterios y calificaciones para las API analizadas en el presente apendíce:

.

Dada la tabla anterior y equiparando los resultados, vemos que el software de elección que mejor se adapta a nuestras necesidades es Java3D.

1.15 ¿POR QUÉ JAVA3D?

Para finalizar este capítulo, diremos que las características de Java3D según lo visto han reflejado una facilidad mayor para aplicarse al problema del presente trabajo.

Se utiliza Java3D OpenGL para aprovechar la facilidad de uso de OpenGL, además de tratar de utilizar en lo posible métodos y clases que no sean específicas de esta versión sino más bien genericas a las dos versiones del mismo software

Si bien podría ser posible el aprendizaje y puesta a punto de programas en la programación tradicional, las ventajas de la orientación a objetos y en particular de Java se adaptan de mejor forma al objetivo principal a resolver.

Aún cuando el number crushing de Java es bajo, el nivel de programación para un prototipo como el que se está desarrollando se adapta de manera conveniente al lenguaje que hemos elegido.

Las desventajas en cuanto a la facilidad del uso de Java3D y el hecho de ser una API cuya evolución parece retardada y no muy extendida en el entorno actual, ha demostrado ser capaz de superarlas y ofrecer resultados aceptables con los criterios establecidos.

Apéndice 2
Manual del Usuario

El presente apendíce es un manual que tiene por objetivo el ayudar al usuario del simulador a manejarlo y entender las partes donde pueden surgir entre otras cosas dudas o errores en el momento de su manejo e interacción con el sistema.

2.16 PUESTA EN MARCHA

Para comenzar el simulador es preciso inicializar el símbolo del sistema, y escribir

java valvula

Esto inmediatamente nos lleva a una interfaz de usuario como sigue

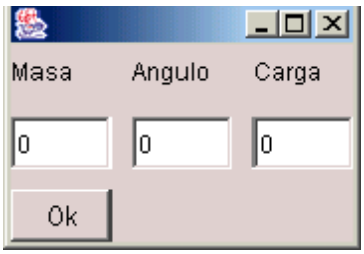

Figure 2.13: Interfaz con el Usuario

Los valores que se deben introducir representan los siguiente(para una mejor referencia vease capítulo 3):

Masa: Las unidades masa que se establecen de forma inicial para las bolas unidas a los brazos.

Angulo: El ángulo incial con el que la rotación de los brazos comienza

Carga: Es la fuerza del vapor que se simula en este caso y es la que se utiliza de forma inicial.

En caso de que los valores que se hayan introducido no sean correctos el simulador no ejecuta nada y muestra una pantalla como la que se muestra a continuación:

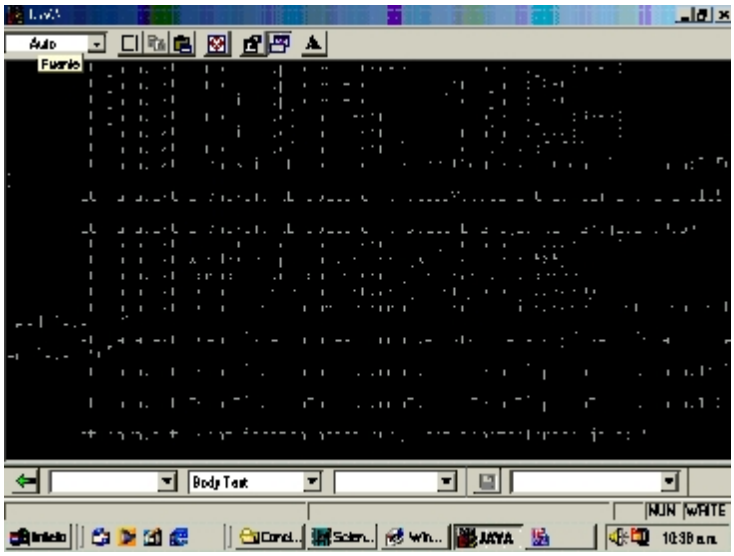

Figure 2.14: Error en la Introducción de Datos

Se espera entonces que se cambien los valores y se vuelva a presionar OK.

Una vez que se han cambiado por valores válidos, ó que los valores sean válidos de primera intención el simulador muestra en la figura 2.3.

Como ya se ha explicado (Cap.3) se estableció un código de colores para la figura anterior con Rojo, Azul y Amarillo.

El botón que se aprecia arriba permite la entrada de nueva cuenta a la interfaz inicial que nuevamente realiza otra ventana similar a la anterior.

Así el simulador permite establecer comparaciones entre los distintos estados, además de poder cambiar los valores para la simulación y observar su efecto.

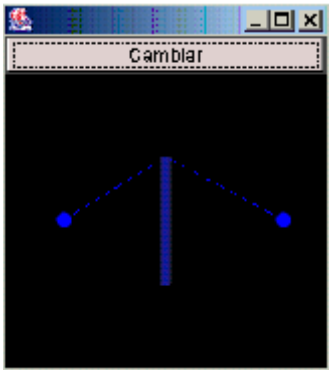

Figure 2.15: Ventana Principal del Simulador

BIBLIOGRAFIA

[1] Alanís J.D., Vallejo Rodríguez D., Modelado Basado en Física : Una Visión Global, Tesis de Licenciatura, BUAP, Dic, 2002

[2] Alanís J.D., Rodríguez Gómez G., Comprendiendo Modelos Através de Visualización, ENOAN, 2003

[3] Bermúdez Júarez B., Algebra Lineal Númerica, Curso de Maestría, Sep-Dic, 2001, FCC, BUAP

[4] Bouvier Dennis J.,Getting Started with the Java3D (Trade Mark) API, Sun Microsystems

[5] Boyce William E., Diprima Richard C., Ecuaciones Diferenciales y problemas con valores en la frontera, Ed. Limusa Noriega Editores

[6] Burden Richard L., Faires Douglas J, Análisis Númerico, Grupo Editorial Iberoaméricana, México 1981

[7] Celaya Borges Carlos, Notas del Curso de Identificación de Sistemas, Primavera 2002, Curso de Maestría, Benémerita Universidad Autónoma de Puebla.

[8] Eckel Bruce, Thinking in Java, Prentice Hall, United States of America, 2000

[9] Gosling James, McGilton Henry, The Java (Trade Mark) Language Environment, A white paper, Sun Microsystems, United States of America, May 1996

[10] Kopka Helmut, Daly Patrick W, A Guide to Latex Document Preparation for Beginners and Advanced Users, Addison-Wesley Publishing Compny United States of America, 1993.

[11] Lambert J.D., Numerical Methods for Ordinary Differential Systems, The Initial Value Problem, Ed. Wiley, West Sussex, England,1991.

[12] Lemay Laura, Rogers Cadenhead, Aprendiendo Java en 21 Días, Pearson Education, México 1999

[13] Ogata Katsui, Ingeniería de Control Moderno, Ed. Prentice Hall, México , 1998

[14] Pontriaguin, Ecuaciones Diferenciales Ordinarias, Ed. Aguilar, México 1967.

[15] Rademacher Paul, GLUI: A GLUT -Based User Interface Libary, June, 1999, Sillicon Graphics.

[16] Rangel Huerta Jose Alejandro, Notas del Curso de Simulación, Primavera 2002, Curso de Maestría, Benemérita Universidad Autonóma de Puebla.

[17] Reyes Rodolfo S., Pascualli,Peral Carrasco, Seminario de Latex, Benemérita Universidad Autónoma de Puebla., México 1990.

[18] Segal Mark, Akeley Kurt, The OpenGL Graphics System: A Specification, Editor Chris Frazier and Jhon Leeech, Sillicon Graphics, 1192-1999.

[19] Selman Daniel, Programming in Java3D, Ed. Manning, New York, US, 2002

[20] Sun Microsystems, The Java (Trade Mark) Language: An Overwiew, Copyright 1994

[21] Witkin A. and Baraff D, Physically Based Modelling, Carniege Mellon University, SIGGRAPH

[22] http://euler.berkeley.edu/~dma/res.html

[23] http://www.cs.cmu.edu/~baraff/sigcourse98

[24] http://mecfunnet.faii.etsii.upm.es/Xitami/webpages/reguwatt.html

[25] http://oldenginehouse.users.btopenworld.com/watt.htm

[26] http://umlab.ru/index/demo/20.htm